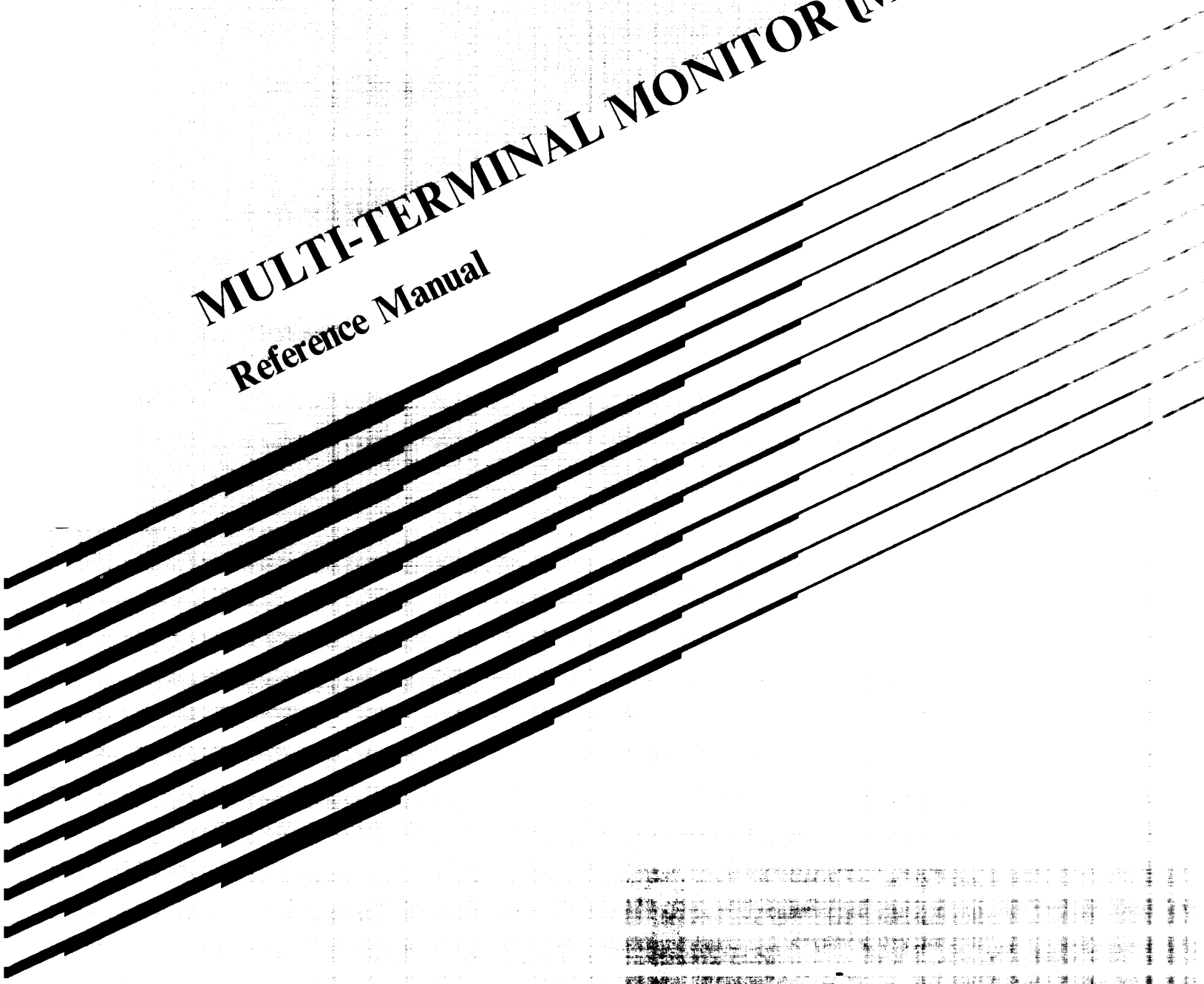
48-043 F00 R03

MULTI-TERMINAL MONITOR (MTM)

Reference Manual



PERKIN-ELMER

The information in this document is subject to change without notice and should not be construed as a commitment by The Perkin-Elmer Corporation. The Perkin-Elmer Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license, and it can be used or copied only in a manner permitted by that license. Any copy of the described software must include any copyright notice, trademarks, or other legends or credits of The Perkin-Elmer Corporation and/or its suppliers. Title to and ownership of the described software and any copies thereof shall remain in The Perkin-Elmer Corporation and/or its suppliers.

The licensed programs described herein may contain certain encryptions or other devices which may prevent or detect unauthorized use of the Licensed Software. Temporary use permitted by the terms of the License Agreement may require assistance from The Perkin-Elmer Corporation.

The Perkin-Elmer Corporation assumes no responsibility for the use or reliability of the software on equipment that is not supplied by Perkin-Elmer.

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place,
Oceanport, New Jersey 07757
Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

FIGURES

TABLES

PREFACE

The Perkin-Elmer Multi-Terminal Monitor (MTM) Reference 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 describes 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 F00 R03 reflects the changes required for the OS/32 features of R08.1. Information about the intelligent peripheral controller (IPC) tape driver has been added to the FRECORD command. The task related arithmetic fault, FLOATING POINT FUNCTION RANGE ERROR, has been included in Appendix D.

A reference to account 255 has been made in the System Account Numbers section. A new command, the LANGUAGE command has been added to the Program Development chapter. The program development procedure will now allow programming in the C language. The SIGNON command now includes a list of restricted userids. The SET CSS command has been included in Chapter 2. In the Command Substitution System chapter, examples have been provided for the CSS commands. The $GOTO command has a new option.

This manual is intended for use with the OS/32 R08.1 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

The multi-terminal monitor (MTM) permits several terminal users to share system resources. Each user perceives that a computer is at his disposal.

Concurrent access from on-line terminals is useful during application task development, because it reduces turnaround time and it 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. 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 can initiate and control 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 on-line 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 support both dedicated and dial-up telecommunication terminals.

1.3.1 Multi-Terminal Monitor (MTM) Devices

The following devices can be used at any local or remote installation:

- Model 550B Video Display Unit (VDU)
- Model 1100 VDU
- Model 1200 VDU
- Model 1250 VDU
- Model 1251 VDU
- Model 6100 VDU
- 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 catalogued 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 that has any or all of these new capabilities enabled is called a privileged user.

In addition to all standard MTM capabilities, privileged users can have extended MTM capabilities such as:

- displaying all jobs in the batch queue,
- moving between private accounts without knowing passwords (SET PRIVATE command),
- changing group account numbers (SET GROUP command),
- setting the priority of a subsequently loaded task via a private command substitution system (CSS) (PRIOR command),
- interfacing with a HASP protocol and returning to MTM control as desired ($HSP), and
- interfacing with a foreground task from an MTM terminal and returning to MTM control as desired ($FRGND).

For information on the specific privileges available through MTM and the procedures for enabling these privileges on an account basis, see 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 system can support up to 64 terminal users and batch streams in any combination.

1.4 MULTI-TERMINAL MONITOR (MTM) SUBTASK 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 interact with the task during its execution. Any user task (u-task) controlled through MTM is processed as a subtask of MTM. Both batch and interactive environments are available to MTM subtasks. For the sake of simplicity, subtasks will be referred to as tasks through the remainder of this manual. MTM provides interactive and batch user environments.

In an interactive environment, the user has the ability to interact with a task via 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 but all interactive tasks initiated by MTM terminal users are executed concurrently. During interactive task execution, a terminal user can direct a command to the MTM and can 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 and then the queue of submitted batch jobs awaiting execution is serviced by MTM. 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 has already completed execution, the returned status will be:

NO JOBS FOUND

1.4.1 Multi-Terminal Monitor (MTM) Terminal Modes

An active terminal is 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, a CSS procedure is not executing, and BUILD is not in effect. All nontask-related commands are accepted. An asterisk (*) is the default prompt displayed in this mode.

- Task-loaded mode: The task is loaded but it is not started, or the task is paused. An asterisk is the default prompt displayed in this mode.

- Task-executing mode: A task is started and executing. If started from a CSS, CSS mode is suspended. A hyphen (-) is the default prompt displayed in this mode. If an interactive task was started and data input is requested by the task, then a greater than character (>) is the default prompt displayed to the terminal user.

- CSS mode: A CSS procedure is being built or executed. A hyphen is the default prompt displayed in this mode. When a CSS terminates, the terminal returns to command mode and an asterisk prompt is output. When BUILD is in effect, a B> is the default prompt displayed.

- Foreground task mode: The terminal is transferred to the control of a foreground task. When the foreground task is completed, the terminal returns control to 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 quotation mark ("). All commands entered while in this mode are sent to the specified HASP task.

1.4.2 Interactive Task to Terminal Mode

When a task issues a supervisor call 1 (SVC1) input/output (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 information is not vital for tasks that perform SVC1 wait I/O, but users with tasks that issue SVC1 proceed I/O (read or write) should be aware that MTM suspends the task until the I/O is completed. MTM then posts an SVC1 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 task status word (TSW).

1.5 MULTI-TERMINAL MONITOR (MTM) IN A MULTIPROCESSOR ENVIRONMENT

The load leveling executive (LLE) feature of MTM is designed to optimize overall system performance in a multiprocessor environment on a Model 3200MPS System.

The Model 3200MPS System includes a central processing unit (CPU) and up to nine satellite processors. The satellite processors can be any combination of auxiliary processing units (APU's) and input/output processors (IOP's).

The load leveling executive (LLE) distributes the system load imposed by MTM user tasks between the CPU and APU's. MTM subtasks are queued for execution on either the CPU ready queue

or APU queue 0. The ready queue is serviced by the CPU alone, while queue 0 is serviced by assigned APUs and the CPU when its ready queue is empty.

Maximum processing power is available for the tasks on queue 0; however, all operating system services (SVC calls, trap and fault handling, etc.) are performed by the CPU. LLE optimizes MTM subtask traffic between the two queues by directing service-intensive tasks (e.g., I/O-intensive) to the CPU ready queue and computation-intensive tasks to queue 0. The LLE minimizes the traffic between the two queues (APU/CPU thrashing) by analyzing the run-time profile of a task at certain intervals to decide whether this particular task should be run on the CPU or on an APU to achieve efficient utilization of processors.

Load-leveling is performed only when the LLE is enabled and then only for those tasks, including executive tasks (e-tasks) and diagnostic tasks (d-tasks), directed to LPU 0, or having no SVC 6 control privileges (option LPU =n disabled).

NOTE

Tasks loaded with an LPU number other than zero and without SVC6 control privilege, are reset to LPU number zero (see Section 2.30).

1.6 LOADING A TASK

The dynamic nature of OS/32 memory management guarantees loading of any size task 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. 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 equal priority. Batch tasks can have user-assigned priorities.

1.7 MULTI-TERMINAL MONITOR (MTM) SPECIAL FEATURES

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

- CSS
- Help Facility
- Program development commands

- Spooling
- Security and access protection of disks
- Signon CSS

1.7.1 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.

1.7.2 The Help Facility

The Help Facility provides a user on-line access to documentation for MTM and program development commands. This information is obtained by entering the HELP command (see Section 2.28).

1.7.3 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.

1.7.4 Spooling

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

1.7.5 Security and Access Protection of Disks

Privately owned disks can be marked on as 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.

1.7.6 Signon Command Subsitution System (CSS)

MTM users can build a special CSS file, USERINIT.CSS, within their private accounts. This 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 on-line disks within the user's private account for the file USERINIT.CSS and automatically executes it. If no USERINIT.CSS file is found within the user's private account, the system account is searched.

1.8 CONVENTIONS

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

- Prompt
- Terminal
- Command
- Statement syntax
- File

1.8.1 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.

1.8.2 Terminal Conventions

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

TABLE 1-2 TERMINAL CONVENTIONS

OPERATION	CONVENTION
Deleting a line	Simultaneously depress the CTRL and character X keys for all terminals except TEC 445 VDU, which uses the hash mark (#). Basic communications support both # and CTRL X for line deletion for asynchronous remote device.
Deleting a character	Depress the BACKSPACE key. For terminals without a BACKSPACE key, simultaneously depress the CTRL and character H keys.
Ending an input line	Depress the carriage return (CR) key.
Communicating with MTM	While an interactive task is executing or when a BUILD command is active, depress the BREAK key and enter a command.

1.8.2.1 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.

1.8.3 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. When a command line is entered from a CSS, all the commands are skipped until a $TERMJOB is found. A character string preceded by an asterisk in column 1 is a comment.

1.8.4 File Conventions

A file is a collection of data stored on a direct access storage device (DASD). 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. 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.

1.8.4.1 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. To supply greater flexibility for file sharing, however, MTM supports the concept of group files.

1.8.4.2 Group Account Numbers

Authorized MTM terminal users are assigned 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; it 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.

1.8.4.3 System Account Numbers

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, but updating or deleting a system file can only be performed by the system operator.

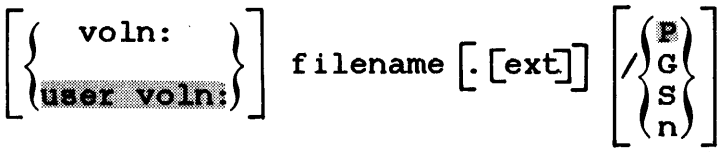
With respect to file ownership within an MTM environment, the system operator is viewed as more privileged than terminal users. The system operator can allocate files on any account in the system and can also change the account number of any file in the system to another account number. Similar to a terminal user, the system operator uses the RENAME command to change file ownership. MTM users can only access the restricted account 255 by signing on to account 255 with the correct password.

1.8.4.4 File Descriptors (fds)

Some commands require fds. An fd for MTM generally includes four fields:

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

Format:



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, 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 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 that 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 can 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	is for users that have the privilege to specify account numbers instead of account class designators (P, G and S) and can do so for some commands such as ASSIGN, LOAD, RENAME and CSS calls. Access is limited to shared read-only (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 and a default extension on volume ABC.
CARD:	is a device name.
A:B.C/G	is a group with filename B and 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.

1. Identify yourself to MTM by signing on to the system. Enter your userid, account number and a valid password.

```
SIGNON MAR,118,SWDOC
```

2. Establish the volume you will be working on by entering the VOLUME command and a valid volume name.

```
V M300
```

3. Load the editor task into memory by entering LOAD and the task name.

```
LOAD EDIT32
```

4. Initiate execution of the task by entering the START command.

5. Save all data appended to your file by entering the SAVE command.

```
.
.
.
S FILE1
```

6. Terminate execution of the task by entering the END command.

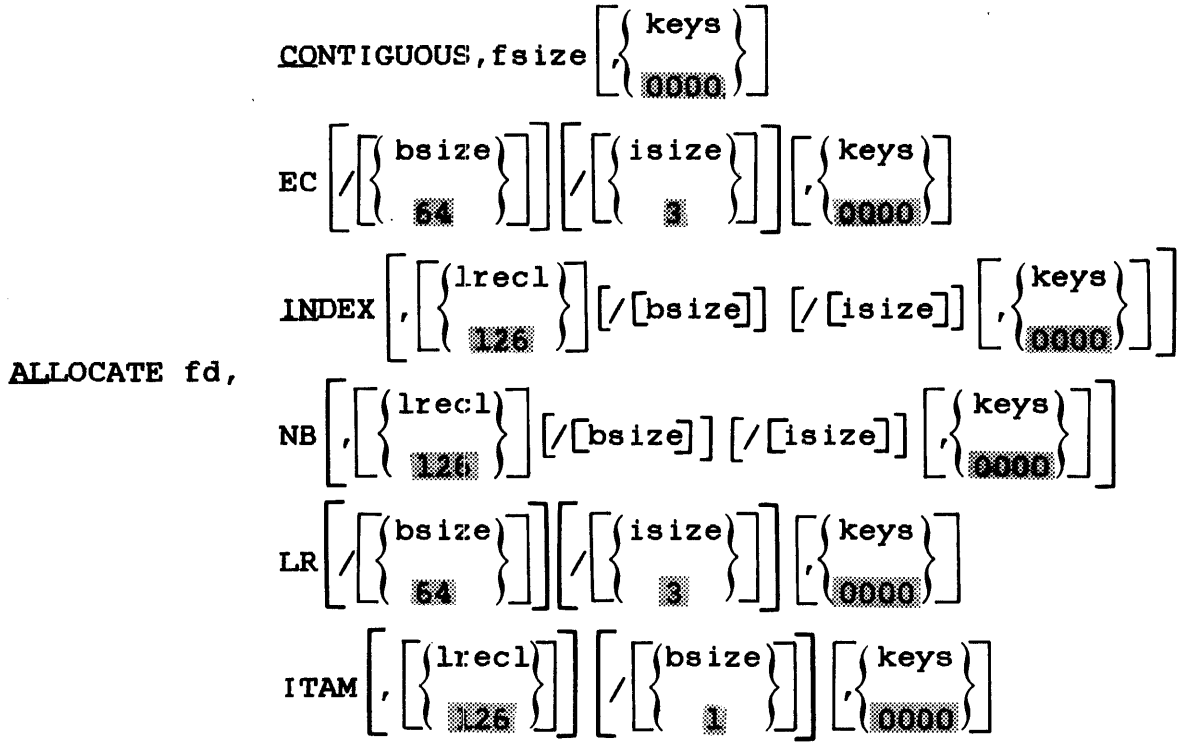
7. End the terminal session by signing off.

```
SIGNOF
```

2.2 ALLOCATE COMMAND

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

Format:



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 the file size required for contiguous files. It specifies 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.

keys — specify the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword; the left byte signifies the write key and the right byte signifies the read key. If this parameter is omitted, both keys default to zero.

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. If this parameter exceeds the maximum block size established for the system, the system maximum is used. If bsize is omitted, the default blocksize established at system generation (sysgen) or by the system operator is used. If no default value was established at sysgen or by the system operator, the default value is one sector for indexed files and 64 sectors for 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 indexed blocksize established at sysgen or by the system operator is used. If no default value was established at sysgen, 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. If the value specified for this parameter is greater than the system maximum, the system maximum is used.

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, NB indexed file or ITAM device. lrecl cannot exceed 65,535 bytes. The default value for lrecl 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.

LR — specifies a long record file. For LR files, the logical record length is specified by the data block size (bsize) parameter (i.e., the logical record length is the data block size).

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. At sysgen time, maximum block size parameters are established in the system and neither isize nor bsize can exceed these constants. If the user specifies numbers greater than the system maximum for bsize or isize, the maximum is used. No error message will be displayed in such cases.

The system maximums for bsize and isize can easily be determined by specifying numbers for them that are obviously too large. When the file is subsequently displayed (DISPLAY FILES command), the system maximums are shown rather than the specified numbers.

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.

For LR files, the absolute maximum data block size (logical record length) that can be specified is 65,535 (64K) sectors. This equals an absolute maximum logical record length of 16,776,960 (16M) bytes. In practice however, the actual maximum logical record length for any given system is limited by the amount of memory available for input/output (I/O) buffering.

The ALLOCATE command can be entered in command mode, task-loaded mode, task-executing mode and command substitution system (CSS) mode.

Examples:

Allocate on the default user volume a contiguous file named JANE.TSK whose total length is 64 sectors (16kb) with protection keys of 0:

```
AL JANE.TSK,CO,64
```

Allocate on volume M300 an indexed file named AJM.BLK with a logical record length of 132 bytes, a data block size of four sectors and the default isize established for the system. The protection keys default to 0. When this file is assigned, the system must have 2.25kb of system space available for buffers.

```
AL M300:AJM.BLK,IN,132/4
```

Allocate 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 THISFILE,IN,256/4/2
```

Allocate 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 the values established for the system and the protection keys default to 0.

```
AL VOL1:AJM.OBJ,IN,126
```

Allocate on volume VO1 an indexed file named AJM.OBJ with logical record length of 126 bytes. The data block size defaults to the value established for the system, the index block size is three sectors and the protection keys default to 0.

```
AL VO1:AJM.OBJ,IN,126//3
```

Allocate on volume SYS an extendable contiguous file named XFILE.DTA with a default data block size of 64 sectors and an index block size of three sectors. The file initially contains no records and has a record length of one sector (same as a contiguous file).

```
AL SYS:XFILE.DTA,EC
```

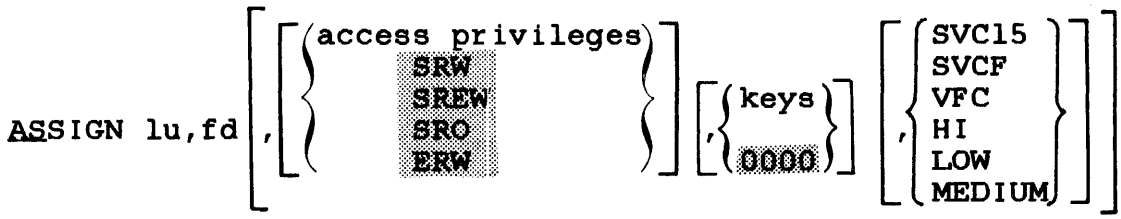
Allocate on the default volume a nonbuffered indexed file named YFILE.DAT with a logical record length of 240 bytes, a data block size of 250 sectors and an index block size of five sectors. The file initially contains no records.

```
AL YFILE.DAT,NB,240/250/5
```

2.3 ASSIGN COMMAND

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

Format:



Parameters:

lu	is a decimal number specifying the logical unit 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 user's private files. • ERW for devices (except the user's console. This has SRW).
keys	signify 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 supervisor call 15 (SVC15) access. SVCF is the hexadecimal equivalent of SVC15 and can also be specified. This option pertains to communications devices only.

If SVC15 access is specified, neither vertical forms control (VFC) 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 VFC 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 group coded recording (GCR) density rate of 6250 bits per inch (bpi).

LOW — indicates that the assigned magnetic tape will operate at the nonreturn to zero inverse (NRZI) density rate of 800 bpi.

MEDIUM — indicates that the assigned magnetic tape will operate at the phase-encoded (PE) density rate of 1600 bpi.

Functional Details:

If the access privileges and keys parameters are omitted and VFC, SVC15, HI, LOW or MEDIUM is 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	Shared read-only
ERO	Exclusive read-only
SWO	Shared write-only
EWO	Exclusive write-only
SRW	Shared read/write
SREW	Shared read, exclusive write
ERSW	Exclusive read, shared 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, for either read or 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 EWO 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	-	-	-	-	-	-	-	-

An asterisk (*) indicates compatible.
A hyphen (-) indicates incompatible.

The keys format is a 4-digit hexadecimal number. The left two digits signify the write protection key and the right two digits signify the read protection key. If omitted, the default is 0000. These keys are checked against the appropriate existing keys for the file or device. If the keys are invalid, the command is rejected. The keys associated with a file are specified at file allocation time. They may be changed by a REPROTECT command or through an SVC7 reprotect function call.

If the value of the keys is 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 to any user task (u-task) for that access mode, 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. If the specified lu is already assigned, the command is rejected. To reassign an lu for an active task, the lu must first be closed.

If the HI, LOW or MEDIUM parameter is not chosen when assigning to a mag tape device, the standard default density is used. The default is dependent upon the type of tape drive in use.

NOTE

If this parameter is used to select the density of the assigned mag tape, SVC15 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:

The following 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,FILE.DAT,EWO,99AA
```

The following assigns a disk file to lu2. VFC is in use. Access privileges and keys parameters are omitted along with their respective commas.

```
AS 2,TEST.JOB,VFC
```

The following assigns a disk file to lu2. VFC is in use. Access privileges and keys parameters are omitted, but positional commas are specified.

```
AS 2,TEST.JOB,,,VFC
```

The following assigns a disk file to lu2. VFC is in effect. The keys parameter, along with the positional comma, is omitted. The privilege is SRO.

```
AS 2,TEST.JOB,SRO,VFC
```

The following 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:,LOW
```

The following assigns a mag tape drive to lu2. The MEDIUM parameter indicates that the drive will operate at the phase-encoded density rate of 1600 bpi.

```
AS 2,MAG1:,SRW,MEDIUM
```

The following 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.

```
AS 2,MAG1:,,,HI
```

Invalid Examples:

The following is an invalid assignment because the positional comma belonging to the omitted access privileges parameter must be specified.

```
AS 2,TEST.JOB,00FF,VFC
```

The following is an invalid assignment because VFC and SVC15 access are mutually exclusive and cannot be specified in the same assignment.

```
AS 2,TEST.JOB,SRO,VFC,SVC15
```

The following is an invalid assignment because tape density and SVCF access are mutually exclusive and cannot be specified in the same ASSIGN command.

```
AS 2,MAG1:,SRW,LOW,SVCF
```

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 logical unit 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 Detail:

The BFILE command can be entered in task-loaded mode.

Examples:

The following example causes the device or file assigned to lu1 to backspace one filemark.

```
BF 1
```

The following example causes file AJM.OBJ, which is assigned to lu4 on volume M300, to backspace one filemark.

```
BF M300:AJM.OBJ,4
```

2.5 BIAS COMMAND

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

Format:

BIAS { address | * }

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 executive task (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 and task-executing modes.

Example:

The following example sets the bias to 100.

```
BI 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 Detail:

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 logical unit 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 Detail:

The BRECORD command can be entered in task-loaded mode.

Examples:

The following example causes the device or file assigned to lu1 to backspace one record.

```
BR 1
```

The following example causes the file AJM.OBJ assigned to lu4 on volume M300 to backspace one record.

```
BR M300:AJM.OBJ,4
```

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:

```
        { fd [,APPEND] }
BUILD   {              }
        { lu           }
   .
   .
   .
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 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, a blocksize of 1 and keys of 0000. If the specified fd already exists, that fd is deleted and a new fd is allocated.
APPEND	allows the user to append data to an existing fd. If the fd does not exist, it is allocated.
lu	is the logical unit 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.

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
```

| CANCEL |

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 PROCESSOR=hh:mm:ss:mmm

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

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:

$$\underline{\text{CLOSE}} \left\{ \begin{array}{c} lu_1 \left[,lu_2, \ldots, lu_n \right] \\ \underline{\text{ALL}} \end{array} \right\}$$

Parameters:

lu	is a decimal number specifying 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:

The following example closes logical units 1, 3 and 5 of the task.

```
CL 1,3,5
```

The following example closes all logical units of the task.

```
CLOSE A
```

| CONTINUE |

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 after the task is paused. Executing this command causes the terminal mode to be switched from task-loaded to task-executing mode.

2.12 DELETE COMMAND

The DELETE command deletes a direct access file.

Format:

DELETE fd_1 [,fd_2,...,fd_n]

Parameter:

fd	identifies the file(s) to be deleted.

Functional Details:

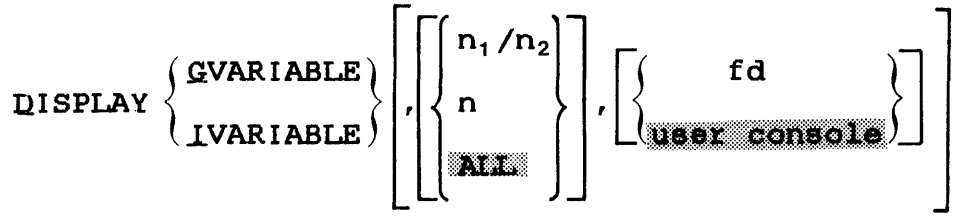
The file being deleted must not be currently assigned to an lu of any task. A file can be deleted only if its write and read protection keys are 0 (X'0000'). If the keys are nonzero, they can be changed using the REPROTECT command. Only private files can be deleted.

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

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:



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 the file descriptor of a file or device to which the display is to be output. The default for this parameter is the user's console.

Functional Details:

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

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

Examples:

The following example illustrates a way to display 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
```

The following example illustrates a way to display information about new global variable 3.

```
*DISPLAY GVARIABLE, 3

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

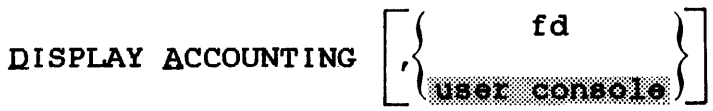
The following example illustrates a way to display all new global variables between 2 and 5.

```
*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:



Parameter:

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

Functional Details:

The general format of the system response to the DISPLAY ACCOUNTING command depends upon the configuration of the system.

In a uniprocessor configuration, the system response is:

```
USER   CPU TIME    hh:mm:ss:ms    ppp.p%
SVC    CPU TIME    hh:mm:ss.ms    ppp.p%
ROLL   CPU TIME    hh:mm:ss.ms    ppp.p%
PROCESSOR TIME     hh:mm:ss:ms    100.0%    ppp.p%
WAIT       TIME    hh:mm:ss.ms              ppp.p%
ROLL       TIME    hh:mm:ss.ms              ppp.p%
ELAPSED    TIME    hh:mm:ss.ms              100.0%
ROLLS              n
I/O                n
```

For a Model 3200MPS System (multiprocessor configuration), the system response is:

```
USER  CPU TIME    hh:mm:ss:ms    ppp.p%
USER  APU TIME    hh:mm:ss.ms    ppp.p%
SVC   CPU TIME    hh:mm:ss.ms    ppp.p%
ROLL  CPU TIME    hh:mm:ss.ms    ppp.p%
PROCESSOR TIME    hh:mm:ss.ms    100.0%
                                            ppp.p%
WAIT      TIME    hh:mm:ss.ms               ppp.p%
ROLL      TIME    hh:mm:ss.ms               ppp.p%
ELAPSED   TIME    hh:mm:ss                  100.0%
ROLLS             n
I/O               n
```

The indicated percentages for USER CPU and USER APU, SVC CPU and ROLL CPU time are calculated as percentages of PROCESSOR TIME. The indicated percentages for PROCESSOR, WAIT and ROLL times are calculated as percentages of ELAPSED TIME. For I/O and ROLLS, n indicates the number of each that has occurred. All information displayed in response to the DISPLAY ACCOUNTING command pertains to the current, or most recently executed, task. If no task has been loaded or executed during the MTM session, this command is meaningless and the system responds to it with a SEQ-ERR message.

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, task-executing mode and CSS mode.

Examples:

The following is an example for a uniprocessor system:

```
*DISPLAY ACCOUNTING
USER  CPU TIME       9:10.344   84.4%
SVC   CPU TIME       1:41.975   15.6%
ROLL  CPU TIME          0.000    0.0%
PROCESSOR TIME      10:52.319  100.0%   85.9%
WAIT      TIME       1:46.764           14.1%
ROLL      TIME          0.000            0.0%
ELAPSED   TIME         12.39   100.0%
ROLLS             0
I/O               176759
```

The following example is for a Model 3200MPS System:

```
*D A
USER   CPU TIME          0.837    16.7%
USER   APU TIME          1.329    26.6%
SVC    CPU TIME          2.834    56.7%
ROLL   CPU TIME          0.000     0.0%
PROCESSOR TIME           5.000   100.0%     8.9%
WAIT      TIME          51.224              91.1%
ROLL      TIME           0.000               0.0%
ELAPSED   TIME          56                 100.0%
ROLLS              0
I/O             1650
```

2.15 DISPLAY DEVICES COMMAND

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

Format:

DISPLAY DEVICES [,{ fd / user console }]

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 Detail:

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

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 off-line. If on-line, 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 an fd 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 (DPFP) registers associated with the loaded task.

Format:

DISPLAY DFLOAT [,{fd / user console}]

Parameter:

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

Functional Details:

This command suspends the current task, displays the contents of the DPFP registers and releases the task. The task is only released if it has not already been suspended by the user.

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 modes.

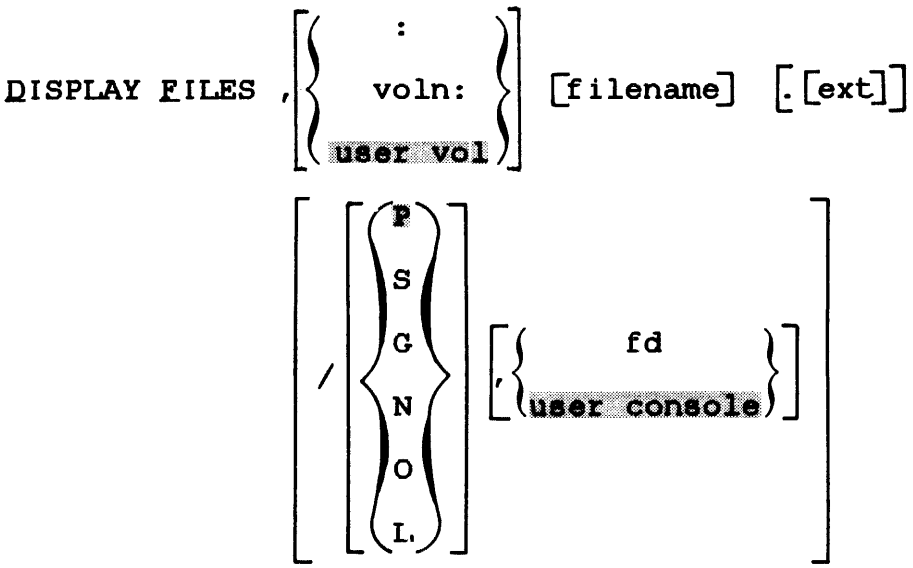
Example:

```
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:



NOTE

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 the volume in which they reside. Entering the colon with part of a filename limits the file search to filenames with the specified characters.
voln:	is the 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 the 1- to 8-character name of a file. The first character must be alphabetic, the remaining alphanumeric.
ext	is the 1- to 3-character extension to a filename.
P	indicates that information is requested for a private file.
S	indicates that information is requested for a system file; the default is private file only.
G	indicates that information is requested for a group file; the default is private file 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 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 hyphen or following the hyphen are displayed and are 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 a 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 the volume on which they reside.

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

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

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

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

```
D F,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:-.-
```

```
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, task-loaded and task-executing modes.

NOTE

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

Examples:

The following example displays to the user terminal all files with the user's account number on the default user volume.

```
D F
```

The following example displays file CAL32.TSK in the private, group and system accounts.

```
D F,CAL32.TSK/-
```

The following example displays all files in the private, group and system accounts on the default user volume.

```
D F,-/-
```

The following example displays to the device MAG1 all files with the user's account number on the default user volume.

```
D F,,MAG1:
```

The following example displays to the user's terminal all files with the user's account number on volume M300.

```
D F,M300:
```

The following example displays to the user's terminal all files on volume M300 with first character A and extension TSK in the user's account number.

```
D F,M300:A-.TSK
```

The following example displays all files on the default user volume in the user's account number with a blank extension, regardless of filename. The display is routed to device PR1:.

```
D F,-.,PR1:
```

The following example 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,CAL**1-.-
```

The following example 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 on-line disk volume whose name starts with the letter M.

```
D F,M-:TASK.5*
```

The following example displays to the user's terminal the files named TASK with any extension. A separate display of these files is done for each on-line disk volume in the system.

```
D F,-:TASK.-
```

The following example displays to the user's terminal all files that start with the four characters EDIT on all volumes and 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,-:EDIT-/*
```

The following example displays to the user's terminal all files in the user's private and group account on the default user's volume.

```
D F,-/N
```

2.18 DISPLAY FLOAT COMMAND

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

Format:

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.

This command suspends the current task, displays the contents of the SPFP registers and releases the task. The task is only released if it has not already been suspended by the user.

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:

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 through file are displayed. The current record number and percentage through file are displayed only for files. For nonprivileged users, the file class is shown; for privileged users the account number is shown.

This command suspends the current task, displays the task's lu assignments and releases the task. The task is only released if it has not already been suspended by the user.

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

Example:

```
DISP LU
 LU  FILE/DEVICE                      RECORD   THRU
  1  M301:DEMO.IN/P,SREW                1500   100.0%
  2  M301:SORT.OUT/P,SREW               1246    83.7%
  3  CON:,SRW
  4  M67C:&2987406.001/P,SREW             14    58.3%
  5  M67C:&2987407.001/P,SREW              2    66.6%
  9  MTM:SRTMRGII.OVY/S,SR0              141    46.7%
```

2.20 DISPLAY PARAMETERS COMMAND

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

Format:

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:

This command suspends the current task, displays the task's parameters and releases the task. The task is released only if it has not already been suspended by the user.

The suspend status (STAT = 100) is masked for tasks internally suspended for commands such as DISPLAY FLOAT, DISPLAY REGISTERS, etc.

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 task status word (TSW)
CLOC	xxxxx	Current location
STAT	xxxxx	Task wait status
TOPT	xxxxxxx	Task options
USSP	xxxxx	Current used system space
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 are 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
7	0100 0000	0 = No extended SVC1 parameter blocks used (excludes communications I/O) 1 = Extended SVC1 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
11	0010 0000	0 = Not in system group 1 = In system group
12	0008 0000	0 = No console I/O intercept 1 = Console I/O intercept enable
13	0004 0000	0 = Universal status reports disabled 1 = Universal status reports enabled
14	0002 0000	0 = Allow e-task load 1 = Prevent e-task load
15	0001 0000	0 = Queued I/Os not purged on error 1 = Queued I/Os purged on error
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

TABLE 2-3 TASK OPTION BIT DEFINITIONS (Continued)

BIT	MASK	MEANING
20	0000 0800	0 = SVC6 control call 1 = Prevent SVC6 control call
21	0000 0400	0 = SVC6 communication call 1 = Prevent SVC6 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
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
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 and task-executing modes.

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           140
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:

DISPLAY REGISTERS [,{ fd / user console }]

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:

This command suspends the current task, displays the contents of the task's general purpose registers and releases the task. The task is released only if it has not already been suspended by the user.

The DISPLAY REGISTERS command can be entered in task-loaded and task-executing modes. For u-tasks, the contents of each register will be 0 until the task has started. For diagnostic tasks (d-tasks) and e-tasks, register F contains the absolute UBOT address 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:

DISPLAY TIME [,{ fd / user console }]

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, task-loaded and task-executing modes.

2.23 DISPLAY USERS COMMAND

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

Format:

Parameter:

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

Functional Detail:

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

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
```

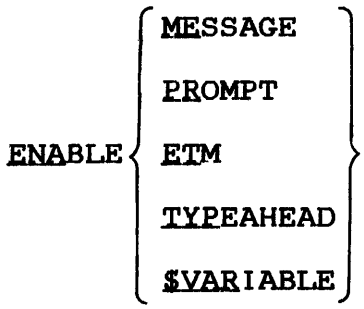
Where:

>	denotes a nonprivileged MTM user.
@	denotes a privileged MTM user.
MTM-MODE	indicates standard MTM usage.
ECM-MODE	indicates environmental control monitor mode.
$	indicates 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:



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.
TYPEAHEAD	enables input without intervening reads by MTM. This allows the user to continually type without waiting for a prompt (i.e., no connected read request).
$VARIABLE	enables variable processing of local and global variables on a per user basis.

Functional Details:

The ENABLE command does not affect messages originating from the system operator.

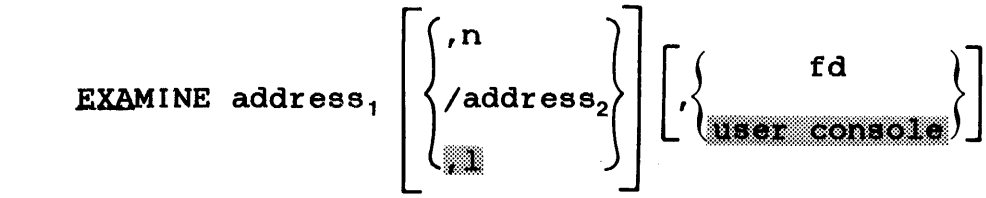
Local and global variable support for all MTM users is included in the MTM sysgen option SGN.VAR.

TYPEAHEAD allows for faster typing without losing characters or having to wait for the prompt to return. Any characters typed during this time are placed on a typeahead character queue and then passed to the editor for processing on the following read request. This command is only effective on terminals using the bidirectional input/output control (BIOC) device driver.

2.25 EXAMINE COMMAND

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

Format:



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:

This command suspends the current task, displays the contents of the specified location(s) and releases the task. The task is released only if it has not already been suspended by the user.

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 and task-executing modes.

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.

Examples:

Examine 24 (decimal) halfwords beginning at relative address 2A0:

```
*EXAMINE 2A0,24
0002A0  FFFF FFFF 0000 0000  C08B 44F7 0000 0000  * ..........D..... *
0002B0  40A5 FAFA 0000 0000  4110 0000 41A0 0000  * @.......A...A... *
0002C0  2E50 4155 5345 0000  CB10 0064 D0B1 0050  * .PAUSE.....d...P *
```

Examine from relative address 2A0 to relative address 2CF:

```
*BIAS 2A0
*EXA 0/2F
0002A0  FFFF FFFF 0000 0000  C08B 44F7 0000 0000  * ..........D..... *
0002B0  40A5 FAFA 0000 0000  4110 0000 41A0 0000  * @.......A...A... *
0002C0  2E50 4155 5345 0000  CB10 0064 D081 0050  * .PAUSE.....d...P *
```

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 logical unit 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 Detail:

The FFILE command can be entered in task-loaded mode.

Examples:

The following example causes the file or device to lul to forward space one filemark.

```
FF 1
```

The following example causes the file AJM.OBJ on volume M300 assigned to lu4 to forward space one filemark.

```
FF M300:AJM.OBJ,4
```

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 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 Detail:

The FRECORD command can be entered in task-loaded mode.

For the streaming tape driver, the FRECORD command moves the streaming tape forward one 512-byte block.

Examples:

The following example causes the device or file assigned to lu1 to forward space one record.

```
FR 1
```

The following example causes file AJM.OBJ on volume M300 assigned to lu4 to forward space one record.

```
FR M300:AJM.OBJ,4
```

2.28 HELP COMMAND

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

Format:

HELP [{ mnemonic }]
     [{ *   }]

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.

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
<mnemonic>.
```

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 command initializes all data on a contiguous file to zero.

Format:

INIT fd [,{segsize increment / 1}]

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:

The following example initializes the file DATA.FIL.

```
INIT DATA.FIL
```

The following example initializes the file DATA2.FIL using a 50kb buffer.

```
INIT DATA2.FIL,50
```

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 from which the task is being loaded.
segsize increment	specifies amount of memory in kilobytes (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, use of this parameter will generate an error message.

Functional Details:

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

For a Model 3200MPS System, a user without SVC6 control privilege cannot load a task to a nonzero logical processing unit (LPU) number. If a nonprivileged user loads a task and then specifies the OPTION LPU command with an argument other than zero, the following message is displayed:

```
*LPU RESET TO 0
```

Users with SVC6 control privilege can load tasks to LPU numbers other than 0 regardless of the LLE.

Examples:

The following example loads the task from file VOL:CAL.TSK.

```
LVOL:CAL
```

The following example loads the task from the paper tape reader punch device.

```
L PTRP:
```

2.31 LOG COMMAND

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

Formats:

```
LOG fd [,[{NOCOPY}]][,{ n  }]
       [{COPY  }]    {15 }

SET LOG [fd][,[{NOCOPY}]][,{ n  }]
              [{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 to 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 and the SET LOG commands are the same. The command can be entered either way and both command formats perform the same function.

Checkpointing can be done on any type of file. Since indexed files are buffered, checkpointing may be useful 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 a contiguous file is meaningless; no operation is performed.

The LOG command can be entered in command, task-loaded and task-executing modes.

Example:

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

2.32 MESSAGE COMMAND

The MESSAGE command sends a message to a specified user.

Format:

MESSAGE { userid / .OPERATOR } message

Parameters:

userid	is the name of the user to whom the message is being sent. 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, task-loaded and task-executing modes.

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:

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

Parameters:

address	is the halfword boundary address at which the modification of the contents of memory is to begin.
data	is a data field consisting of zero to four hexadecimal digits that represents 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 memory, beginning at the halfword location specified by address (modified by any previous BIAS command), to be replaced with the specified data. The task is suspended, the contents of the specified location are modified, and the task is released if it is not already suspended by the user. The modify address must be aligned on a halfword boundary. The number of halfwords that can be modified by a single command is limited only by the command buffer size established at MTM sysgen.

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

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

Examples:

In the following example each command is used to modify a single halfword.

```
*EXA 2C0,8
0002C0  2E50 4155 5345 0000  CB10 0064 D0B1 0050  * .PAUSE.....d...P *
*
*MODIFY 2C0,2B43
*MODI 2C2,4841
*MOD 2C4,4E47
*MO 2C6,452B
*
*EXA 2C0,8
0002C0  2B43 4841 4E47 452B  CB10 0064 D0B1 0050  * *CHANGE+...d...P *
```

The command can also be used to modify several halfwords. The number of halfwords that can be modified by a single command is limited by the size of the command buffer.

```
*BIAS 2C0
*MO 0,2E50,4155,5345,0
*
*EXA 2C0,8
0002C0  2E50 4155 5345 0000  CB10 0064 D0B1 0050  * .PAUSE.....d...P *
```

2.34 OPTIONS COMMAND

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

Format:

```
OPTIONS [ AFPAUSE    ][,{ SVCPAUSE    }] [,NONRESIDENT][[,LPU=[n]] [,NLPU]]
        { AFCONTINUE }[,{ SVCCONTINUE }]
```

Parameters:

AFPAUSE	specifies that the task is to pause after any arithmetic fault (AF).
AFCONTINUE	specifies that if the arithmetic fault trap enable bit is set, a trap is taken. If the bit is not set, the task continues after an AF occurs and a message is sent to the log device.
SVCPAUSE	specifies that SVC6 is treated as an illegal SVC (applies to background tasks only). If an SVC6 is executed within a background task, the task is paused.
SVCCONTINUE	specifies that SVC6 is treated as a NO-OP (applies to background tasks only). If an SVC6 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.
LPU=n	sets the task as an LPU-directed task. The specific LPU can be specified by n.
NLPU	sets the task as a central processing unit (CPU)-directed task (invalid if the APUONLY option is set by Link).

Functional Details:

For a Model 3200MPS System with load-leveling executive (LLE) enabled, specification of an LPU number other than zero by a user without SVC6 control privilege causes the following message to be displayed:

```
PARM-ERR POS=LPU=n
```

The OPTIONS command can be entered in task-loaded mode.

The LPU and NLPU parameters affect the LPU-directed task status. The NLPU parameter resets the LPU-directed task status (i.e., task is CPU-directed). If the combination "LPU=n, NLPU" is entered, the task's LPU number is set to "n" and the task is CPU-directed. This assigned LPU number has no effect until the task is changed to LPU-directed.

Example:

In the following example, task options are set to allow the task to take a trap or log a message depending on the setting of the arithmetic fault trap enable bit. If the task is a background task, any SVC6 will be treated as a NO-OP.

```
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 match the user's current account password exactly.
new password	specifies the new account password. This password replaces the current password in the AUF. 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 Detail:

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

| PAUSE |

2.36 PAUSE COMMAND

The PAUSE command pauses the currently running task.

Format:

PAUSE

Functional Details:

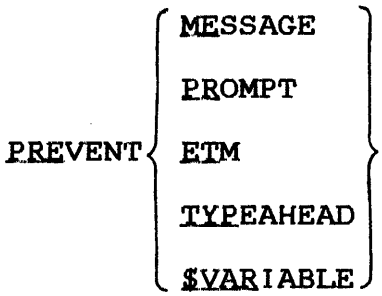
Any I/O proceed ongoing at the time the task is paused is allowed to go to completion. This command is rejected if the task is dormant or paused at the time PAUSE is entered.

The PAUSE command can be entered in 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:



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.
TYPEAHEAD	MTM resumes a normal command read ("*" command read).
$VARIABLE	disables variable processing on a per user basis.

Functional Details:

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

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 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, this command results in an error.

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

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, this command will result in an error.

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

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:

$$\$RELEASE \left\{ \begin{matrix} GVARIABLE \\ IVARIABLE \end{matrix} \right\} \left[\left\{ \begin{matrix} n_1/n_2 \\ n_1\,[,\ldots,n_n] \\ ALL \end{matrix} \right\} \right]$$

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. 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 can be entered in command, task-loaded, task-executing and CSS modes.

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; no error message is generated.

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

Examples:

The following example releases all new global variables from 1 through 5.

```
$RELEASE GVARIABLE,1/5
```

The following example releases the new internal variables numbered 16, 19, 18 and 25.

```
$RELEASE IVARIABLE,16,19,18,25
```

The following example releases all new internal variables.

```
$RELEASE IVARIABLE,ALL
```

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 new file descriptor of the renamed file.

Functional Details:

The volume ID field of 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, task-loaded and task-executing modes.

The user can only rename private files.

Example:

The following example renames file AJM.CUR to AJM.NEW on volume VOL.

```
REN VOL:AJM.CUR,AJM.NEW
```

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, task-loaded and task-executing modes.

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 Detail:

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

Examples:

The following example causes the file or device assigned to lu1 to be rewound.

```
REW 1
```

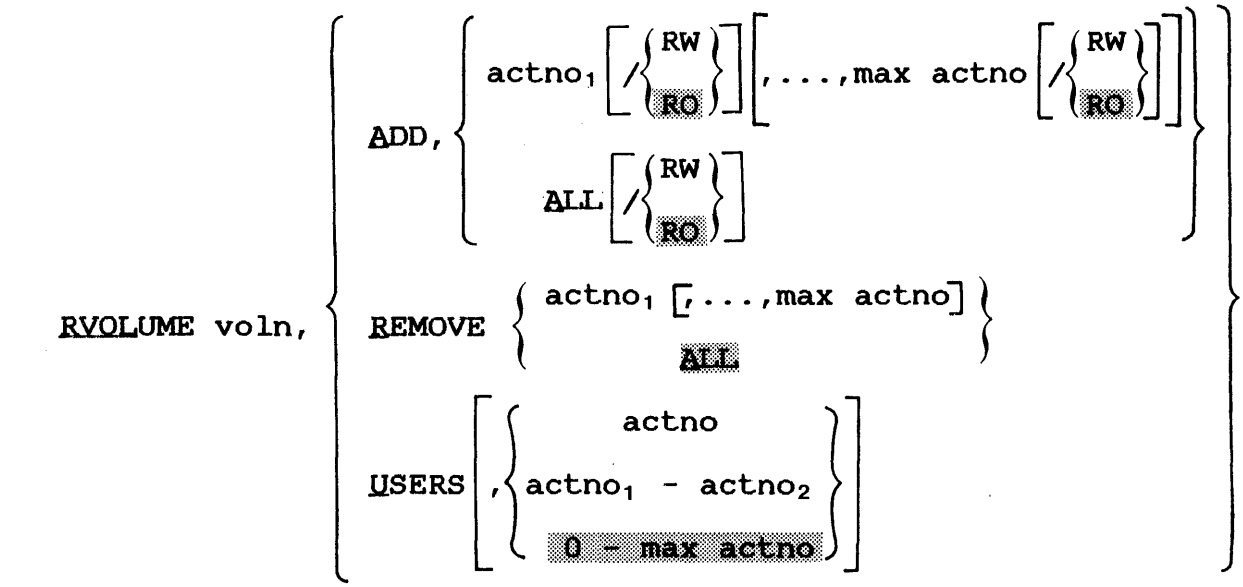
The following example causes file AJM.OBJ assigned to lu4 on volume M300 to be rewound.

```
REW M300:AJM.OBJ,4
```

2.44 RVOLUME COMMAND

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

Format:



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 to 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 to the maximum account number allowed on the system (limit 65,635) can access 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 limit was 1023
ACCT-ERR  POS=87-1200
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 SVC6 send message. Following standard SVC6 procedures, the message consists of an 8-byte task ID 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 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. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on SVC6.

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 CSS COMMAND

The SET CSS command enables a user to specify an account to be searched just prior to searching the system account, when looking for a CSS. The CSS account is designed to be used in conjunction with the CSS option of the VOLUME command.

Format:

SET CSS { P | G | S | n }

Parameters:

P	Private Account
G	Group Account
S	System Account
n	a decimal number specifying any account number except 255. In order to enter this form of the SET CSS command a user must have SET GROUP privileges.

Functional Details:

This command, in conjunction with the CSS option of the VOLUME command, allows a user to establish a volume/account where his CSSs are found.

MTM signon sets the CSS account to the signon account.

The SET CSS command can be entered in command, task-loaded, task-executing and CSS modes.

2.47 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

Parameter:

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 to the maximum account number set in AUF (cannot exceed 65,535).

Functional Details:

The SET GROUP command can be entered in command, task-loaded, task-executing or CSS modes. 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 can change group numbers as often as desired.

2.48 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, 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, task-loaded, task-executing and command modes.

2.49 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 to access, 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 AUF (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, task-loaded, task-executing and CSS modes. 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 displays 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 displays all files in account 210. The user can alter private accounts as often as 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.

2.50 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          PROCESSOR=hh:mm:ss:mmm      TSK-ELAPSED=hh:mm:ss
SIGNON LEFT=hh:mm:ss           PROCESSOR LEFT=hh:mm:ss
TIME OFF=mm/dd/yy  hh:mm:ss
```

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

SIGNON LEFT and PROCESSOR LEFT messages are not displayed if no limits have been set for the account in the AUF.

2.51 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[:]}

[,{PROCESSORTIME}=maxtime]
  {   CPUTIME   }

[,classid=iocount1 ,...,classid=iocount32]
```

Parameters:

userid	is a 1- to 8-character alphanumeric string specifying the terminal user's identification. This parameter must not match any active userid or foreground task name. See the functional details for a list of restricted userids.
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 on-line disks for the signon CSS procedure USERINIT.CSS/P. If a USERINIT.CSS procedure is not found on the private account, the system account on the system volume is searched. 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 USERINIT.CSS to be executed (if found).

PROCESSORTIME=
CPUTIME=
is a decimal number specifying the maximum session is limited. Processor time in a Model 3200MPS System is CPU+APU time, whereas processor time in a uniprocessor system is only CPU time. 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 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:

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

The following is a list of restricted userids:

HELP, STAT, DLINK, TIP, RMT, HASP

The userid parameter can not start with or consist solely of these combinations of characters.

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. There are several methods used to establish the environment that a user will be placed into upon signon. The method that offers the most standardization is to establish a system USERINIT.CSS file. This file must reside on the system volume on the system account.

Any (or every) account may have a USERINIT.CSS file specific to the account. As mentioned, a search for a USERINIT.CSS/P file (for the account being signed on to) is conducted prior to the search of the system files. By establishing a private USERINIT.CSS file, an environment peculiar to the specified account can be created through the SIGNON command.

Individual users within an account may wish to be placed into an environment other than the standard system or account environment. This is accomplished through the creation of a file containing the commands necessary to establish the desired environment. The user then specifies the fd for this file through the ENVIRONMENT parameter.

Parameters can be entered as specified above or an interactive signon procedure can be utilized. By entering only the SIGNON command, the user will be prompted to enter each parameter as demonstrated in the following example:

```
SIGNON
>USERID:
>ACCOUNT: <no-echo>
>PASSWORD: <no-echo>
```

The purpose of this procedure is mainly for security. When the account and password are entered, MTM does a no-echo read which is only effective on a terminal configured with the BIOC device driver. A privileged user is additionally prompted with the ENVIRONMENT= parameter. Invalid entries to this signon procedure will return the following error message:

```
-INVALID PARAMETER -- SIGNON REQUIRED
```

The user is again prompted for entry with no limit on the number of attempts to signon.

Examples:

In the following example, either the account or system USERINIT.CSS file can be used to create the signon environment:

```
SIGNON ME,12,PSWRD
```

In the following example, the desired environment is to be created after signon:

```
SIGNON DAVE,118,SWDOC,ENV=NULL
```

In the following example, the environment is to be created by execution of the commands contained in the file specified with the ENVIRONMENT= parameter:

```
S BETTYSUE,119,DIFRNT,ENV=M301:EOU119.ENV
```

The following listing presents an example of a file that might be used for environment creation:

```
 * EOU119.ENV - this file establishes an alternate EASE OF USE
 *              (EOU) environment for account 119.  It is used in
 *              conjunction with the ENVIRONMENT parameter of the
 *              the SIGNON command.
 *
 PREVENT PROMPT;$WR
 $RELEASE GVAR,ALL                  ;* clear new global variabl
 * Define default system devices:
   $DEFINE G10,SSYSLST,ST(P)        ;* list
   $DEFINE G11,SSYSIN,ST(CON:)      ;* input
   $DEFINE G12,SSYSOUT,ST(PR1:)     ;* output
   $DEFINE G13,SSYSPRT,ST(CON:)     ;* print
   $DEFINE G14,SSYSCOM,ST(CON:)     ;* command input
   $DEFINE G15,SSYSMSG,ST(CON:)     ;* message output
   $DEFINE G2,MTMUSR,CUR(USER)      ;* current userid
 *
 FORT                               ;* set FORTRAN (D compiler)
                                    ;* as current environment.
 XALLOCATE @*G2.LOG,IN,80           ;* establish and set a log
 LOG @*G2.LOG,COPY                  ;* file for the session.
 *
 *Display some useful info:
  DISPLAY USERS                     ;: show co-users.
  $WRITE
  DISPLAY TIME                      ;* show date and time.
  $WRITE
  VOLUME M301;VOLUME                ;* set user volume & display.
  $WR;INQUIRE                       ;* any batch jobs?
  $WR
  $WR WELCOME ABOARD - @*G2         ;* say hi.
  ENABLE PROMPT
$EXIT
```

2.52 SPOOLFILE COMMAND

This command is valid only on systems that 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 the specified lu of the currently selected task. This command makes all spooling options available at terminal or CSS level.

Format:

```
SPOOLFILE lu&lu1,pseudo-dev,FORM=formname [,{VFC   }]
                                           {IMAGE}

[,{NOIMAGE}] [,{CHECKPOINT  }] [,COPIES=n]
  {NOVFC  }  {NOCHECKPOINT}

[,{HOLD   }] [,{DELETE  }] [,PRIORITY=p]
  {RELEASE}  {NODELETE}

[,BLOCK=bsize/isize]
```

Parameters:

lu	is a decimal number specifying the logical unit to which the pseudo device is to be assigned.
lu_1	indicates that lu is to be assigned to the same spool file as lu_1. lu_1 must be the first lu assigned to the spool file.
pseudo-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 IMAGE	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 VFC character. If IMAGE is specified, there is no VFC for the device assigned to the specified lu.
NOIMAGE NOVFC	turns the IMAGE option or VCF option off for the assigned lu. NOVFC is the default option.
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.
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.
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.
BLOCK	specifies the index and/or data block size.
bsize	is a decimal number specifying the physical block size in 256-byte sectors, to be used for buffering and debuffering operations involving the file. If this parameter is not specified, the default data block size established at sysgen or by the system operator is used. If this value exceeds the maximum block size established for the system, the maximum set at sysgen is used.

isize — is a decimal number specifying the index block size in 256-byte sectors. If this parameter is not specified, the default index block size established at sysgen or by the system operator is used. Index size cannot exceed the maximum index block size established for the system. If a value greater than the maximum established for the system is specified, the maximum is used.

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:

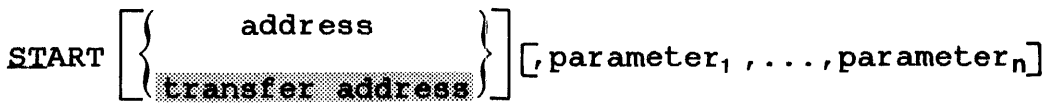
The following example causes a spool file to be allocated for pseudo device PD1: and assigns that file to lu4 of the current task. VFC has been specified for the specified lu and the DELETE option has been selected, which means the file will be deleted after output. The default physical and index block sizes set at sysgen will be used.

```
SPOOLFILE 4,PD1:,VFC,DELETE
```

2.53 START COMMAND

The START command initiates execution of a dormant task.

Format:



Parameters:

address	specifies the address at which task execution is to begin. For u-tasks, this is not a physical address but an address within the task's own program. For e-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 (CR) is stored at UTOP.

Functional Detail:

The START command can be entered in task-loaded mode.

Examples:

The following example starts the currently selected task at X'138'.

```
ST 138
```

The following example starts the currently selected task at X'100' and passes NOSEG,SCRAT to the task.

```
ST 100,NOSEG,SCRAT
```

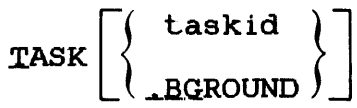
The following example starts the currently selected task at transfer address and passes 1000,ABC to the task.

```
ST ,1000,ABC
```

2.54 TASK COMMAND

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

Format:



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.

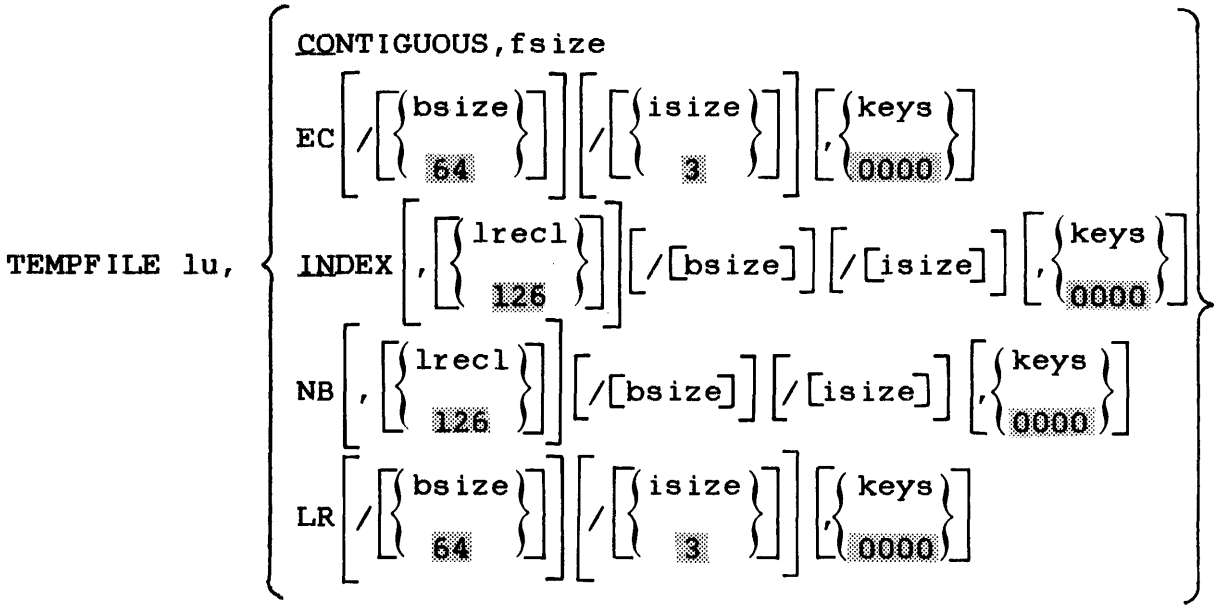
Examples:

```
T .BG

T COPY
```

2.55 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:



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 sysgen. If a value greater than the system maximum is specified, the system maximum is used. 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 and NB files is the value set at sysgen or by the system operator. For EC files, the default is 64 sectors.

isize — is a decimal number specifying the index block size. For INDEX and NB files, the default value is the value set at sysgen or by the system operator. For EC files, the default value is three sectors (768 bytes). The index block size cannot exceed the maximum disk block size established by sysgen. If a value greater than the system maximum is specified, the system maximum is used. isize may not exceed 255.

keys — specify the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword; the left byte signifies the write key and the right byte signifies the read key. If this parameter is omitted, both keys default to zero.

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.

LR — specifies a long record file. For long record files, the logical record length is specified by the data block size (bsize) parameter (i.e., the logical record length is the data block size).

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 for the system at sysgen or by operator command. If bsize and isize exceed this constant, the established maximum is used.

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 and task-executing modes.

For LR files, the absolute maximum data block size (logical record length) that can be specified is 65,535 (64K) sectors. This equals an absolute maximum logical record length of 16,776,960 (16M) bytes. In practice, however, the actual maximum logical record length for any given system is limited by the amount of memory available for I/O buffering.

Examples:

The following example allocates, on the temporary volume, a contiguous file with a total length of 64 sectors (16kb) and assigns it to lu2 of the loaded task.

```
TE 2,CO,64
```

The following example allocates, on the temporary volume, an index file with a logical record length. The data block and index block sizes default to the size established at sysgen or by the system operator. The file is assigned to lu14 of the loaded task.

```
TE 14,IN,126
```

The following example allocates, on the temporary volume, an extendable contiguous file with a default data block size of 64 and index block size of three 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 5,EC
```

The following example allocates, on the default temporary volume, a temporary nonbuffered indexed file with a logical record length of 240 bytes, data block size of 250 sectors and index block size of five sectors. The file initially contains no records. The file is assigned to lu7 of the task.

```
TE 7,NB,240/250/5
```

2.56 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, temporary or CSS volume.

Format:

```
VOLUME[{voln    [/CSS]}]
       {[*/CSS]       }
```

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.
CSS	is an option that allows the MTM user to specify a volume to contain user CSSs.
*/CSS	disables the CSS volume.

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 on-line 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) when the user signs on. If no volume was specified at MTM sysgen, the default is the system volume.

If CSS volume processing is enabled, the search order for CSS files is as follows:

Current volume/Private account

CSS volume/Private account

CSS volume/CSS account

System volume/System account

NOTE

See the SET CSS command for CSS volume/CSS account determination.

MTM signon sets the CSS volume to '*' to initially disable the CSS volume.

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

Examples:

In the following example, the VOLUME command is used to query the system.

```
VOL
USR=MTM   SYS=MTM   SPL=M67B   TEM=M67C   RVL=MTM  CSS=*
```

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

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

In the following example, the VOLUME command is used to change the default user volume and again to query the system.

```
V M301;V
USR=M301  SYS=MTM  TEM=M67C  RVL=MTM  CSS=*
```

In the following example, the VOLUME command is used to enable CSS volume processing:

```
V VOL1/CSS
```

where VOL1 is any volume name.

```
USR=M67B  SYS=MTM  TEM=TEMP  RVL=TEMP  CSS=VOL1
```

To disable the CSS volume searches, enter:

```
V */CSS
```

```
USR=M67B  SYS=MTM  TEM=TEMP  RVL=TEMP  CSS=*
```

2.57 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 Detail:

The WFILE command can be entered in task-loaded mode.

Examples:

The following example causes a filemark to be written on the device or file assigned to lu1.

```
WF 1
```

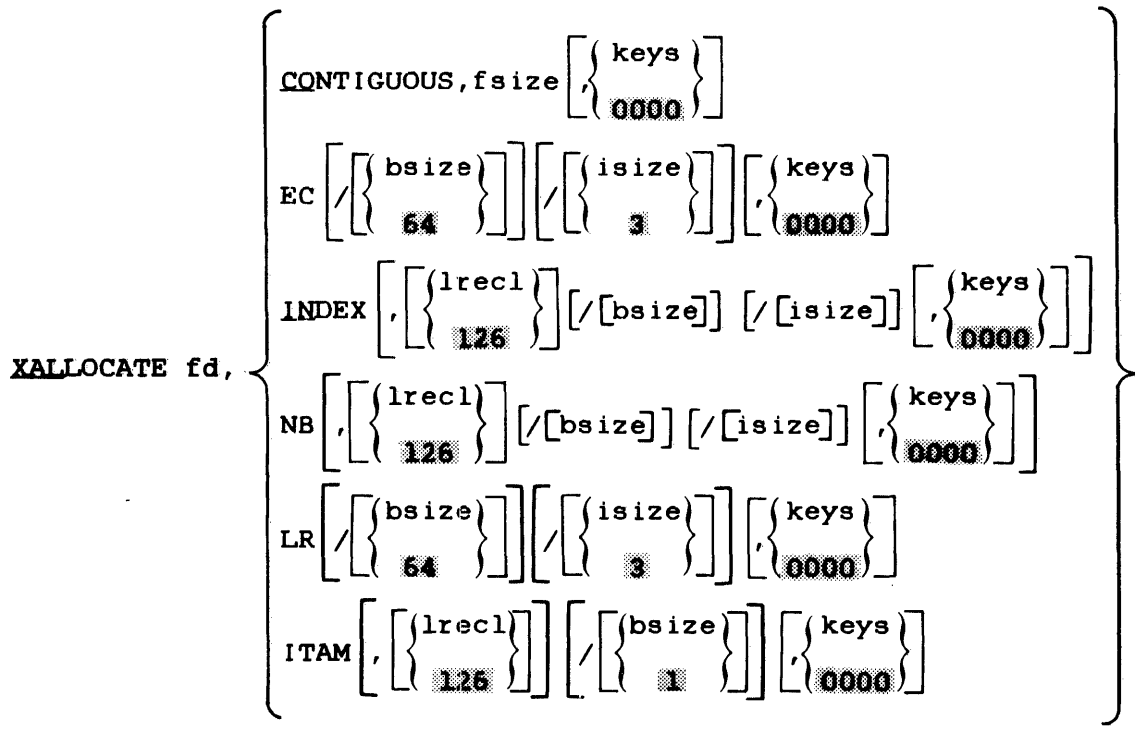
The following example causes a filemark to be written on file AJM.OBJ, which is assigned to lu4 on volume M300.

```
WF M300:AJM.OBJ,4
```

2.58 XALLOCATE COMMAND

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

Format:



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 — specify the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword; the left byte signifies the write key and the right byte signifies 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 ITAM buffers, this parameter cannot exceed the maximum block size established for the system. If a value larger than the system maximum is specified, the system maximum is used. For EC files, this parameter can be any value between 1 and 255 inclusive.

The default blocksize is established at sysgen and can be altered by the system operator. If no value is entered for this parameter, the default value set at sysgen (or by the system operator) is used. If no default was set at sysgen or by the system operator, 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 and NB files, the default value is established at sysgen or by the system operator. If no default was established through sysgen or by the operator, the default value is one sector. For EC files, the default value is three sectors (768 bytes). The index block size cannot exceed the maximum disk block size established for the system. If a value larger than the system maximum is specified, the system maximum is used. Neither bsize nor isize can 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 can optionally be followed by a slash (/), which delimits lrecl from bsize.
NB	specifies that the file type to be allocated is nonbuffered indexed.
LR	specifies a long record file. LR record files, the logical record length is specified by the data block size (bsize) parameter (i.e., the logical record length is the data block size).
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.

For LR files, the absolute maximum data block size (logical record length) that can be specified is 65,535 (64K) sectors. This equals an absolute maximum logical record length of 16,776,960 (16M) bytes. In practice, however, the maximum logical record length for any given system is limited by the amount of memory available for I/O buffering.

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

2.59 XDELETE COMMAND

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

Format:

XDELETE fd_1 [,fd_2...,fd_n]

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').

A nonprivileged MTM user can only delete private files.

Example:

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

CHAPTER 3
MULTI-TERMINAL MONITOR (MTM)/NON-MTM TASK INTERFACES

3.1 INTRODUCTION

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 taskid of 1- to 7-characters specifying the selected foreground task to which the MTM terminal is to be connected. The following taskids 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 command substitution system (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 supevisor call 6 (SVC6). For further information regarding SVC6 use, see the OS/32 Supervisor Call (SVC) Reference Manual.

Example:

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

```
$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 taskid can 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	is the device name of the user's terminal (variable length from two to five characters including a colon (:).
priv-acc	is the user's private account number (fixed length of five characters, right-justified, leading zeros).
group-acc	is the user's group account number (fixed length of five characters, right-justified, leading zeros).
userid	is the 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'0D')

Functional Detail:

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 ten 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 system generation (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. To allow command input from MTM, the start parameter IN=/MTM must be used.

Format:

$HASPxx

Parameters:

xx	is a 2-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 a double quote (").

All commands entered on the terminal are sent to the specified HASP task. All commands starting with a dollar sign ($) are prefixed with the HASP message command (i.e., $cmd is expanded to MES $cmd; this is transparent to the user) 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 can be 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:

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

```
$HASP03
```

3.4 INTEGRATED TRANSACTION CONTROLLER (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, see 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
C	C [[voln:] filename]	.C

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 (RTLs) 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 the user from constantly typing or even remembering 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:

In the following example, the FORTRAN language command entered with a user-supplied filename allocates an empty file, PROG1.FTN, then loads and starts the editor. The FORTRAN tab settings are set and displayed. The specified filename with the default extension is set as the current program and is always accessed and/or executed if the user does not specify another filename.

```
*FORT PROG1
*
*    New Language Environment -- FORTRAN VII D R05-01
*
*
* Editing new file -- PROG1.FTN  (APPEND mode set)
*
PERKIN-ELMER OS/32 EDIT 03-145 R03-01
OPTION TAB=\,7,73;OPTION INPLACE=OFF
GET PROG1.FTN;OPTION COM=CON:;AP
     1   >
    .
    .
    .
(edit session)
    .
    .
    .
>SAVE*
>END
-WORK FILE = M67B:PROG1.000/P
-RENUMBERED INPUT FILE AVAILABLE, M67B:PROG1.FTN/P
```

A source file can also be created by entering a language command without a filename and then entering the EDIT command with a filename. The EDIT command allocates a file, then loads and starts the editor. All edit commands can be employed to create a source file.

Example:

In the sequence below, 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.

```
*FORT
*
*   New Language Environment -- FORTRAN VII D R05-01
*
*EDIT PROG1
*
* New current program - PROG1
*
*
* Editing new file -- PROG1.FTN   (APPEND mode set)
*
PERKIN-ELMER OS/32 EDIT 03-145  R03-01
OPTION TAB=\,7,73;OPTION INPLACE=OFF
GET PROG1.FTN;OPTION COM=CON:;AP
    1   >
  .
  .
  .
(edit session)
  .
  .
  .
>SAVE*
>END
```

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:

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.

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

4.3 EXECUTING A PROGRAM

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

Example:

The following 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.

```
*EXEC
*Execution of PROG1.FTN follows:
-END OF TASK CODE=0
```

4.4 MODIFYING A PROGRAM

To modify a 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:

In the following 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.

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

4.5 REEXECUTING 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 (the object and image files will be older than the source file).

Dates and times (to the nearest minute) 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 date and timechecked. The source file is then compiled and/or linked if the object or image files are out of date or do not exist. The object module is assumed to be out of date if it is older than the source module, or if the dates and times of creation of the source and object modules are equal to the nearest minute. The image module is assumed to be out of date if it is older than the object module or if the dates and times of creation of the object and image modules are equal to the nearest minute. The EXEC command then loads and runs the image program.

Examples:

The following is an example of output from the EXEC command when compilation and link-edit are required. This occurs if no object file exists or if the object file is out of date.

```
*EXEC
* Compilation required
FORTRAN-VIID R05-01.00
MARYANN -END OF TASK CODE=   0    PROCESSOR=0.158/0.731
PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 R01-00
MARYANN -END OF TASK CODE=   0    PROCESSOR=3.197/2.442
*
*    Execution of PROG1.FTN follows:
*
 This is a demonstration of the EXEC command
STOP
MARYANN -END OF TASK CODE=   0    PROCESSOR=0.010/0.016
```

The following is an example of when only link-edit is required. This occurs if no image file exists or if the image file is out of date.

```
*EXEC
PERKIN-ELMER OS/32 LINKAGE EDITOR 03-242 R01-00
MARYANN -END OF TASK CODE=   0    PROCESSOR=3.196/2.466
*
*    Execution of PROG1.FTN follows:
*
 This is a demonstration of the EXEC command
STOP
MARYANN -END OF TASK CODE=   0    PROCESSOR=0.010/0.016
```

In the following example, the EXEC command executes all the modules as one program and displays end of task code = 0 after successful execution.

```
*EXEC
*
*     Execution of PROG1.FTN follows:
*
 This is a demonstration of the EXEC command
STOP
MARYANN -END OF TASK CODE=   0     PROCESSOR=0.011/0.016
```

In all three cases, PROG1.FTN was previously established as the current program in the proper environment.

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:
*
 This is a demonstration of the RUN command
STOP
MARYANN -END OF TASK CODE=  0      PROCESSOR=0.010/0.016
```

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

```
RUN
*
*  Must have current program or specify file in order to run
*
```

If a user only wants 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 (multimodule environment), the user must include the file descriptors (fds) of each source file in an environment descriptor file (EDF). The EDF retains the identity of all the source files in the multimodule environment that will be used to create a program.

When using the program development ENV command, the user indicates that a source program exists in more than one file and is to be created in a multimodule environment. The ENV command creates the multimodule environment and allocates an EDF to contain the fds of the source files.

Example:

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

```
*ENV ALLPROG
*
*   New multimodule environment is ALLPROG.EDF
*
*          No current program
*         Link commands are standard
```

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 multimodule environment.

Example:

In the following example, the multimodule 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 multimodule environment.

```
*ENV ALLPROG
*
*   New multimodule environment is ALLPROG.EDF
*
*          No current program
*         Link commands are standard
*
*ADD PROG1.FTN
*ADD PROG2.CBL
```

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 multimodule environment, it is added. If it is already in the multimodule environment, the following message is displayed:

```
*ADD PROG1.FTN
*
*  PROG1.FTN already exists in environment ALLPROG.EDF
```

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

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

Example:

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

```
*LIST
*
*  Current multimodule environment is ALLPROG.EDF
*
*       Current program = PROG2.CBL
*        Link commands are standard
*
 Contents of Environment file:
 PROG1.FTN
 PROG2.CBL

*REMOVE PROG2.CBL
*LIST
*
*   Current multimodule environment is ALLPROG.EDF
*
*        Current program = PROG2.CBL
*         Link commands are standard
*
 Contents of Environment file:
 PROG1.FTN

*EXEC
*
*    Execution of ALLPROG.EDF follows:
*
 This is a demonstration of the EXEC command
STOP
MARYANN -END OF TASK CODE=   0    PROCESSOR=0.011/0.016
```

If the ADD or REMOVE command is entered without an fd or if the fd is incorrect, a brief description of the command is displayed.

Not all program development commands are available in both language and multimodule 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 multimodule environment is entered in a language environment, the following message is displayed:

```
*
* Must be in a multimodule environment to use the command xxxxxxxx
*
```

The xxxxxxxx portion of this message is replaced with the name of the command. A brief description of the command is then displayed.

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

Example:

In the following example, the multimodule environment is entered via the ENV command and the EDF name, ALLPROG. PROG1.FTN is added to the multimodule environment. The LIST command displays the filenames saved 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).

```
*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
>
```

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:

```
* PROG1.FTN Compilation 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 reexecute the program. The EXEC command will recompile only the modified modules.

The EXEC command will occasionally cause a successfully compiled program to be recompiled. This happens when the creation times of the source and object files are equal (to the nearest minute). Any program that is recompiled will be relinked. A program that is successfully compiled and linked may be unexpectedly relinked by the EXEC command if the creation time of the object and image files are equal (to the nearest 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	LU
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 preallocated 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
- LANGUAGE
- LINK
- LIST
- REMOVE
- RUN

4.9.1 ADD Command

The ADD command adds the new source file fds to the multimodule environment. These fds are retained in the EDF.

Format:

ADD source-fd [,compile-css] [,arg_1,...arg_7]

Parameters:

source-fd	is the file descriptor of the source file to be added to the EDF file.
compile-css	specifies the name of the command substitution system (CSS) file to be used in the compilation of the source-fd file.
arg	can be up to seven compilation arguments. These arguments are dependent upon the language of the source file. The user should refer to the appropriate language environment commands for details.

Functional Details:

The ADD command causes the filename of the specified source-fd to be searched for in the current EDF. If the filename is not found, the source-fd is added to the multimodule environment that was previously invoked by the ENV command. If a matching filename (regardless of extension) currently exists in the environment, the file will not be added to the EDF and the following message is displayed:

```
* FILENAME already exists in environment edfname.EDF
```

If the filename is omitted or is in an incorrect format, this message is displayed, followed by the HELP feature of the ADD command:

```
*
* ADD requires at least one argument
*
```

If the filename is entered without an extension, the following message is displayed, followed by the HELP feature of the ADD command:

```
*
* Must specify extension or compile css name
*
```

Any file added to the EDF file becomes the current program.

The compile-css 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 using a nonstandard extension, a search is made for a file named COMPext.CSS (where ext is the nonstandard extension) and the following message is displayed:

```
*
* Compile css COMPext.CSS does not exist
*
```

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

The ADD command is valid only in a multimodule environment. An error message is output when an attempt is made to use this command in a language or null environment.

Examples:

The following example demonstrates the addition of three source files to the current multimodule environment EDF:

```
*ADD DEMO1.FTN
*ADD DEMO2.CAL
*ADD DEMO3.RPG
```

The following example demonstrates the attempted addition of an fd for which a matching filename already exists within the EDF. As shown in the example, the filename must be unique. When adding an fd to the EDF, different extensions will not suffice.

```
*ADD DEMO3.CBL
*
*  DEMO3.CBL already exists in environment ANYTHING.EDF
*
```

This example demonstrates the message displayed when the ADD command is issued without an fd. This message is immediately followed by the HELP feature for the ADD command.

```
*ADD
*
*  ADD requires at least one argument
*
```

The following example illustrates the message displayed when the ADD command is issued with a filename only (no extension specified). This message is immediately followed by the HELP feature of the ADD command.

```
*ADD PROG4
*
*  Must specify extension or compile css name
*
```

In the following example, an fd with a nonstandard extension is specified, but the compile-css parameter is not used to pass the name of the required compilation CSS.

```
*ADD DEMO4.DIF
*
*  Compile css COMPDIF.CSS does not exist
*
```

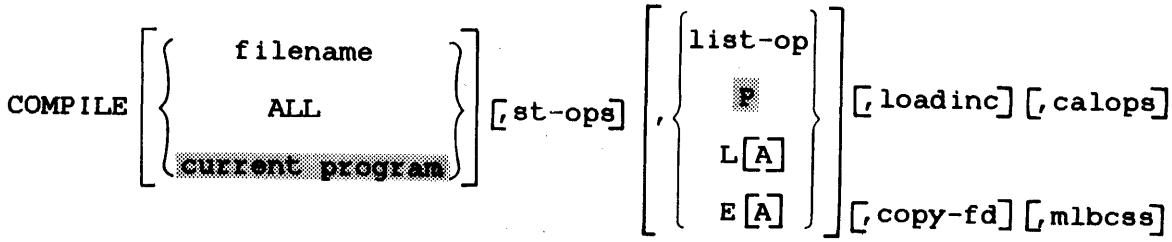
In this example, an fd with a nonstandard extension is added to the EDF file. The name of the CSS procedure required to compile this file is passed via the compile-css parameter.

```
*ADD DEMO4.DIF,OTHRCOMP.CSS
```

4.9.2 COMPILE Command

The COMPILE command compiles a source file or group of source files when in a language, multimodule or null environment. The COMPILE command conditionally compiles when the ALL parameter is specified in the multimodule environment. The COMPILE command does not execute a program.

Format:



Parameters:

filename	specifies a 1- to 8-character alphanumeric filename of the source file to be compiled and/or assembled. The extension for the source file fd is assigned depending upon the current language environment. If the extension is other than the default of the current environment, or only one file of a multimodule environment is to be compiled, the extension must be specified.
ALL	specifies that all source files in the EDF file are to be compiled. If neither filename nor ALL is specified, the current program is the default. This only applies in a multimodule environment.
st-ops	can be used to specify START options for the compilation process. If this parameter is omitted, START options default by language.
list-op	is the fd of an existing file to which the compilation listing is sent.
P	indicates that the compilation listing is to be sent to the file or device specified by the SSYSPRINT global variable.

L	indicates that a new file, currprog.LST, is to be allocated to receive the compilation listing. If A is included, the listing is to be appended to the existing currprog.LST file.
E	indicates that a new file envir.LST, is to be allocated to receive the listing. If A is included, the listing is appended to the existing envir.LST file. This only applies in a multimodule environment.
loadinc	specifies the segment size increment to be used for compilation or assembly. The default is by language.
calops	specifies the START options for environments other than CAL, FORT and PASCAL. The default is by language.
copy-fd	specifies the name of the copy file (if required) to be assigned to lu7 for assembly.
mlbcss	used in the MACRO environment to specify the CSS file to be used in assigning the required library units for Macro/32.

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 list-op parameter is omitted, the value specified by the SSYSLST global variable is used to determine the destination of the compilation listing.

If the environment is not set and no filename is specified when the COMPILE command is entered, the following message is displayed:

```
*
*  Filename must be specified or current program established
*  before compilation
*
```

If no environment is set and a filename is specified, the following message is displayed:

```
*
*  Must be in environment or specify extension to compile
*
```

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

```
*
* FILENAME not found in environment edfname.EDF
*
```

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

Examples:

No environment is set and no filename is specified.

```
*COMPILE
*
*  Filename must be specified or current program established
*  before compilation
*
```

In the following example, no environment is set but a filename is specified.

```
*COMPILE DEMO1
*
*  Must be in environment or specify extension to compile
*
```

In the following example, the environment is set (to FORT) but the specified file does not exist.

```
*COMPILE NOSUCH
*
* NOSUCH.FTN not found in environment ALLPROG.EDF
*
```

In the following example, the command is issued with the proper environment established, and the specified file exists.

```
*COMPILE DEMO1
FORTRAN-VIID R05-00.00
.MAIN         NO  ERROR(S)      TABLE SPACE USED:      1 K
BARNEY  -END OF TASK CODE=     0     PROCESSOR=0.632/0.117
```

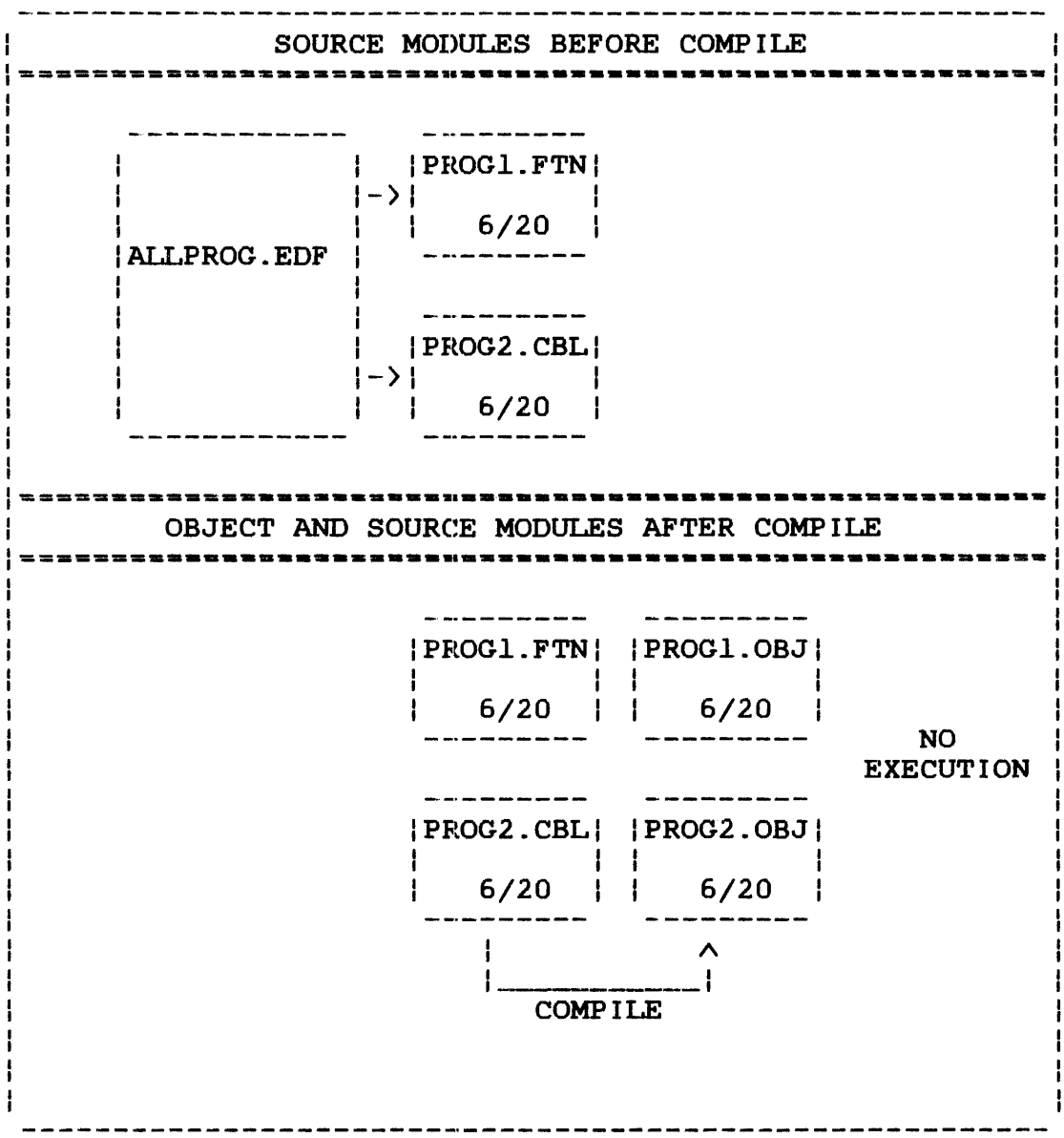


Figure 4-1 COMPILE Command Functions in the Language Environment

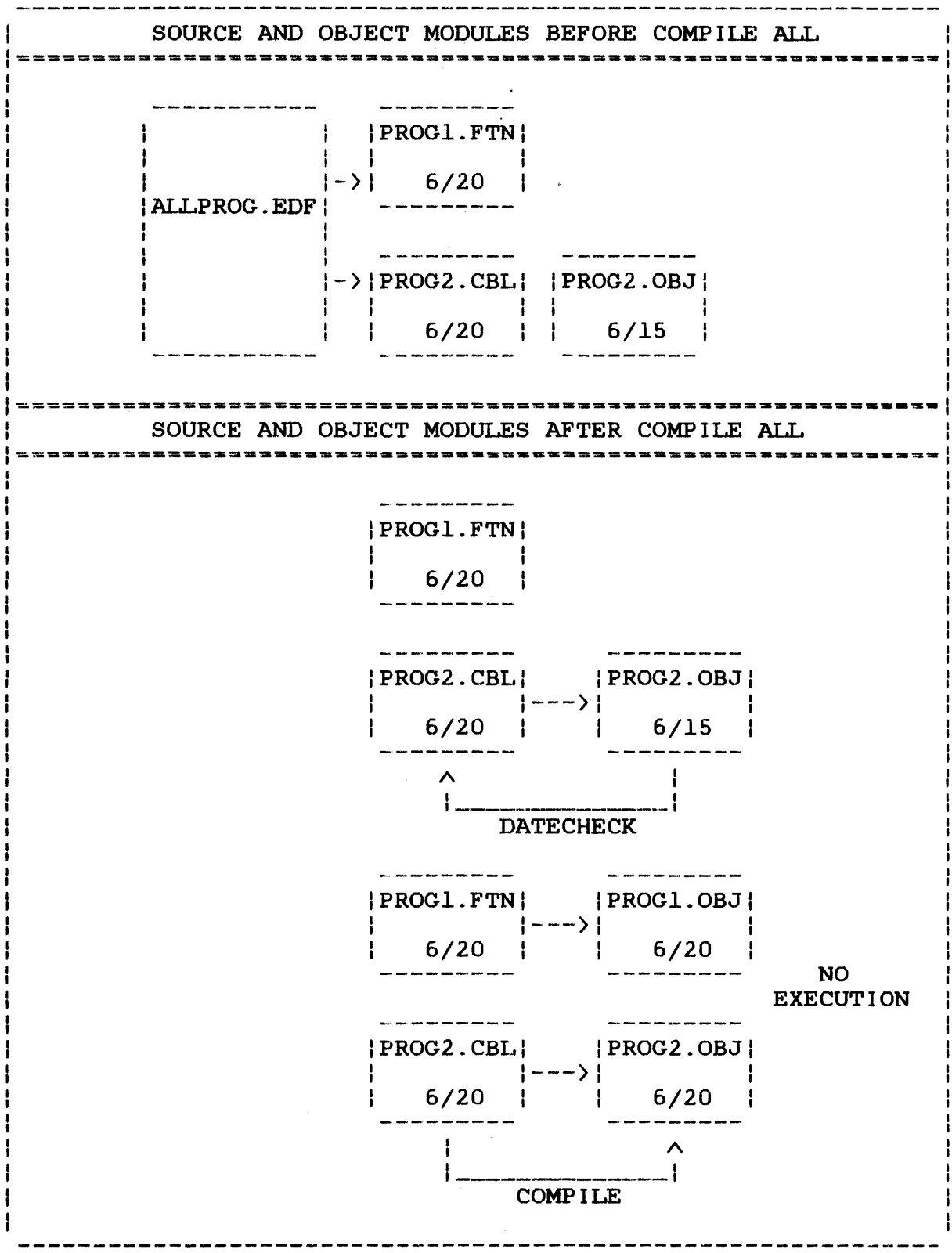
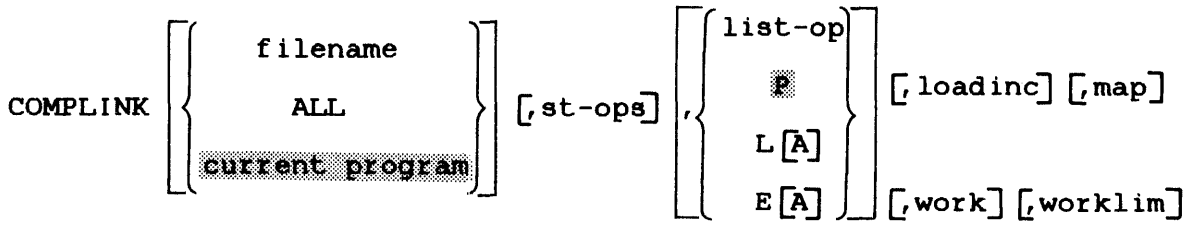


Figure 4-2 COMPILE Command Functions in the Multimodule Environment

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, multimodule or null environments. If all modules are up to date, this command does not perform any function. This command does not execute the program.

Format:



Parameters:

filename	is a 1- to 8-character alphanumeric filename of the source file to be compiled and/or linked. If the filename is specified and has no extension or has the extension of the current language environment, it is checked for existence and becomes the current program. If the filename is omitted (as it must be in a multimodule environment), the current program or environment is the default.
st-ops	can be used to specify START options for the compilation process. If this parameter is omitted, START options default by language.
list-op	is the fd of an existing file to which the compilation listing or Link output is sent.
P	indicates that the compilation listing is to be sent to the file or device specified by the SSYSPRINT global variable.
L	indicates that a new file, currprog.LST, is to be allocated to recieve the compilation listing. If A is included, the listing is to be appended to the existing currprog.LST file.

E	indicates that a new file, envir.LST, is to be allocated to receive the listing. If A is included, the listing is appended to the existing envir.LST file. This only applies in a multimodule environment.
loadinc	specifies the segment size increment to be used for compilation or assembly. The default is by language.
map	is the fd of an existing file to which the Link map is sent. If this parameter is omitted, the default is to list-op.
work	indicates memory workspace is to be allocated to a task. The default is dependent upon the language.
worklim	indicates the upper boundary of the load increment size.

Functional Details:

When the COMPLINK command is used in a multimodule environment, all the fds contained in the EDF are datechecked, compiled and/or linked.

If the object file, filename.OBJ, does not exist or is older than the source file, it will be recompiled. If any recompilation is required, a task does not exist, or the task file is older than any object file, OS/32 Link is invoked to produce a new task.

When a Link command file, filename.LNK (envir.LNK for multimodule environments) exists, it is unconditionally used; otherwise, one will be built automatically and discarded after use. The BLINK command enables the user to build a permanent command file. It can then be edited to incorporate any special linking requirements.

If the list-op parameter is omitted, the value specified by the SSYSLST global variable is used to determine the destination of the compilation listing.

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. The following message is then displayed:

```
* FILENAME Compilation errors-listing on (device):
```

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

```
*
* FILENAME not found in environment edframe.EDF
```

If any arguments are specified in a multimodule environment, the following message is displayed:

```
*
*  First argument filename is not permitted when in a
*  multimodule environment.  Environment name is
*  always used.
*
```

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

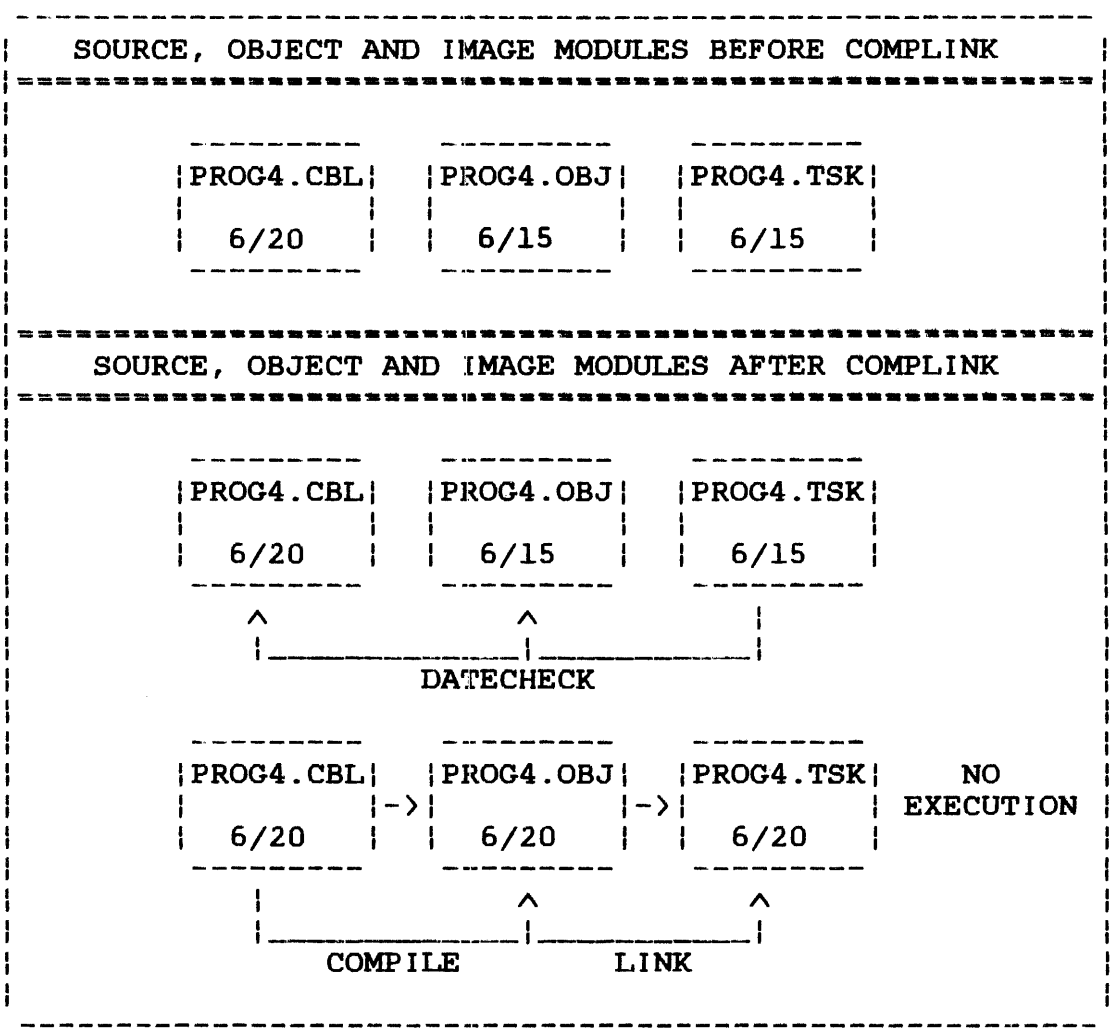


Figure 4-3 COMPLINK Command Functions in the Language Environment

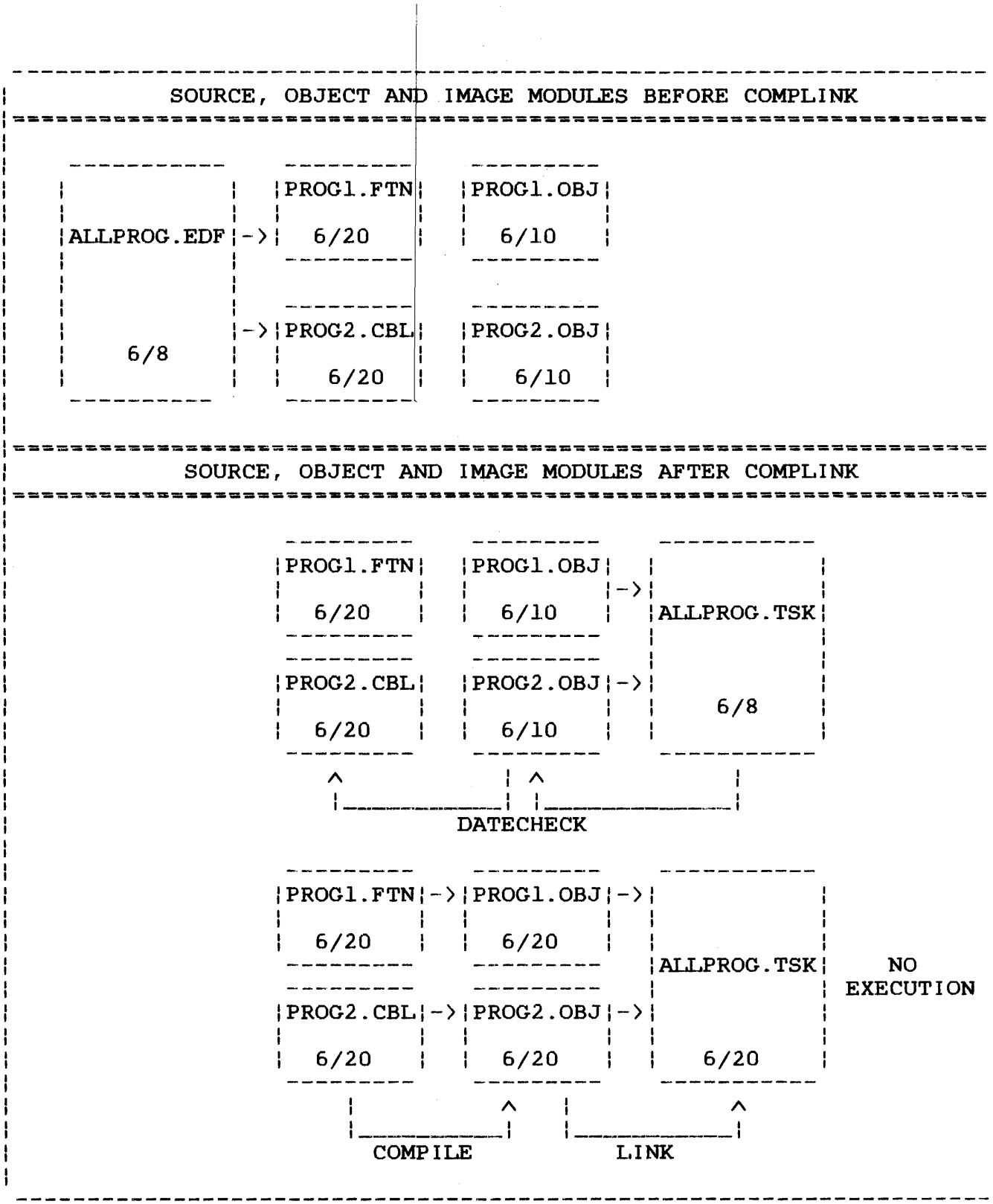
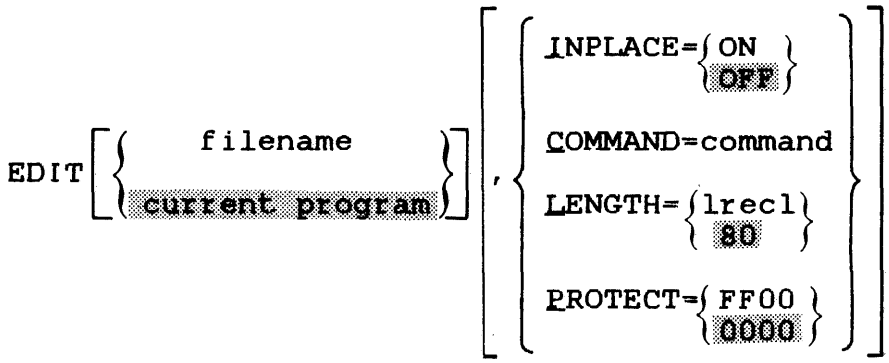


Figure 4-4 COMPLINK Command Functions in the Multimodule Environment

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:



Parameters:

filename	specifies a 1- to 8-character alphanumeric filename of the source file to be created or edited. If this parameter is omitted, the current program is the default.
INPLACE	editing is done in place if the edit file exists.
COMMAND	specifies the first edit command to be performed on the source file if the file exists. If the file specified for editing does not exist, it is allocated and COMMAND= is forced to APPEND.
LENGTH	allocates the edit file with the specified record length if file does not exists. The default is 80.
PROTECT	protection keys of the edit file are cleared during editing, but are set to FF00 after editing is completed.

Functional Details:

The EDIT command can be used whether or not any environment has been established. If no environment has been established, the source file is simply edited.

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

The user can simply type EDIT, which causes the editor to be loaded and started without the intervention of the program development system. If no current program is established, the following message is displayed to prompt the user to supply the desired filename:

```
GIVE FILENAME=
```

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 multimodule environment, the following message is displayed:

```
* FILENAME not in edframe.EDF environment--will edit anyway
```

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

```
*
* Enter name of file to be edited
* or * to just start EDITOR
*
GIVE FILENAME=
```

If the EDIT command is entered while a current program is established, the current program is made available for editing. For example, if the language environment is set to FORT with a current program of PROG1.FTN, the EDIT command produces the following results.

```
*EDIT
PERKIN-ELMER OS/32 EDIT 03-145 R03-01
OPTION TAB=\,7,73;OPTION INPLACE=OFF
GET PROG1.FTN;OPTION COM=CON:;SC
     1    >C    GO RIGHT AHEAD AND
     2    >C    ENTER THE PROGRAM
     3    >C    YOU ARE DEVELOPING.
!UNABLE TO TYPE FULL SCREEN
>
```

If no current program is established, the result of the EDIT command is the following:

```
*EDIT
*
*  Enter name of file to be edited
*  or * to just start EDITOR
*
GIVE FILENAME=
```

If the user responds at this point by supplying an fd, the editor either retrieves an existing file or allocates a new one, depending on the existence or absence of the specified file.

```
GIVE FILENAME=ANYTHING.EXT      (file does not exist)
*
*  Editing new file -- ANYTHING.EXT  (APPEND mode set)
*
PERKIN-ELMER OS/32 EDIT 03-145 R03-01
OPTION TAB=/;OPTION INPLACE=OFF
GET ANYTHING.EXT;OPTION COM=CON:;AP
     1    >
```

or

```
GIVE FILENAME=ANYTHING.EXT    (file exists)
PERKIN-ELMER OS/32 EDIT 03-145 R03-01
OPTION TAB=/;OPTION INPLACE=OFF
GET ANYTHING.EXT;OPTION COM=CON:;AP
     1    >ETCETERA
```

If no extension is specified, the default extension for the current language environment is assigned. If no language environment has been established, no default extension is assigned and the source file is edited.

If the user enters the asterisk (*) character, the system responds in the following manner:

```
GIVE FILENAME=*
*
*  Entering EDITOR with no file (in APPEND mode)
*
PERKIN-ELMER OS/32 EDIT 03-145 R03-01
OPTION COM=CON:;AP
    1   >
```

If a filename other than the filename of the current program is specified, that file is established as the current program and is made available for editing (provided the file has the proper extension). For example, if the current language environment is FORT, and no current program exists or the current program is other than PROG1.FTN, the following EDIT command will produce the indicated response.

```
*EDIT PROG1
*
*  New current program - PROG1
*
PERKIN-ELMER OS/32 EDIT 03-145 R03-01
OPTION TAB=\,7,73;OPTION INPLACE=OFF
GET PROG1.FTN;OPTION COM=CON:;SC
    1     C   GO RIGHT AHEAD AND
    2     C   ENTER THE PROGRAM
    3     C   YOU ARE DEVELOPING.
!UNABLE TO TYPE FULL SCREEN
>
```

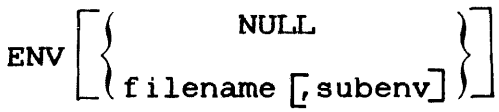
For information on the edit commands, see the OS/32 Edit User Guide.

```
 ----------
|   ENV    |
 ----------
```

4.9.5 ENV Command

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

Format:



Parameters:

NULL	clears the language or multimodule environments.
filename	is a 1- to 8-character alphanumeric name specifying the EDF, filename.EDF, which creates and/or sets multimodule environments. The EDF extension is automatically appended and must not be entered by the user. The optional subenv permits users to specify subenvironments within language or multimodule environments.

Functional Details:

If the filename parameter is entered with an extension other than .EDF, the following message is displayed:

```
* Environment name must have no extension or .EDF
```

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

```
* Current multimodule environment is xxxxxxxx
```

If the environment was not set or the NULL parameter was specified at signon, the following message is displayed:

```
* No current environment
```

4.9.6 EXEC Command

The EXEC command will compile and link a program in the language or null environments or compile and link all modules in a multimodule environment if they are outdated. When the image program is current, it is loaded and run.

Format:

EXEC [{ filename / current program }] [,runops] [,runincr]

Parameters:

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.
runops	is used to start the resulting task.
runincr	once the object files are linked to create the task, the task is loaded with a load increment, which is set by runincr.

Functional Details:

The source file is compiled if no object file exists, or if it is not older than the object file.

The object file is linked if no image file exists, or if it is not older than the image file.

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

If a link error occurs, the following message is displayed:

* Link errors--listing on device:

The filename must not be specified in a multimodule environment since the entire environment is assumed.

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

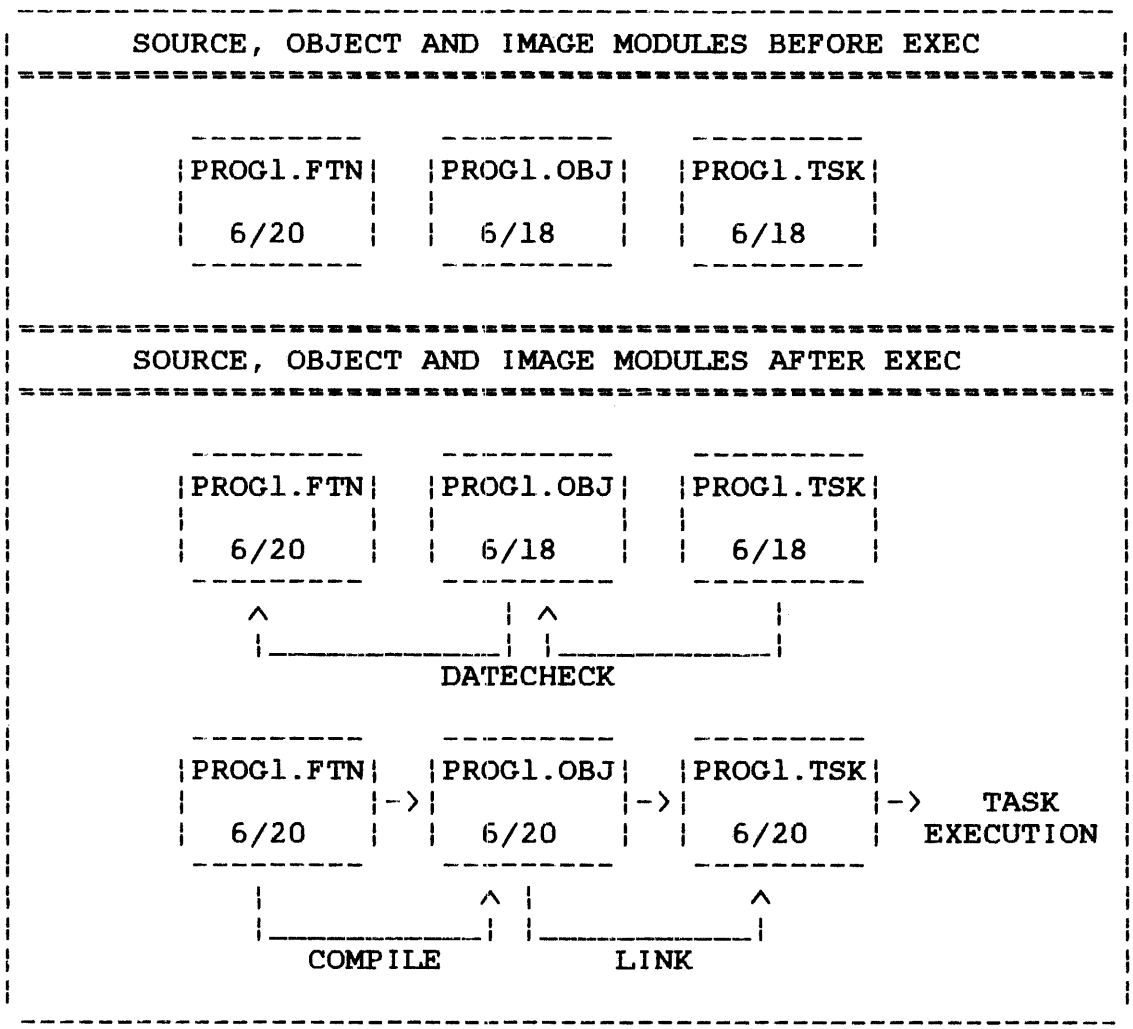


Figure 4-5 EXEC Command Functions in the Language Environment

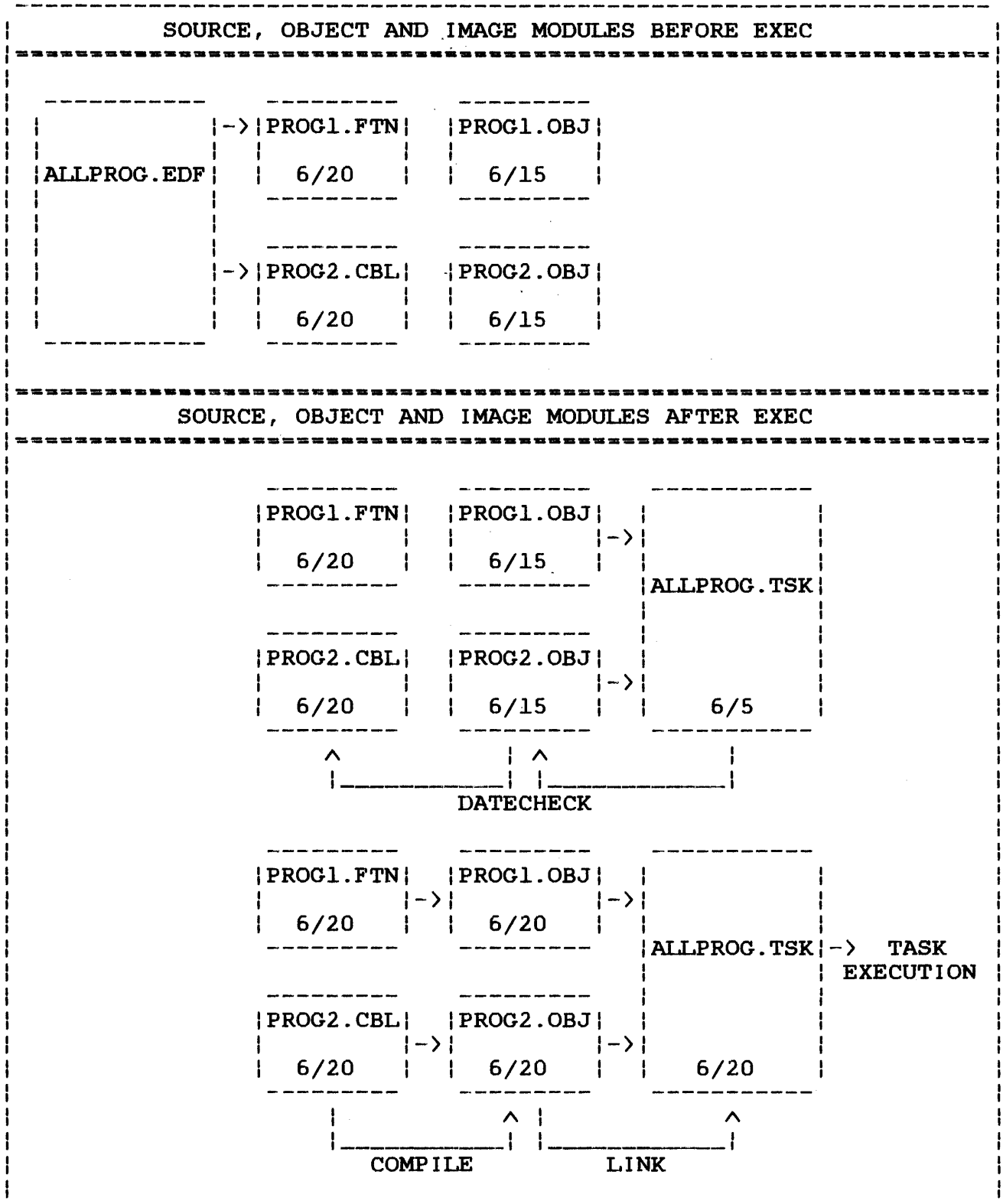


Figure 4-6 EXEC Command Functions in the Multimodule Environment

| ----------
| | LANGUAGE |
| ----------

4.9.7 LANGUAGE Command

The LANGUAGE command is used to change the program development language environment or, for multimodule environments, to designate the FORTRAN VII compiler to be used to compile any FORTRAN program in the EDF file.

Format:

```
LANGUAGE        [language environment]
```

Parameters:

FORT	specifies the FORTRAN VII development compiler or environment.
FORTO	specifies the FORTRAN VII global optimizing compiler or environment.
FORTZ	specifies the FORTRAN VII universal optimizing compiler or environment.
COBOL	specifies the COBOL language environment.
CAL	specifies the CAL/32 environment.
MACRO	specifies the MACRO environment.
PASCAL	specifies the PASCAL language environment.
RPG	specifies the RPG language environment.
C	specifies the C language environment.

Functional Details:

The selected argument must be entered in upper-case for the LANGUAGE command to function correctly.

The LANGUAGE command is used to change the program development environment or to determine the current environment.

FORTRAN is a special case, because more than one compiler may be available to process files with the .ftn extension. The default compiler for FORTRAN programs is the FORTRAN VII development compiler. To change the default language processor, type LANGUAGE FORTO or LANGUAGE FORTZ. For multimodule environments, the LANGUAGE command assigns the compiler to be used to compile any FORTRAN modules found in the EDF file. This command does not affect the compilation of programs for which a special compilation CSS was specified via the ADD command.

When clearing the language or multimodule environments through the ENV command with the option NULL, the compiler is set to the default FORT. The compiler is also set to the default (FORT) when the filename option is used with the ENVIRONMENT command.

The default FORTRAN compiler is not changed when the ENVIRONMENT command is issued with no arguments to determine what environment the user is currently in.

Examples:

Set the language environment to PASCAL.

```
*PASCAL
*
*   New Language Environment -- Pascal R01
*
```

Determine the current environment.

```
*LANGUAGE
* LANGUAGE = PASCAL
```

Change the current environment to COBOL.

```
*LANGUAGE COBOL
* LANGUAGE = COBOL
```

This example demonstrates the use of the language command to determine which FORTRAN VII compiler will be used in the multimodule environment. The EDF file contains a FORTRAN source file named PROG1.FTN.

```
*ENV PROG
*
*   Current multimodule environment is PROG.EDF
*
*          No current program
*         Link commands are standard
*

*LANGUAGE FORTO
* LANGUAGE = FORTO

* COMPILE PROG1
FORTRAN-VIIO R05-01.00

                        .
                        .
                        .

COMPILER FILE:  MTM:F7051/S            SOURCE LISTING:  3,CON:
INPUT FILE:  1,M301:PROG1.FTN/P  OBJECT FILE: 2,M301:PROG1.OBJ/P
ELAINE  -END OF TASK CODE=   0    CPUTIME=0.670/0.247
```

The following example illustrates that the use of the ENV command to report the current environment will not affect the FORTRAN compiler chosen by a previous LANGUAGE command.

```
*ENV
*
*   Current multimodule environment is PROG.EDF
*
*        Current program = PROG1.FTN
*         Link commands are standard
*
*COMPILE PROG1

FORTRAN-VIIO R05-01.00

                                  .
                                  .
                                  .
COMPILER FILE:  MTM:F7051/S            SOURCE LISTING:  3,CON:
INPUT FILE: 1,M301:PROG1.FTN/P OBJECT FILE:  2,M301:PROG1.OBJ/P
ELAINE  -END OF TASK CODE=   0    CPUTIME=0.667/0.250
```

This example illustrates the use of the filename option in the ENV command, which will cause the default compiler (FORT) to be used.

```
*ENV PROG
*
*   Current multimodule environment is PROG.EDF
*
*          No current program
*         Link commands are standard
*
*COMPILE PROG1

FORTRAN-VIID R05-01.00
.MAIN        NO  ERROR(S)    TABLE SPACE USED:     1 K

                             .
                             .
                             .

STATEMENT BUFFER:  20 LINES/1321 BYTES STACK SPACE: 40 WORDS
ELAINE  -END OF TASK CODE=   0    CPUTIME=0.159/0.936
```

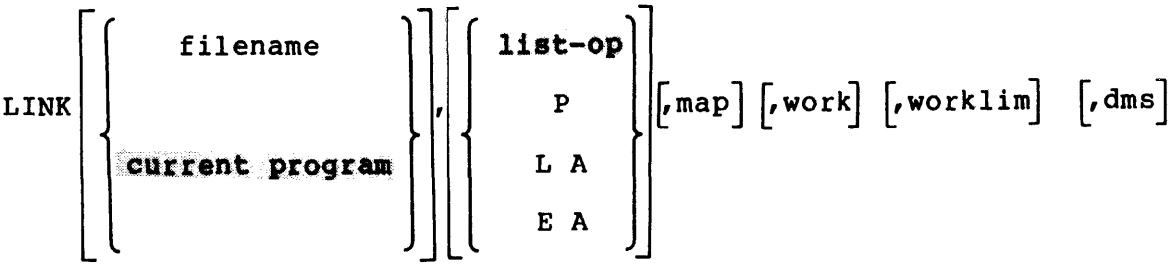
This example illustrates the option NULL, which will cause the compiler to default to FORT.

```
*env NULL

*compile prog1.ftn
*
*   New Language Environment -- FORTRAN VII D R05-01
*
FORTRAN-VIID R05-01.00

                             .
                             .
                             .

 STATEMENT BUFFER: 20 LINES/1321 BYTES STACK SPACE: 40 WORDS
ELAINE  -END OF TASK CODE=   0    CPUTIME=0.159/0.943
```

4.9.8 LINK Command

The LINK command links the object module to yield the image module in language, multimodule or null environments. If no object module exists, the LINK command causes the source module to be compiled to yield the object module. The LINK command does not datecheck, load or execute a program.

Format:



Parameters:

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.
list-op	is the fd of an existing file to which the compilation listing is sent.
P	indicates that the compilation listing is to be sent to the file or device specified by the SSYSPRINT global variable.
L	indicates that a new file, currprog.LST, is to be allocated to receive the compilation listing. If A is included, the listing is to be appended to the existing currprog.LST file.
E	indicates that a new file, envir.LST, is to be allocated to receive the listing. If A is included, the listing is appended to the existing envir.LST file. This only applies in a multimodule environment.
map	is the name of an existing file which will receive the Link map. If this parameter is omitted, list-op is the default.

work	is used to allocate memory workspace to a task. If this parameter is omitted, the default is by language.
worklim	specifies the upper boundary of the load increment size for a task.
dms	is the name of the data management system (DMS) run-time library (RTL) segment to be resolved against and can only be used with DMS subenvironments. Defaults to DMS.RTL.

Functional Details:

When the LINK command is entered in a multimodule environment and no object module exists, all source file fds contained in the EDF are compiled. The resulting object modules are then linked.

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

4.9.8.1 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 and a filename matching the EDF name or the filename of the current program. When 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.

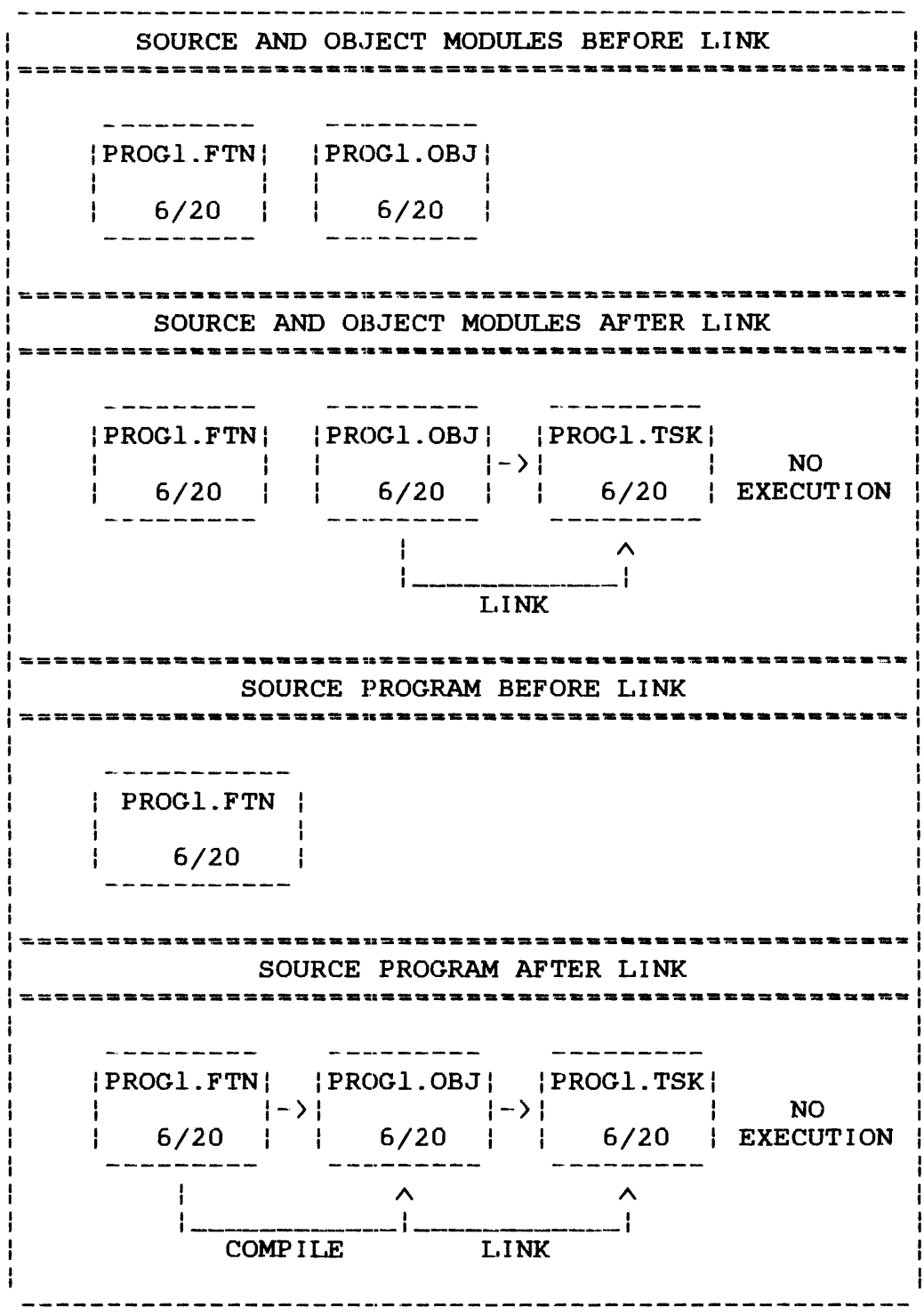


Figure 4-7 LINK Command Functions in the Language Environment

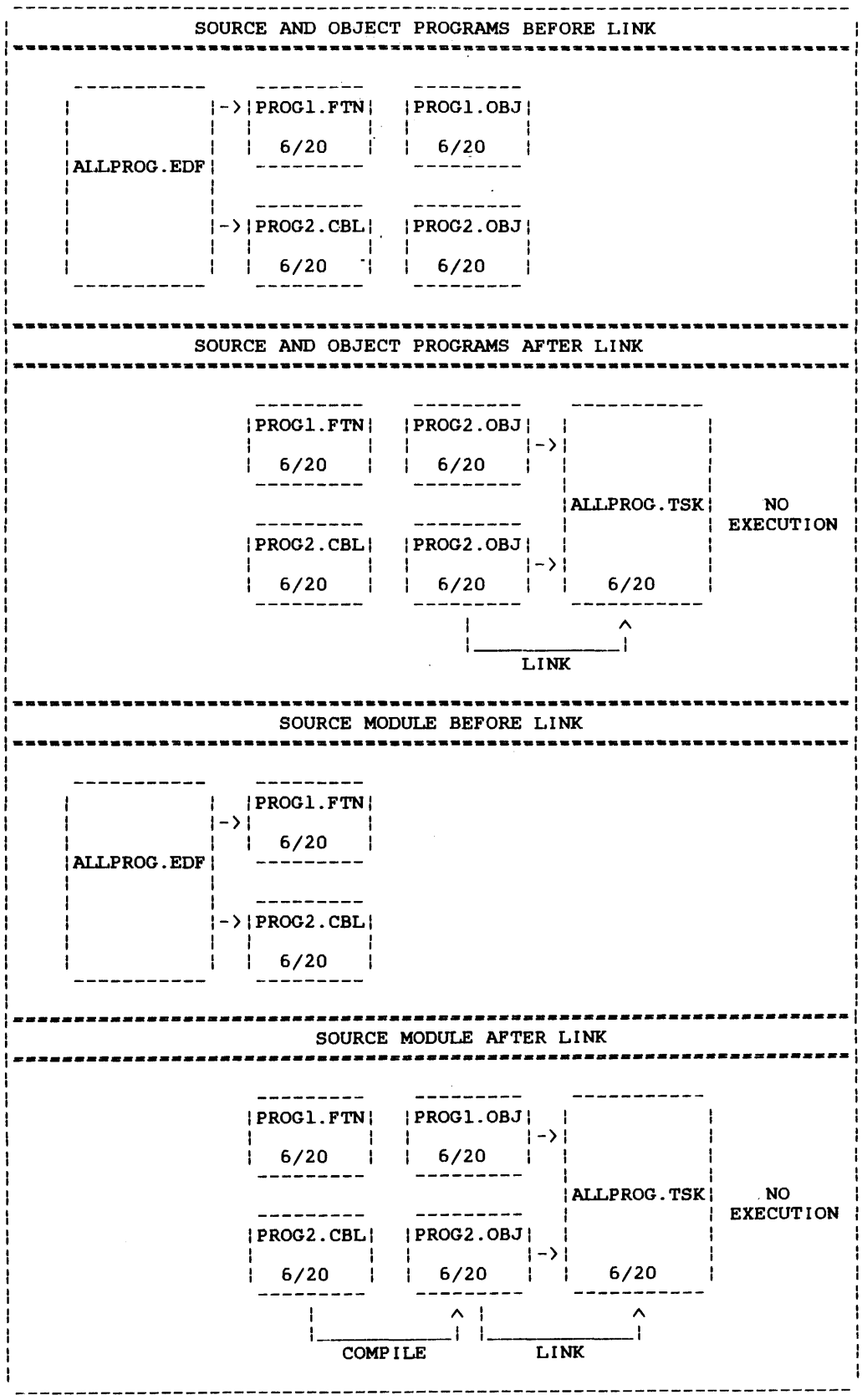


Figure 4-8 LINK Command Functions in the Multimodule Environment

4.9.9 LIST Command

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

Format:

LIST [{ filename / * }]

Parameters:

filename	is a filename of one of the members of the current EDF. If a filename is specified, and special compiler arguments applicable to the source file, the special compile CSS (if any) are displayed.
*	indicates that compiler arguments and special compile CSSs are to be displayed for all members of the current EDF.

Functional Details:

The LIST command causes a listing to be sent to the list device specified by SSYSPRT when lu assignments were made. The user must be in a multimodule environment to use the LIST command. If not, an appropriate message is displayed.

When the user is in a multimodule environment, the LIST command can be used without parameters to display pertinent current information and the fds of all files in the environment. If a file with a nonstandard extension is contained within the environment, the name of the required compilation CSS is also displayed.

```
*LIST
*
*  Current multimodule environment is ANYTHING.EDF
*
*          Current program--DEMO4.DIF
*         Link commands are standard
*
 Contents of Environment file:
 DEMO1.FTN
 DEMO2,CAL
 DEMO3.RPG
 DEMO4.DIF            compiled with OTHRCOMP
```

If a particular fd is specified with the LIST command, the file, its compile arguments and the name of the nonstandard compilation CSS (if any) are displayed. This information is displayed for all members of the environment if an asterisk (*) is entered with the LIST command.

```
*LIST *
*
*   Current multimodule environment is ALLPROG.EDF
*
*         Current program = DEM03.PAS
*          Link commands are standard
*
 Contents of Environment file:

 DEMO1.FTN            compiled with OTHRCOMP
 compile arguments = "LCNT=55",TRACE,WARN,"",XREF,,
 DEMO2.CAL
 compile arguments = "",SCRAT,SQUEZ,"",FREEZE,,
 DEM03.PAS
 compile arguments = "",,,"",,,
```

If the LIST command is entered and no fds are in the multimodule environment, an appropriate message is displayed.

4.9.10 REMOVE Command

The REMOVE command deletes specified source fds from the current multimodule 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 multimodule environment. If the fd is not found, the following message is displayed:

```
*
*  FILENAME not found in environment edfname .EDF
*
```

The REMOVE command is only valid in a multimodule environment, otherwise an error message is output when an attempt is made to use it in a language or null environment.

4.9.11 RUN Command

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

Format:

```
RUN [{ filename        }][,st-ops]
    [{ current program }]
```

Parameters:

filename	is a 1- to 8-character name specifying the image module. If this parameter is omitted, the default is the current program.
st-ops	is used to start the task.

Functional Details:

If a filename is not entered with the RUN command, the following message is displayed:

```
*
*  Must have current program or specify file in order to run
*
```

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

```
*
*  CANNOT RUN -- filename.TSK DOES NOT EXIST
*
```

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

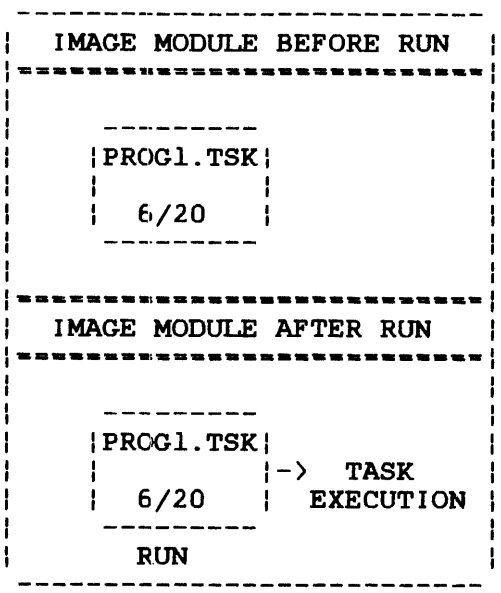


Figure 4-9 RUN Command Function in the Language Environment

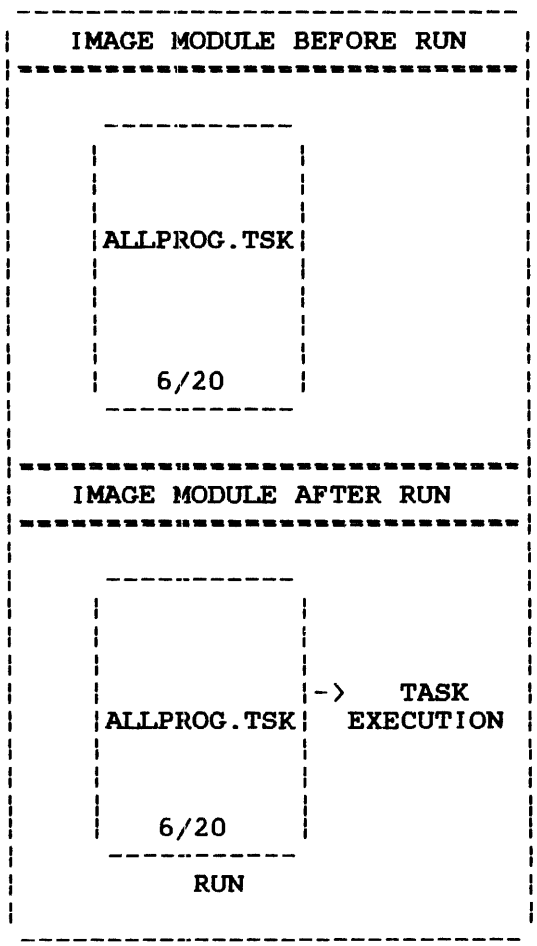


Figure 4-10 RUN Command Function in the Multimodule Environment

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 RTLs 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                          Create FORTRAN language
** NEW PROGRAM                     environment with the
-EDIT                               FORT language command

   .                                Specify TEST as filename
   .                                to be allocated
   .                                FORT command loads and
(edit session)                      starts editor with
   .                                TEST.FTN as current
   .                                program
   .
SAVE*
>END
```

```
*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
                                        Reexecute
** 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 multimodule 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 multimodule 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 multimodule environment
*EXEC
-MACRO:YSUB.MAC                         YSUB.MAC object is outdated
                                        Expand, assemble and link-edit
-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 job streams, MTM queues the requests until a job stream becomes available.

The batch queue is an indexed file containing the file descriptors (fds) 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 on an 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 (see Chapter 2), program development commands (see Chapter 4), batch processing commands and calls to a CSS file (see Chapter 6). A batch job file is not a CSS. This invalidates CSS commands, with the exception of $IF..., $ELSE and $ENDC. Any command that can be used at a terminal can be used in the batch job file.

Examples:

The following is an example of a single batch job file:

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

CSS can be used to build a batch job file and submit a job as follows:

```
** 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:

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.
fd_1	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, task-loaded and task-executing modes.

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:

All jobs with the user account number are displayed with the following:

```
INQ
```

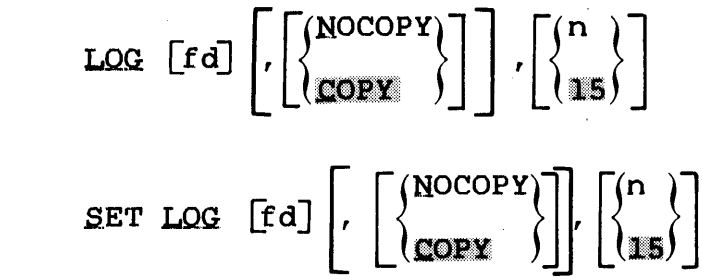
The status of TASK.JOB is displayed with the following:

```
INQUIRE TASK.JOB
```

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:



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 exclusive write-only (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.
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.
COPY	specifies that all output is 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 can be done on any type of file. For contiguous files, however, the checkpoint operation is treated as a no-operation. On nonbuffered indexed and extendable contiguous files, the checkpointing 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 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:

The following will purge TASK.JOB.

```
PURGE  TASK.JOB
```

```
 ---------
| SIGNOFF |
 ---------
```

5.2.4 SIGNOFF Command

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

Format:

SIGNOFF

Functional Details:

When a batch job user signs off the system, these messages are output to the log device or file:

```
ELAPSED TIME=hh:mm:ss          PROCESSOR=hh:mm:ss:mmm TSK-ELAPSED=hh:mm:ss
SIGNON LEFT=hh:mm:ss           PROCESSOR LEFT=hh:mm:ss
TIME OFF=mm/dd/yy  hh:mm:ss
```

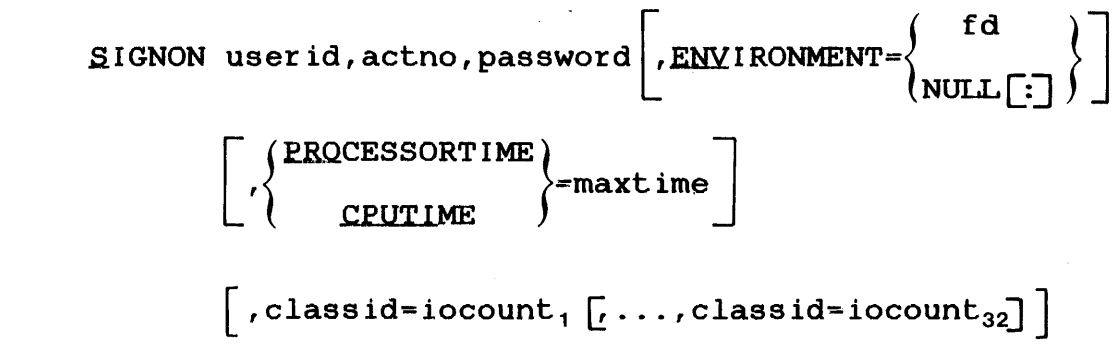
The SIGNOFF command can be entered in command, task-loaded and task-executing modes.

If no signon or processor time limits are established in the authorized user file (AUF) for the account under which the batch job is signed on, the SIGNON LEFT and PROCESSOR LEFT messages are not output.

5.2.5 SIGNON Command

SIGNON must be the first command in a batch job.

Format:



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 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 on-line 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.

PROCESSORTIME= maxtime is a decimal number specifying the CPUTIME= maximum processor time to which the batch job is limited. Processor time in a Model 3200MPS System is central processing unit and auxiliary processing unit (CPU+APU) time, whereas in a uniprocessor system, processor time is only CPU time. If this parameter is omitted, the default established at system generation (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 input/output (I/O) transfers associated with the 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. When ENVIRONMENT=NULL is specified, the colon is optional. This allows the user 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 is ignored if the user does not have the ENVIRONMENT AT SIGNON privilege.

Examples:

```
SIGNON  ME

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

S  ME,CPUTIME=120

S  ME,ENV=NULL,PROCESSORTIME=120

S  ME,ENV=XYZ
```

```
 ---------
| SUBMIT  |
 ---------
```

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=	is a decimal number that specifies the priority at which a batch job will run. The range of valid priority numbers is dependent upon the user's account privileges, sysgen options and MTM priority. The maximum range allowable is MTM 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 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 priority and the default batch priority established at MTM sysgen time. The user task (u-task) priorities are established at Link time and can be reset with the PRIORITY parameter of the SUBMIT command. Interactive tasks run at a default priority that is 12 priorities lower than MTM. Batch jobs run at a default priority that is 12 lower than MTM plus the value specified at MTM sysgen time. If the MTM sysgen priority is set to equal 1 and MTM 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, task-loaded and task-executing modes.

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
SIGNOFF
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 can later 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. The job is terminated unless the batch task pause option was enabled at MTM sysgen (see Section 5.5). When a batch task is completed, 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 immediately cancelled by MTM. MTM logs the following message to the system console if a batch task enters the paused state:

```
hh:mm:ss      .MTM > taskid      BTCH TSK PAUSED
```

In this message, taskid 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 CSS is an extension of the OS/32 command language. It allows the user to establish files consisting of sequences of commands ranging from elementary to extremely complex. It also incorporates such features as parameter substitution, variables, decision making and branching.

Once a CSS file has been established, it can be executed by entering a single command from a multi-terminal monitor (MTM) terminal, or by calling it from another CSS. Parameters can be passed from one CSS level to the next.

Through CSS, even the most complex operations can be reduced to a simple sequence of commands or even a single command.

The features available to users via CSS are:

- 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 or keywords take on specific meaning when the parameters are substituted 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 stored as a deck of cards, on a magnetic tape or as 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 ANYFILE.EXT,CO,40
AS 1,OTHRFILE.EXT
AS 2,ANYFILE.EXT;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 ESTABLISHING A COMMAND SUBSTITUTION SYSTEM (CSS) FILE

Since CSS files differ from other files in content only, they can be created just as a source or data file would be; i.e., by using the BUILD and ENDB commands or via an editor.

Both of the following sequences demonstrate the creation of the same CSS file:

```
*BUILD DEMO
B>LOAD ANYTASK,20
B>ASSIGN 1,INFILE.IN
B>ASSIGN 2,OUTFILE.OUT
B>ASSIGN 3,CON:
B>START
B>$EXIT
B>ENDB
*
*LO EDIT32
*ST
-PERKIN-ELMER OS/32 EDIT 03-145 R03-01
->APPEND
-    1   ->LOAD ANYTASK,20
-    2   ->ASSIGN 1,INFILE.IN
-    3   ->ASSIGN 2,OUTFILE.OUT
-    4   ->ASSIGN 3,CON:
-    5   ->START
-    6   ->$EXIT
-    7   ->
->SAVE DEMO.CSS
->END
-RAY    -END OF TASK CODE=  0  PROCESSOR=0.037/0.042
```

The advantage with using an editor is that any mistakes can be rectified immediately. Using BUILD and ENDB saves a few steps for users not prone to errors, and also supplies the default extension of .CSS. BUILD and ENDB are very useful for establishing short files. No matter which approach is taken, the last command in every CSS file should be the $EXIT command.

The extension .CSS is not required when a CSS is named. If another extension is used, it must be specified when the CSS is called.

6.3 CALLING A COMMAND SUBSTITUTION SYSTEM (CSS) FILE

A CSS file is called and executed from the terminal by specifying the file descriptor (fd) of the CSS file. The demonstration CSS established in the previous section would be executed by the following entry:

```
*DEMO
```

If only the filename is specified, MTM assumes the extension .CSS and first searches the user default volume in the user's private account. If CSS volume is enabled (see Volume Command), then the users CSS volume, private account is searched. If the CSS account (see SET CSS Command) is different from the private account, then the CSS volume, CSS account will be searched. If the file is not found, the system account (on the system volume) is searched. If the volume name or account class is specified by the user, no search of the system account is made. If the CSS file resides on a volume other than the user default volume (and is not a system CSS) the volume name must be supplied.

In summary, the order in which MTM searches for a CSS is as follows:

Current volume, current private account

If the CSS volume is not set to a '*',

CSS volume, current private account
CSS volume, CSS account

If CSS volume is not set,

Current volume, CSS account

Finally,

System volume, system account

For example, if a CSS was called in the following manner:

```
*M301:DEMO
```

MTM would search for the CSS in this order:

```
M301:DEMO.CSS/P
M301:DEMO.CSS/CSS ACCOUNT
M301:DEMO.CSS/SYSTEM ACCOUNT
```

If a CSS was called in this manner:

```
*DEMO/P
```

MTM searches for the CSS in this order:

```
user:DEMO.CSS/P
cssvol:DEMO.CSS/P
system:DEMO.CSS/P
```

If a CSS was called in the following manner:

```
*DEMO
```

MTM searches for the CSS in this order:

If the CSS volume is not set to a '*',

```
CSS:DEMO.CSS/P
CSS:DEMO.CSS/CSS ACCOUNT
```

If a CSS volume is set to a '*',

```
user:DEMO.CSS/CSS ACCOUNT
system:DEMO.CSS/S
```

It is important to remember that MTM will only test any given combination of volume, filename and account once, even if the same fd is built using different parameters.

Specification of a volume name also allows the user to call CSS files that belong to the user's private account, but do not reside on the current default user volume, or CSS volume.

If a CSS file is saved under an fd with an extension other than .CSS, the extension must be specified when the CSS is called. For example, a CSS file saved as CSSFILE.ANY would be called by the following entry:

```
*CSSFILE.ANY
```

CSS calls can also be made directly into the system or group accounts by appending the appropriate account type to the fd, as follows:

DIVE	CSS call to private file
CLIMB/G	CSS call to group file
ROLL/S	CSS call to system file
TUMBLE/actno	CSS call to file on another account

NOTE

A user's account must have EXTENDED ACCOUNT ACCESS privileges to use an account number when calling a CSS routine.

A CSS can call another CSS file. The level to which CSS calls can be nested is set at MTM system generation (sysgen). If this maximum number is exceeded, an error message is displayed and all active CSSs are cleared.

A user must have the CSS privilege in order to call CSS files in the user's private account or group. If not privileged, the user can only call system CSSs. If the user also has the privilege to specify account numbers instead of classes, the user can call a CSS in any account. If the leading characters of a CSS fd are the same as a command, MTM assumes a command, but if the .CSS extension is entered, the CSS will be assumed.

Examples:

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

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

Example:

```
M300:CLO
AS3.CSS
```

6.4 USE OF PARAMETERS

Parameters can be passed to a CSS when it is called. There are two types of parameters available to the user, positional parameters and keyword parameters. These parameters will be explained later in this section.

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 a comma. 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.

Examples:

In the following example, the DEMO.CSS file is called. Two positional parameters are passed. The first is A, the second is "B,C".

```
DEMO A,"B,C"
```

In this example, three parameters (two positional and one keyword) are passed. The first positional parameter is A, the second is null. The parameter passed via the keyword ANYTHING is C.

```
DEMO2 A,,ANYTHING=C
```

6.4.1 Positional Parameters

Within a CSS file, a positional 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; it refers to the name by which the CSS is called. 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).

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.

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

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

One CSS file can call another. Within a CSS, 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. In a CSS file, @@1 refers to the first parameter of the calling file.

Example:

Within a file named PROC.CSS there is a call to another CSS file named SETUP.CSS. The PROC.CSS file consists of the following:

```
PREVENT PROMPT;PRE ETM
$COPY
LOAD @1,@2
SETUP
START,@3
$NOCOPY
ENABLE PR;ENA ETM
$EXIT
```

SETUP.CSS contains:

```
XAL @@1.OUT,IN,132
AS 6,@@1.IN
AS 7,@@1.OUT
$EXIT
```

The call to PROC.CSS shown below would produce the subsequent display at the user's terminal.

```
*PROC ANYTASK,20,"COM=CON:,LIST=PR:"
-LOAD ANYTASK,20
SETUP
XAL ANYTASK.OUT,IN,132
AS 6,ANYTASK.IN
AS 7,ANYTASK.OUT
$EXIT
START,COM=CON:,LIST=PR:
$NOCOPY
```

There can be as many @ symbols used to reference higher level CSSs as there are allowable CSS levels. If the SETUP.CSS file from the previous example contained a call to another CSS file, the CSS called by SETUP.CSS could refer to the parameters passed to the PROC.CSS file as @@@1, @@@2 and @@@3.

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

Parameter @0 (or @@0) is a special parameter used to reference the name of the CSS file in which it is contained. Parameter @0 (or @@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 the following commands:

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

If this file is called with the filename CR, logical unit 1 (lul) is assigned to the card reader (CR:). The executed form of the ASSIGN command becomes:

```
AS 1,CR:
```

If this file were called with the filename MAG1, the result is:

```
AS 1,MAG1:
```

It should be remembered that a CSS file is called by (as a minimum) the filename portion of the fd under which it is stored; no flexibility for calling statements should be inferred from the example above.

To further illustrate the use of the @0 parameter, the PROC.CSS file from previous examples could easily be expanded to contain the following:

```
XAL @0.LOG,IN
LOG @0.LOG,NOCOPY
  .
  .
  .
LOG CON:
$EXIT
```

6.4.2 Keyword Parameters

The CSS language also provides a means of passing parameters via keywords in a CSS call. A straightforward substitution procedure is applied. Keywords enable the user to specify a value that is subsequently substituted for each reference to the keyword encountered within the CSS. The value of a keyword is defined in the CSS call in the following format.

Format:

```
keyword keyoperator [parameter]
```

Parameters:

keyword	is the 1- to 8-character name of a keyword. The characters must be alphabetic.
keyoperator	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). The equal sign (=) is the default keyoperator.
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 (CR) 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. If a user wishes 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 prior to any keywords. Positional parameters and keywords must be separated by 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 are not valid.
TEST B,,=HOLL	Keyword name is missing.
TEST A'='B,C'=D	Second quote is not matched.

Within a CSS file, a keyword parameter is referred to by the use of the @= symbols (similar to the @ symbol usage for positional parameters). As with positional parameters, multiple @ symbols can be used to refer to the keyword parameters of higher level CSS files.

Format:

```
[@[@...@]]@=/[keyword]/
```

Parameters:

@=	is the symbol that 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.

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, use of a keyword in the following manner:

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

indicates that 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 nonexisting keywords or to higher CSS levels which do not exist are not expanded. The same applies to references without a keyword. References with a keyword expand in the usual manner. The following examples show the result of using keyword references in a CSS file and then passing keyword parameters in the CSS call. 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 be passed as part of the keyword value as long as they are bracketed by double quotes. Similarly, double quotes can be passed as long as they are enclosed within single quotes.

Examples:

The following listing presents an example CSS that uses keywords:

```
| XAL @=/O.UTPUT/,IN,66
  XAL @=/W.ORK/,IN,66
  LO SRTMRGII,50
  AS 1,@=/I.NPUT/
| AS 2,@=/O.UTPUT/
  AS 4,@=/W.ORK/
  AS 5,CON:
  AS 9,MTM:SRTMRGII.OVY/S
  START
  $EXIT
```

The CSS listed above can be executed by any of the three calls listed below (provided the CSS had been saved as SORT.CSS):

```
SORT INPUT=DEMO.IN,OUTPUT=DEMO.OUT,WORK=DEMO.WRK

SORT IN=DEMO.IN,OUT=DEMO.OUT,WO=DEMO.WRK

SORT W=DEMO.WRK,O=DEMO.OUT,I=DEMO.IN
```

The executed form of the CSS would be the listing below.

```
XAL DEMO.OUT,IN,66
XAL DEMO.WRK,IN,66
LO SRTMRGII,50
AS 1,DEMO.IN
AS 2,DEMO.OUT
AS 4,DEMO.WRK
AS 5,CON:
AS 9,MTM:SRTMRGII.OVY/S
START
$EXIT
```

6.5 USE OF VARIABLES

MTM and batch users can allocate a predetermined number of variables to be used within a CSS. The maximum number of variables that can be allocated by a user is set at MTM sysgen. In general there are two types of variables: those that exist from signon to signoff, and those that only exist while the defining CSS is active. There are now further distinctions between the types of variables available with MTM.

6.5.1 Types of Variables

There are four types of variables available to MTM users:

- global variables,
- local variables,
- new global variables, and
- new internal variables.

The first two types, global and local variables, should be familiar to all users of releases of MTM prior to R06.2. Global variables exist from signon to signoff or until they are freed via the $FREE command. Local variables can only be used while the CSS in which they are defined is active. 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 similar to the local and global variables in terms of usage. The way in which they are defined and 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, until they are released via the $RELEASE command, or until assigned an undefined value by the $DEFINE command. 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 while the CSS is active. New internal variables are released when the highest level CSS exits (back to MTM command mode). They are global within CSS calls.

Example:

```
FIRST.CSS
$WR IN FIRST
$DEF 1,,STR(IN FIRST)
Y; *make a call to Y
$WR @*1
$EXIT
```

```
SECOND.CSS
$WR IN SECOND
$DEF 1,,STR(came from second)
$EXIT
```

The following call

```
*FIRST.CSS
```

produces this output:

```
IN FIRST
IN SECOND
came from second
```

The user can 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 (maximum of 99).

NOTE

Users should be familiar with the use of both new global and new internal variables. These variable types will eventually replace the local or global variables usage.

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; the remaining characters can be alphanumeric.

Examples:

```
@LKL1

@GLBL5

@ANYTHING
```

Local variables are named via the $LOCAL command. Global variables are named via the $GLOBAL command. Values can be assigned to predefined local and global variables via the $SET command. These commands are discussed in detail later in this chapter. Their basic use is shown in the following example:

```
$LOCAL @SEGSIZE;$GLOBAL @PRINTDEV
$SET @SEGSIZE=@3;$SET @PRINTDEV=@4
LOAD SOMETASK, @SEGSIZE
OTHERCSS
$EXIT
```

Once a global variable has been defined, it can be referred to by any CSS file called between the time it is defined and the time the user signs off or frees the variable. A local variable can only be referred to within the CSS in which it is defined. To illustrate, the CSS file named OTHERCSS called in the example above could refer to the global variable @PRINTDEV. OTHERCSS could not, however, refer to the local variable @SEGSIZE unless it was specifically defined within OTHERCSS.

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 alphanumeric.

Examples:

```
INTRNL1

GLBL12

WHATEVER
```

New global and new internal variables are named via the $DEFINE command and, at that time, are associated with a decimal number. The number associated with the variable is for reference purposes; it has nothing to do with the value of the variable. Because new global and internal variables are associated with numbers, the use of names is optional.

The $DEFINE command provides the user with many options for establishing and assigning values to new global and new internal variables.

$DEFINE is an extremely flexible and useful command; the following examples represent only basic implementations.

The following example establishes a new global variable. The decimal reference number associated with the variable is 7, its name is GSYSVOL and the value assigned to it is the name of the default system volume.

```
$DEFINE GVARIABLE 1, GSYSVOL, DVOLUMENAME (SYSTEM)
```

The following example establishes new global variable number 2. This variable is not given a name; it is assigned the value of the userid under which the terminal user or batch job is signed on.

```
$DEF G 2,,CURRENT (USERNAME)
```

The following example establishes new internal variable number 7, gives it the name LSTDEV and assigns it the string value CON:

```
$DEF IVAR 7,LSTDEV, STRING (CON:)
```

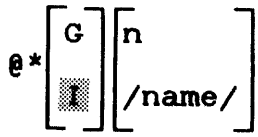
In this example, a new internal variable is established (by default) and associated with the number two. It is given no name, and is identified as a required variable.

```
$DEFINE 2,,REQUIRED
```

Once defined, the variable can be referred to 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 format can be used.

  Format:

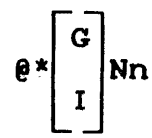
  

  Where:

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.

  Format:

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:

The following example references global variable number 3.

```
@*G3
```

The following example references the internal variable name VOLUME.

```
@*/VOLUME/
```

The following example references the name of internal variable number 3.

```
@*N3
```

6.5.4 Command Substitution System (CSS) Line Expansion

The MTM preprocessor expands the entire CSS line in one step. Because of this, 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 COMMAND SUBSTITUTION SYSTEM (CSS) FILE

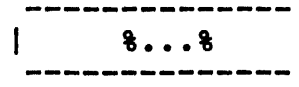
All of the MTM supported commands can be used in a CSS file (see Chapter 2), as well as a number of commands specifically associated with the CSS Facility. Several of these commands can also be used in the command mode. For example, all global variables can be defined and assigned a value from the users console. The $COPY and $NOCOPY commands can also be used in command mode.

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. See 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:

$$\%\left\{\begin{array}{l} \text{char1char2}_1\ [\text{char1char2}_2 \ldots \text{char1char2}_4]\ \% \\ \%\ \text{new delimiter} \end{array}\right\}$$

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 that 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 examples in Table 6-1 are used to illustrate the basic functionality of the character replacement command. The uses of this command are not limited to those shown in the table. The command becomes extremely powerful as the user introduces more involved substitution and replacement within the same line.

TABLE 6-1 EXAMPLES USING THE CHARACTER REPLACEMENT COMMAND

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:

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.

```
$BUILD TEST

%%%\\*@\%%+@%$WR @1
$EX
$ENDB
```

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
```

Replacing 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 the + is replaced by an @ symbol, according to the second character replacement specification. The line now looks like this:

```
$WR @3
```

Replacing 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:

```
          { fd }[,APPEND]
$BUILD    { lu }
    .
    .
    .
$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.

Examples:

The example CSS from section 6.4.1 can be altered to illustrate the use of the $BUILD and $ENDB commands:

```
$BUILD SORT.CMD
  KEY @=/K.EYS/
  SORT @=/I.NPUT/ > SORT.OUT
  END
$ENDB
XAL SORT.OUT,IN,66
LO SRTMRGII,20
AS 1,@=/I.NPUT/
AS 2,SORT.OUT
AS 3,CON:
TEMP 4,IN,5/3
AS 5,SORT.CMD
AS 9,MTM:SRTMRGII.OVY/S
ST
DEL SORT.CMD
$EXIT
```

The following version of the example CSS demonstrates the use of the lu parameter of the $BUILD command.

```
XAL SORT.OUT,IN,66
LO SRTMRGII,20
AS 1,@=/I.NPUT/
AS 2,SORT.OUT
AS 3,CON:
TEMP 4,IN,5/3
$BUILD 5
  KEY @=/K.EYS/
  SORT @=/I.NPUT/ >SORT.OUT
  END
$ENDB
AS 9,MTM:SRTMRGII.OVY/S
ST
$EXIT
```

```
 --------------
|    $CLEAR    |
 --------------
```

6.6.3 $CLEAR Command

The $CLEAR command terminates a CSS stream, closes all CSS files and returns to MTM command level.

Format:

$CLEAR

Functional Detail:

The $CLEAR command can be entered in command, task-loaded and task-executing modes.

Example:

The following CSS is called from another CSS routine:

```
$JOB
  LOAD SOMETASK
  START
  $IFNE 0
     $WR PROCESS ABORTED - SOMETASK
     $CLEAR
  $ENDC
  LOAD NEXTASK
  START
  $IFNE 0
     $WR PROCESS ABORTED - NEXTASK
     $CLEAR
  $ELSE
     $WR PROCESS COMPLETE
  $ENDC
$TERMJOB
$EXIT
```

If either SOMETASK or NEXTASK ends with an end of task code other than 0, all CSS levels will be exited; otherwise, CSS execution will return to the calling CSS.

6.6.4 $CONTINUE Command

The $CONTINUE command resumes execution of a CSS procedure suspended by a $PAUSE or $WAIT command.

Format:

$CONTINUE

Example:

In this example the $CONTINUE command is entered in command mode. The name of the CSS used is EXMPL.CSS.

```
$WR This CSS will pause after this message and will
$WR only resume processing after you enter $CON.
$PAUSE
$WR  This is the end of EXMPL.CSS.
$EXIT

*EXMPL

    This CSS will pause after this message and will
    only resume processing after you enter $CON.

*$CON

    This is the end of EXMPL.CSS.

*
```

```
 --------------
|  $COPY AND   |
|   $NOCOPY    |
 --------------
```

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. 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

Example:

This example illustrates the use of the $COPY command and the $NOCOPY command. The CSS, LOADER.CSS, loads and starts a task. The $COPY command is set for the first execution of LOADER.CSS. The $COPY command is turned off with the $NOCOPY command and LOADER.CSS is run again.

```
LOAD @1
ASSIGN 1,CON:
START
$EXIT
```

Execution of LOADER.CSS with $COPY used in MTM command mode.

```
*$COPY
*LOADER FORT2
LOAD FORT2
ASSIGN 1,CON:
START
 THIS FORTRAN PROGRAM WILL DISPLAY
 THIS MESSAGE ON THE TERMINAL SCREEN
STOP
ELAINE  -END OF TASK CODE=    0     CPUTIME=0.012/0.013
$EXIT
```

Execution of LOADER.CSS with $NOCOPY used in MTM command mode.

```
*$NOCOPY

*LOADER FORT2

 THIS FORTRAN PROGRAM WILL DISPLAY
 THIS MESSAGE ON THE TERMINAL SCREEN
```

6.6.6 $DEFINE Command

The $DEFINE command is used to define or to redefine new global or new internal variables.

Format:

$$\$DEFINE \left\{\begin{matrix}GVARIABLE\\IVARIABLE\end{matrix}\right\} n\ ,[name],operator_1\ [operator_2\ ...operator_n]$$

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 one to eight characters long and can consist of any character A through Z or any number 0 through 9.

operator
operator ...operator

is one or more of the following operators, which select a particular function to be performed to determine the variable's value.

fd 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
- POSITION
- REQUIRED
- SEARCH
- STRING
- SUBSTRING

The following sections define the format and function of each of these operators within the $DEFINE command.

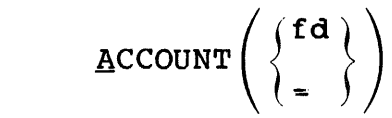
6.6.6.1 File Descriptor (fd) Operators

The following four operators can be used to determine the account, filename, extension or volume name of a specified fd and 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 fd and assign the designator as the value of the variable being defined.

Format:



Parameters:

fd	is the file descriptor of the file or device for which the account designator is to be assigned as the value of the variable.
=	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.

Examples:

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 the $WR @*6 command is:

```
/G
```

6.6.6.1.2 EXTENSION Operator

The EXTENSION operator of the $DEFINE command enables the user to assign the extension of a given fd as the value of the variable being defined.

Format:

Parameters:

fd	is the file descriptor of the file or device for which the extension is to be assigned as the value of the variable.
=	the current total result for this $DEFINE command is used to determine the extension.

Functional Detail:

The returned value will contain a leading period if an extension was specified; otherwise, the value of the variable is undefined.

Example:

The following CSS is built:

```
BUILD TEST
$DEFINE 10,,EXTENSION(@1)
$WR @*10
$EX
ENDB
```

When called with the following CSS call:

```
TEST FORTRAN.FTN
```

the $WR @*10 command would output .FTN.

6.6.6.1.3 FILENAME Operator

The FILENAME operator of the $DEFINE command enables the user to assign the filename of a given fd as the value of the defined variable.

Format:

FILENAME({ fd })
         ({ = })

Parameters:

fd	is the file descriptor or device for which the filename is to be assigned as the value of the variable.
=	the current total result for this $DEFINE command is used to determine the filename.

Functional Details:

If an fd 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.

Examples:

The following CSS is built:

```
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.

The FILENAME operator of the $DEFINE command of the CSS is used in the following example:

```
$DEF 1,,F(@=/I.NPUT/)
XAL @*1.OUT,IN,66;  XAL @*1.LST,IN,65
LO SRTMRGII,20
AS 1,@=/I.NPUT/
AS 2,@*1.OUT
AS 3,@*1.LST
TE 4,IN,5/3
$B 5
  KEY @=/K.EYS/
  SORT @=/I.NPUT/ > @*1.OUT
  END
$ENDB
AS 9,MTM:SRTMRGII.OVY/S
ST
$EXIT
```

6.6.6.1.4 VOLUMENAME Operator

The VOLUMENAME operator of the $DEFINE command enables the user to assign the volume name of a given fd to the variable being defined.

Format:

$$\underline{V}\text{OLUMENAME}\left(\left\{\begin{matrix}\text{fd}\\ =\end{matrix}\right\}\right)$$

Parameters:

fd	is a file descriptor of the file for which the volume name is to be assigned as the value of the variable.
=	the current total result for this $DEFINE is used to determine the volume name.

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:

The following CSS is built:

```
BUILD TEST.CSS
$DEFINE 20,, VOLUMENAME (M301:SOURCE.FTN)
$WR @*20
$EX
ENDB
```

Calling the above CSS with the following call:

```
*TEST
```

| the output from $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:

GO { n / $name }

L { D / U } { n / $name }

T { D / U } { n / $name }

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. • 0 indicates exit the $DEFINE command. • 1-999 indicates 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:

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.

```
BUILD TEST.CSS
$DEFINE 5,, ST (@1) EXT (=) LU2 CL(L) GOO ST(.FTN)
$WR @*5
$EX
ENDB
```

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:

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:

-033 becomes the value of variable 7 (referenced as @*7). The default number of digits (4) is used.

```
$DEFINE 7,,DCOMPUTE (-33)
```

-00004 becomes the value of variable 4 (referenced as @*4). The number of digits in the result is defined as 6.

```
$DEFINE 4,,DCOMPUTE (6,-2+5/-2*4)
```

+232 becomes the value of variable 5 (referenced as @*5). This is determined by multiplying the value of variable 4 (referenced as @*4), which is defined above as -4 with the value of variable 7 (referenced as @*7), which is defined above as -33, then adding 100 to the result. The default number of digits (4) is used.

```
$DEFINE 5,,DC(@*4*@*7+100)
```

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:

$$\text{DHCONVERT}\left(\left[\left\{\begin{matrix}\#\text{digits}\\ 4\end{matrix}\right\},\right]\text{operand}_0\left[\left[\text{operator}_1\ \text{operand}_1\right]\left[\text{operator}_n\ \text{operand}_n\right]\right]\right)$$

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. Absolute Y'OFFFFFFF' is the maximum value allowed.

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:

In the following example, the value of variable 7 (@*7) becomes hexadecimal 0021.

```
$DEFINE 7,,DHCONVERT(+33)
```

In the following example, the value of variable 4 (@*4) becomes a hexadecimal 00000C.

```
$DEFINE 4,,DHCONVERT (6,2+5/2*4)
```

In the following example, the value of variable 5 becomes a hexadecimal 00B8. (4x21)+100 = 184 = 00B8 in hex.

```
$DEFINE 5,,DHCONVERT (4*@+7+100)
```

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:

HCOMPUTE([{#digits / 4},] operand$_0$ [[operator$_1$ operand$_1$] [operator$_n$ operand$_n$]])

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:

$$\text{HDCONVERT}\left(\left[\left\{\begin{matrix}\#\text{digits} \\ 4\end{matrix}\right\},\right]\text{operand}\ \left[\left[\text{operator}_1\ \text{operand}_1\right]\ \left[\text{operator}_n\ \text{operand}_n\right]\right]\right)$$

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'OFFFFFFF'.
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'OFFFFFFF'. 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,CO/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:

$$\underline{C}\text{LEAR}\left(\left\{\begin{matrix}\text{L}\\ \text{T}\end{matrix}\right\}\right)$$

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.

Examples:

The following is an example of how to add the default extension .FTN to a fd. The fd passed in the CSS call is allowed with or without an extension.

```
BUILD TEST.CSS
$DEF 5,,VOL(@1) FI(@1) EXT(@1) LD1 STR (@1.FTN) ACC(@1)
$WR @*5
$EX
ENDB
```

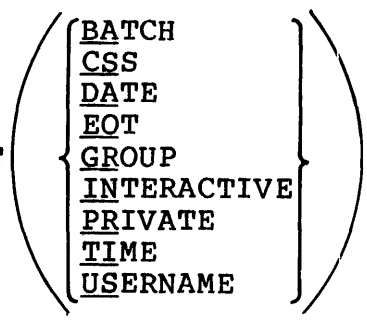
This example CSS tests to see if an extension is included in the CSS call. If an extension is specified, it is not changed. If no extension is specified, the default extension .FTN is added. If this CSS was called with the following, the results would be as follows:

```
V M301

TEST SYS:ABC/P          The result @*5 = SYS:ABC.FTN/P
TEST BBBB.XYZ           The result @*5 = M301:BBBB.XYZ/P
```

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:



Parameters:

BATCH	in batch mode, the value returned is the batch job fd; in interactive mode the value is undefined.
CSS	the value returned is the 5-digit, current CSS account number with leading zeros.
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 sysgen.

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 five 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 five 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:

Execution of the following CSS will cause the current time to be output.

```
BUILD TIME.CSS
$DEFINE 5,,CURRENT(TIME)
$WR @*5
$EX
ENDB
```

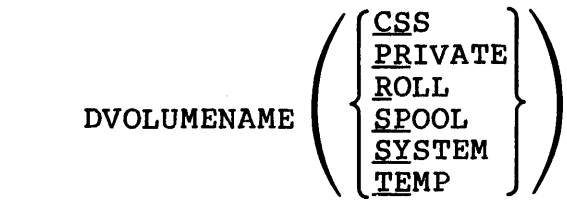
The following statement returns the private, group and CSS accounts.

```
$DEF 1,,CURRENT(PRIVATE) CURRENT(GROUP) CURRENT(CSS)
```

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:



Parameters:

CSS	returns the volume name of the user's CSS volume. If CSS processing is disabled, a null value is returned.
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 Detail:

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)
```

Reference by variable would return SCRT:

```
$WR @*6
```

Reference by variable name would also return SCRT:

```
$WR @*/TEMPVOL/
```

The following statement returns the private and CSS volume or a CSS undefined message if there isn't a CSS volume.

```
$DEF 1,,DVOLUME(PRIVATE) DVOLUME(CSS) LDO STR(CSS UNDEFINED)
```

6.6.6.4.4 POSITION Operator

The POSITION operator will return the position of a substring in a given search string.

Format:

```
$DEFINE POSITION    ('delimeter',substring,searchstring)
```

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.

substring	is the string being looked for in the 'searchstring'.
searchstring	is the string to be searched for the first occurrence of 'substring'.

Example:

```
BUILD PROG1
$DEFINE 1,FRED,POSITION('/',USER3/,USER1,USER2,USER3,USER4/)
$WRITE @*1
END
```

Call the above CSS with the following call:

```
*PROG1
```

The resulting output of the $WRITE @*1 statement is:

```
000013
```

```
BUILD PROG2
$DEFINE 1,FRED,POSITION('/',USER6/,USER1,USER2,USER3,USER4/) LD0 STR(NOT FOUND)
$WRITE @*1
ENDB
```

Call the above CSS with the following call:

```
*PROG2
```

The resulting output of the $WRITE @*1 statement is:

```
NOT FOUND
```

NOTE

Positions in the searchstring start at 1. If the string is not found, a null string (undefined operation) is returned. The comma is required after the first '/' and the second '/', and a ')' must follow the third. Commas between the second and third '/' are treated as text.

6.6.6.4.5 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 can 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:

The above CSS identifies three new internal variables (3, 4 and 5) as required variables.

```
BUILD TEST.CSS
$DEFINE 3,LISTDEV,REQUIRED
$DEFINE 4,OPTION,REQUIRED (NEWNAME)
$DEFINE 5,,REQUIRED
$EXIT
ENDB
```

If this CSS is called as follows, the following message prompts will be issued at the user's 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.

6.6.6.4.6 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:

$$\underline{\text{SEARCH}}\ \text{delimiter}_1 \left\{ \begin{matrix} \text{'}d_2\text{'} \\ \text{'}d_2\text{+} \end{matrix} \right\}, [\text{keyword}_1\ [\text{keyword}_2\text{'...'keyword}_n]], [\text{string}_1\ [d_2\text{string}_2]] \text{delimiter}_1$$

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 used to separate the strings to be searched. The string delimiter can be any character except CR or semicolon. If the 'd ' option is used, the delimiter (d_2) following the matched string is not included when the string is moved. If the 'd +' 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 can 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 that can contain any character except CR 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
```

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
```

Example:

The following 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.

```
BUILD TEST.CSS
$DEF 5,,SEARCH('#',OP.TION'BA.TCH, @1)
$WR @*5
$EX
ENDB
```

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. The 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 searched (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.

6.6.6.4.7 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

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 that can contain any characters except CR 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:

The following CSS is built:

```
BUILD TEST
$DEFINE 7,, STRING (ABC) ST # A ($$) A#
$WR [@*7]
$EX
ENDB
```

Call the above CSS with the following call:

```
*TEST
```

The resulting output of the $WR @*7 statement is:

```
[ABC A ($$) A]
```

6.6.6.4.8 SUBSTRING Operator

The SUBSTRING operator will return a specified portion of the given string.

Format:

```
$DEFINE N,NAME,SUBSTRING  ('delimiter',string,*starting position,*length)
```

Parameters:

string	is the string from which the substring will be taken.
starting position	is the start of the substring. If the starting position is negative or greater than the length of the string, a parameter error will be given.

length	is the number of consecutive characters to include. If length is greater then the number of characters remaining in the string, only the remaining portion of the string is taken.
*	These parameters may be compound expressions that follow the same syntax and order of evaluation as argument 2 in the decimal compute (DC) $DEFINE function, (e.g. 10+2*6).

Example:

```
BUILD EXAMPL
$DEFINE 1,FRED,SUBSTRING(':',CT31:,2,2)
$WRITE @*1
ENDB
```

Call the above CSS with the following call:

```
*EXAMPL
```

The resulting output of the $WRITE @*1 statement is:

```
T3
```

```
BUILD EXMPL2
$DEFINE 1,POS,PO('/',./,FILE00123./); * returns a 10
$DEFINE 2,FRED,SUBSTRING('.',FILE00123.,@*1-5,5)
$WRITE @*2
ENDB
```

Call the above CSS with the following call:

```
*EXMPL2
```

The resulting output of the $WRITE @*2 statement is:

```
00123
```

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

Functional Detail:

The CSS processor must encounter a $EXIT command before it reaches the end of the CSS file; if it doesn't, an error message is generated.

Example:

This example illustrates the use of the $EXIT command in a CSS file and a called CSS file. The name of the main CSS is ONE.CSS and the name of the called CSS is TWO.CSS.

```
*** ONE.CSS ***
$WR  This is ONE.CSS.
TWO
$EXIT
$WR  This is after the exit command and will not print.
$EXIT


*** TWO.CSS ***
$WR This is TWO.CSS.
$EXIT
$WR This is after the exit command and will not print.
$EXIT

*ONE.CSS

This is ONE.CSS.
This is TWO.CSS.
```

| $FREE |

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:

$FREE varname$_1$ [,...,varname$_n$]

Parameter:

varname, ...varname$_n$ — is a 1- to 8-character name specifying the variable whose name and value are to be freed.

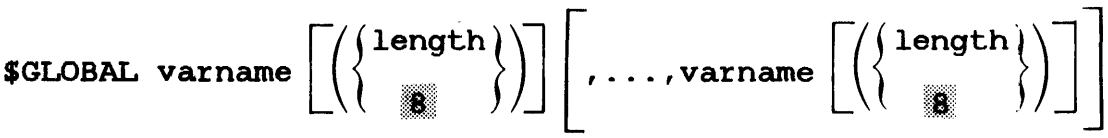
Example:

```
$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:



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 the end of a CSS job that contains all the user CSS commands.

Format:

```
$JOB [ { PROCESSORTIME } =maxtime ]
     [ { CPUTIME     }            ]
       [,classid=iocount1] [,...,classid=iocount32]
   .
   .
   .
$TERMJOB
```

Parameters:

PROCESSORTIME= CPUTIME=	maxtime is a decimal number specifying the maximum processor 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 input/output (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. They can be used, however, to 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. The user can specify central processing unit (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.

Example:

This example illustrates the use of the $JOB command and the $TERMJOB command. In the CSS, JOBTERM.CSS, the loading and starting of a task is delineated by the $JOB and $TERMJOB commands. If an error occurs, control passes to the next command after the $TERMJOB command. The commands following the $TERMJOB command determine the error message the user will receive.

```
$JOB
 LO @1
 ASSIGN 1,CON:
 START
 $EXIT
$TERMJOB
 $IFNX @1
    $IFNULL @1
     $WR ********************************************************
     $WR *                                                      *
     $WR *  You need to pass the filename of the task you want  *
     $WR *  to load as a parameter.  Please enter the CSS       *
     $WR *  name and the filename of the task you want to       *
     $WR *  load.                                               *
     $WR *                                                      *
     $WR ********************************************************
     $EXIT
     $ENDC
```

```
$WR **************************************************
$WR *                                                *
$WR *  The task you selected does not exist.  Please  *
$WR *  enter the CSS name and the correct filename.   *
$WR *                                                *
$WR **************************************************
$EXIT
$ENDC
```

The CSS jobterm is called without a filename parameter.

```
*JOBTERM
**********************************************************
*                                                        *
*  You need to pass the filename of the task you want    *
*  to load as a parameter.  Please enter the CSS         *
*  name and the filename of the task you want to         *
*  load.                                                 *
*                                                        *
**********************************************************
```

The CSS jobterm is called with a nonexistant filename.

```
*JOBTERM PAL
**********************************************************
*                                                        *
*  The task you selected does not exist. Please          *
*  enter the CSS name and the correct filename.          *
*                                                        *
**********************************************************
```

The CSS jobterm is called with a legitimate filename.

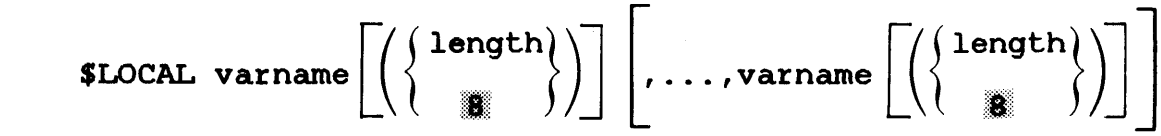
```
*JOBTERM FORT2

 THIS FORTRAN PROGRAM WILL DISPLAY
 THIS MESSAGE ON THE TERMINAL SCREEN
```

| $LOCAL |

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:



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 Detail:

When $PAUSE is entered, the CSS procedure remains suspended until the $CONTINUE command is continued or the $CLEAR command is entered to terminate a procedure suspended by a $PAUSE.

Example:

This example illustrates the use of the $PAUSE command. The $PAUSE command is issued within the CSS EXMPL.CSS. The $CONTINUE command is used to continue the process of the CSS.

```
$WR  This CSS will pause after this message and will
$WR  give you a command mode prompt.  To resume
$WR  processing, enter the command $CONTINUE.
$PAUSE
$WR  This is the end of EXMPL.CSS.
$EXIT
```

```
*EXMPL

    This CSS will pause after this message and will
    give you a command mode prompt.  To resume
    processing enter the command $CONTINUE.

*$CON

    This is the end of EXMPL.CSS.
```

| PRIOR |

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, 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.) 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 through 11 by a nonprivileged user), the invalid priority specification is ignored, no message is generated and the default MTM priority is used.

If the priority number specified causes the priority to be lower (i.e., a higher number) than 255, the task priority will default to 255.

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:

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

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	indicates that all variables (of the type selected via the preceding parameter) between the range n_1/n_2 be released. 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 can be entered in command, task-loaded, task-executing and CSS modes.

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:

All new global variables from 1 through 5 are released.

```
$RELEASE GVARIABLE, 1/5
```

The new internal variables numbered 16, 19, 18 and 25 are released.

```
$RELEASE IVARIABLE, 16, 19, 18, 25
```

All new internal variables are enclosed.

```
$RELEASE IVARIABLE, ALL
```

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'
```

| SET CODE |

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

Example:

This example illustrates the use of the $SKIP command. The CSS JOB.CSS will skip the section that loads and starts the task if a user has not entered the fd parameter.

```
$JOB
$IFNULL @1
  $SKIP
$ELSE
  LOAD @1
  ASSIGN 1,CON:
  START
  $EXIT
$ENDC
$TERMJOB
$WR ************************************************
$WR *                                              *
$WR *  You need to pass the fd of the task you want *
$WR *  to load and start as a parameter.           *
$WR *                                              *
$WR ************************************************
$EXIT
```

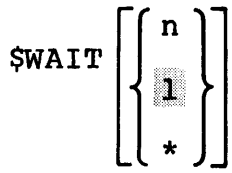
```
*JOB
***********************************************
*                                             *
* You need to pass the fd of the task you want *
* to load and start as a parameter.           *
*                                             *
***********************************************
```

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:



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.
*	is for console CSS compatibility and is treated as a nonoperation.

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 CR. The semicolon is not printed.

Example:

The following is an example of the $WRITE command.

```
        $WR  This sentence will print on the terminal screen.
        $WR  This sentence will also print on the terminal
        $WR  screen.;$EXIT
```

```
*WRITE

    This sentence will print on the terminal screen.
    This sentence will also print on the terminal
    screen.
```

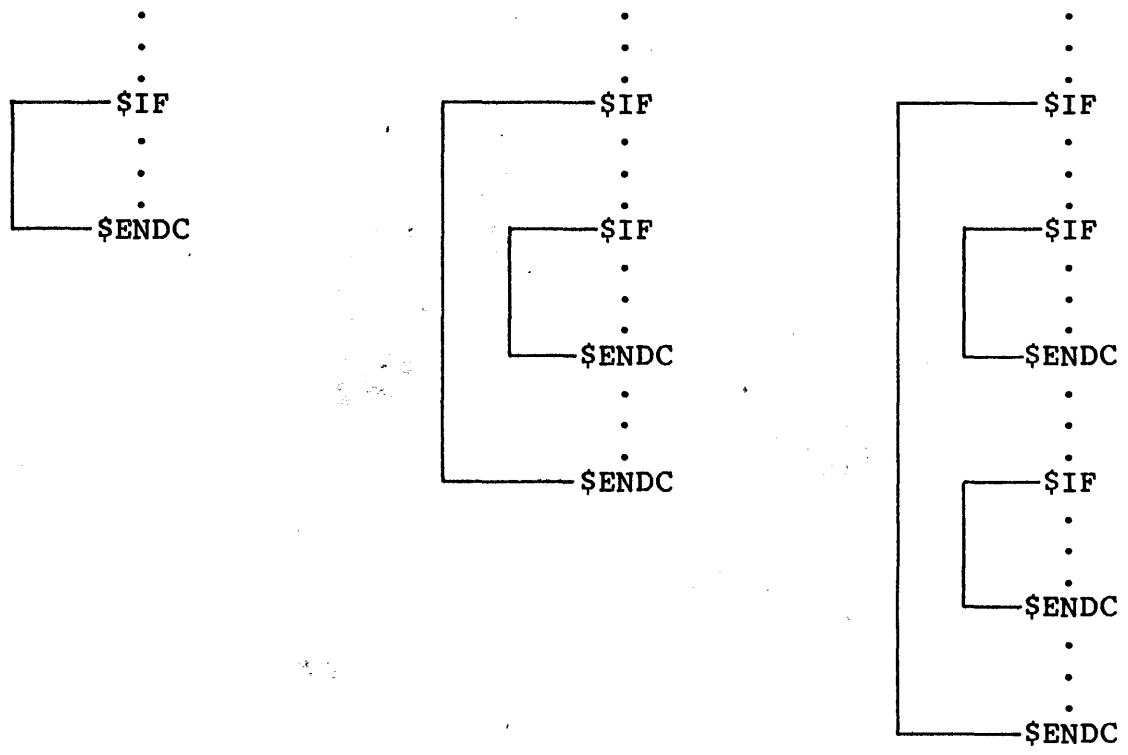
6.7 LOGICAL IF COMMANDS

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. They differ from previous CSS commands in that each one tests a specific built-in, defined condition rather than causing a specific action.

The logical IF commands fall into three categories:

- end of task code testing,
- file existence testing, and
- 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 (SVC3,n)	When any task terminates by executing the end of task program command (SVC3,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	Tests if end of task code is equal to n.
$IFNE n	Tests if end of task code is not equal to n.
$IFL n	Tests if end of task code is less than n.
$IFNL n	Tests if end of task code is not less than n.
$IFG n	Tests if end of task code is greater than n.
$IFNG n	Tests 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 end of file (EOF), a command error is generated.

6.7.2 File Existence Testing Commands

There are two commands dealing with file existence:

$IFX fd	Tests fd for existence.
$IFNX fd	Tests 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 fd 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 Tests if @n is null.

$IFNNULL @n Tests 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
```

This tests to insure parameters @1, @2 and @3 are not null. If any parameter is defined, the test 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

| $GOTO and $LABEL |

6.7.5 $GOTO and $LABEL Commands

The $GOTO command is used to skip to a specific label within a CSS procedure. The $LABEL is used to define the object of a $GOTO.

Format:

$GOTO label [,REWIND]

$LABEL label

Parameters:

label	is from one to eight alphanumeric characters, the first of which must be alphabetic.
REWIND	specifies that the CSS file is to be reset to the beginning of the file. The search for a label starts at the beginning of the file. The REWIND option is used when it is known that a label precedes the current line. If REWIND is ommitted, the file is searched from the current position till the end of file and is then rewound and the search continued.

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.

Any commands following the $GOTO, with REWIND option or the same line, will be executed before the rewind takes place.

For example:

```
$GOTO TOP,REWIND;$WR Going to Top.
```

will print out the message before the rewind.

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
```

A $GOTO may refer to a label preceding the $GOTO. If the label is not found after searching the entire file, the following message is displayed by MTM:

```
      I/O ERR  TYPE=EOF
```

Example:

```
      $LABEL TOP
      $WR Infinite Loop (use $CLEAR to cancel)
      $GOTO TOP
```

This example illustrates the use of the REWIND option.

```
$LABEL  LABEL;*1st label
            .
            .
            .
$GOTO LABEL,R;*goto 1st label
            .
            .
            .
$GOTO LABEL;*goto 2nd label
            .
            .
            .
$LABEL LABEL;*2nd label
            .
            .
            .
$GOTO LABEL;*goto 1st label
            .
            .
            .
$EXIT
```

| $IFEXTENSION |

6.7.6 $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 Detail:

| $IFEX (with no fd) is always considered false.

6.7.7 $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.

6.8 $IF...CONDITIONAL Command

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 test conditions provided by the user instead. 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.

For decimal and hexadecimal mode, the comparison is performed by comparing the binary value of the numbers. The values can be enclosed in parentheses for compound expressions.

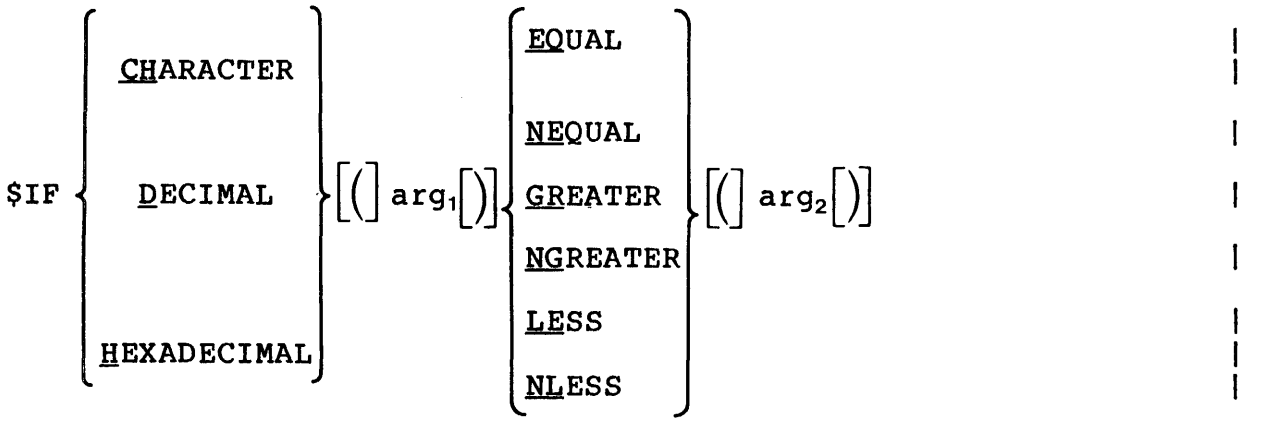
If, after comparing the arguments for each of the commands, the condition is determined to be true, subsequent commands are executed up to the corresponding $ELSE and $ENDC. If the condition is false, commands are skipped up to the corresponding $ELSE or $ENDC.

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.

The $IF...GREATER command is used to determine if the first argument is greater than the second argument. The $IF... NGREATER command is used to determine if the first argument is not greater than the second argument.

The $IF...LESS command is used to determine if the first argument is less than the second argument. The $IF...NLESS command is used to determine if the first argument if not less than the second argument.

Format:



Functional Details:

For the $IF...Conditional command, the arguments can be placed within parentheses. If the argument is within parentheses, spaces are allowed within the argument.

Example:

The following statement

```
$IF D (11 * 3) EQ (30 + 3)
```

is the same as

```
$IF D 11*3 EQ 30+3
```

The following are valid $IF...CONDITIONAL command statements:

```
$IF CHAR ABC EQ DEF; $WR EQUAL; $ENDC
$IF CHAR "D E F" EQ "DEF"; $WR EQUAL; $ENDC
$IF HEX A0 EQ (60+40); $WR EQUAL; $ENDC
```

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 multi-terminal monitor (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 system generation (sysgen) statement. Only one spooler can be active on the system at any given time. The OS/32 System Generation (Sysgen/32) 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. 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 file descriptor (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 with which the terminal user signs on.
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 with which the terminal user signs on.
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.

See 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:

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 can now 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 is completed, 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 by 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 to be printed or punched to one or more pseudo devices. As soon as the logical unit (lu) is closed, the OS/32 Spooler will automatically 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 (u-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.

Examples:

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 OS/32 Revision R06.2 or higher.

SPL/32 offers increased flexibility in creating and controlling the spooling environment of a system. Some of the features of SPL/32 include:

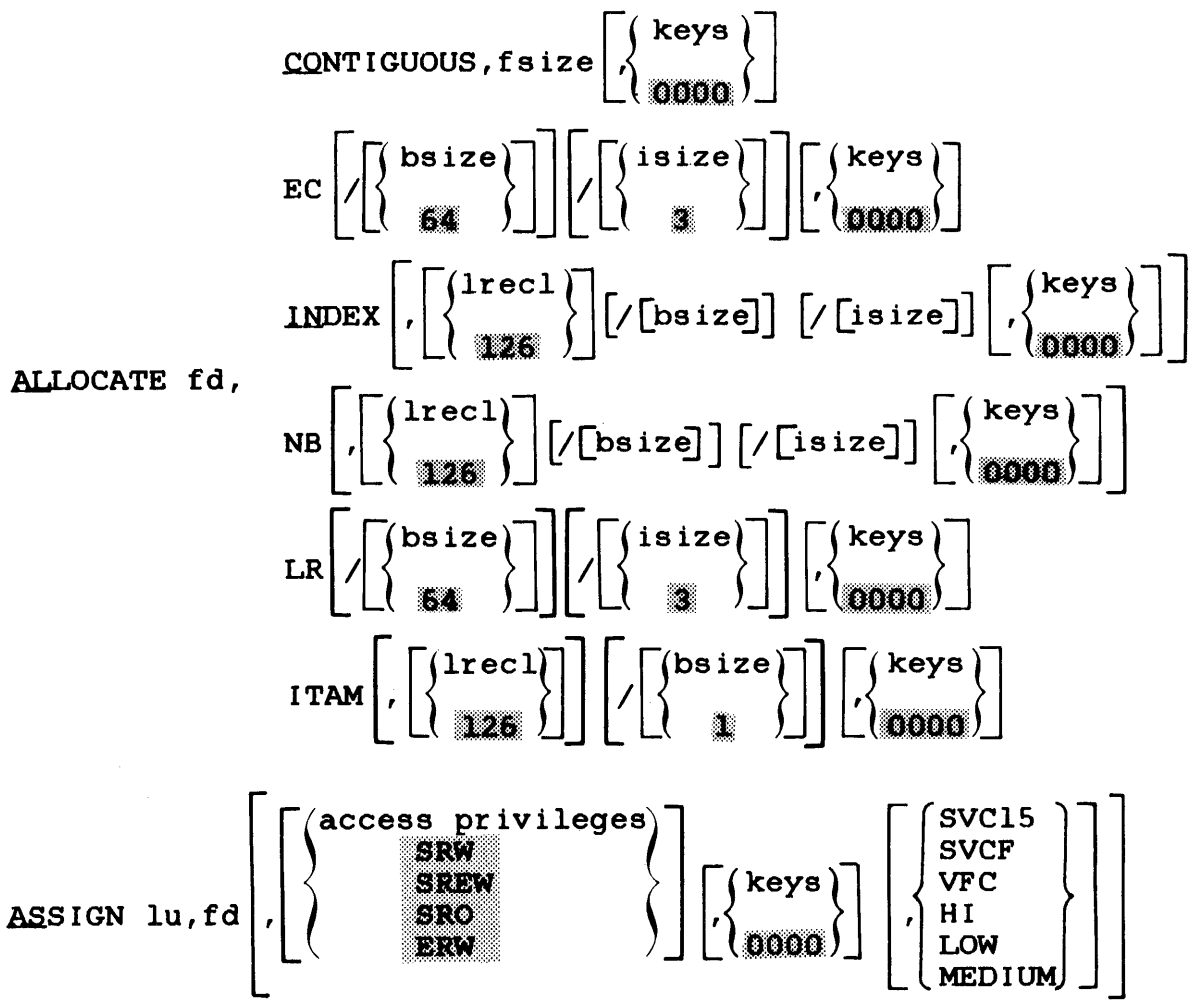
- The number of output devices is dependent only on the amount of available memory
- The capability of retaining a spooled output file after it is sent to a device
- The cability of holding spooled files from output processing
- The option to backspace, forward space or rewind a file that is currently 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 of altering the output requirements of a file waiting to be output
- The capability of altering the order in which files are output
- The capability of controlling 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 Multi-Terminal Monitor (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 see 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
MULTI-TERMINAL MONITOR (MTM) COMMAND SUMMARY



```
BUILD { fd [,APPEND] }
      { lu           }
   .
   .
   .
ENDB
```

```
CANCEL
```

```
CLOSE { lu1 [,lu2,...,lun] }
      {        ALL         }
```

```
CONTINUE [address]
```

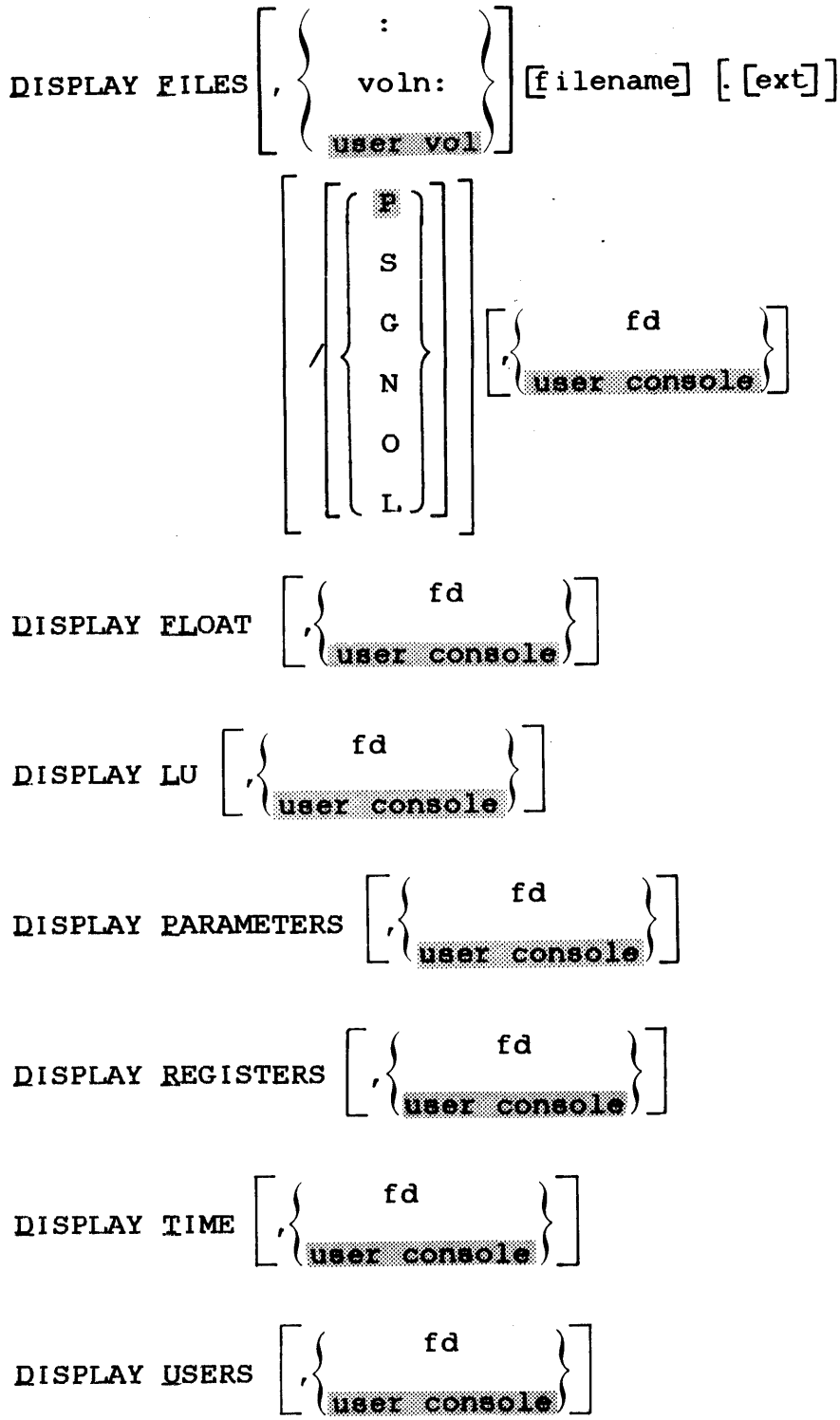
```
DELETE fd1 [,fd2,...,fdn]
```

```
DISPLAY { GVARIABLE } [ [ [ { n1/n2 } ] ] , [ { fd           } ] ]
        { IVARIABLE }       [ { n    } ]       [ { user console } ]
                            [ { ALL  } ]
```

```
DISPLAY ACCOUNTING [ , { fd           } ]
                       { user console }
```

```
DISPLAY DEVICES [ , { fd           } ]
                    { user console }
```

```
DISPLAY DFLOAT [ , { fd           } ]
                   { user console }
```


DISPLAY FILES
:
voln:
user vol
filename
ext
P
S
G
N
O
L
fd
user console
DISPLAY FLOAT
fd
user console
DISPLAY LU
fd
user console
DISPLAY PARAMETERS
fd
user console
DISPLAY REGISTERS
fd
user console
DISPLAY TIME
fd
user console
DISPLAY USERS
fd
user console

```
ENABLE { MESSAGE   }
       { PROMPT    }
       { ETM       }
       { TYPEAHEAD }
       { $VARIABLE }

EXAMINE address1 [ { ,n           } ] [ , { fd           } ]
                   { /address2    }       { user console }
                   { ,1           }

FFILE [fd,] lu

FRECORD [fd,] lu

HELP [ { mnemonic } ]
       { *        }

INIT fd [ , { segsize increment } ]
            { 1                 }

INQUIRE [ fd { ,fd1           } ]
             { user console   }

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

LOG [fd] [ , [ { NOCOPY } ] ] , [ { n  } ]
             { COPY   }           { 15 }

MESSAGE { userid    } message
        { .OPERATOR }
```

MODIFY address,[{data$_1$ / @}] [,data$_2$,...,data$_n$]

OPTIONS [{AFPAUSE / AFCONTINUE}][,{SVCPAUSE / SVCCONTINUE}] [,NONRESIDENT][[,LPU=[n]] [,NLPU]]

PASSWORD current password, new password

PAUSE

PREVENT{MESSAGE / PROMPT / ETM / TYPEAHEAD / $VARIABLE}

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

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

PURGE fd

$RELEASE {GVARIABLE / IVARIABLE}[,{n$_1$/n$_2$ / n$_1$ [,...,n$_n$] / ALL}]

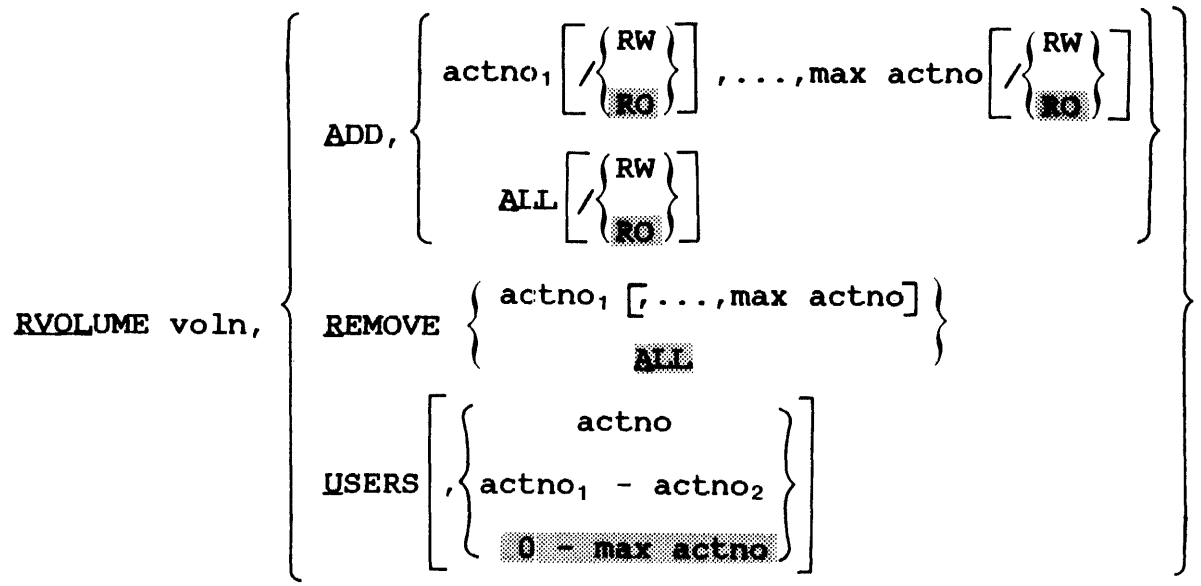
RENAME oldfd,newfd

REPROTECT fd,new keys

REWIND [fd,] lu

or

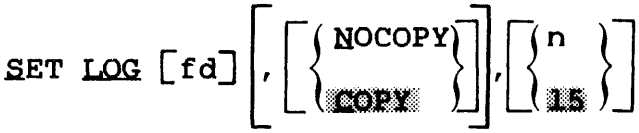
RW [fd,] lu



SEND message [;]

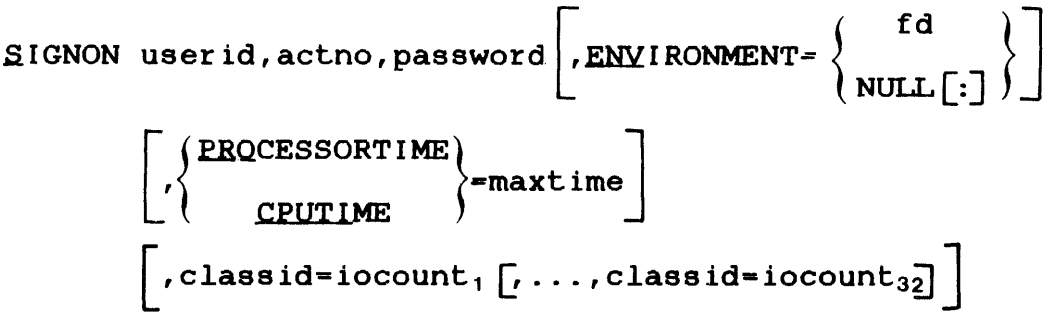
SET GROUP n

SET KEYOPERATOR [character]

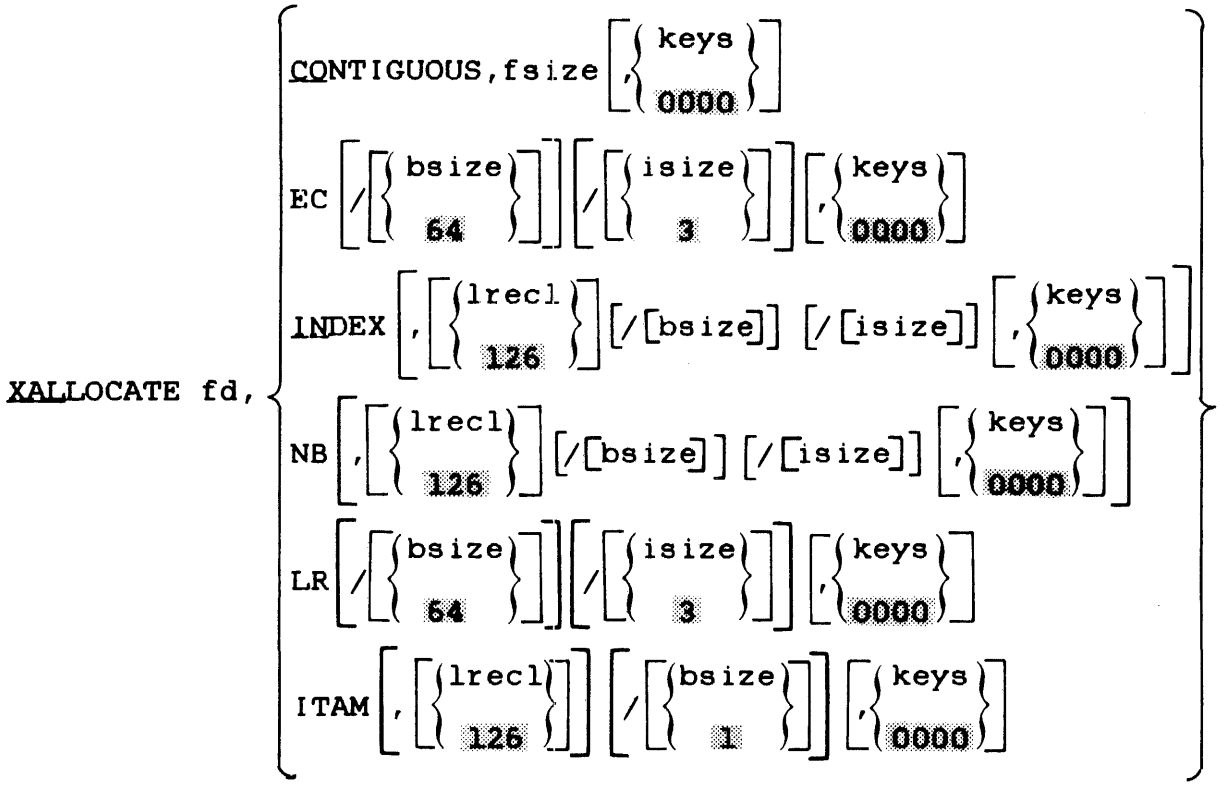


SET PRIVATE n

SIGNOFF

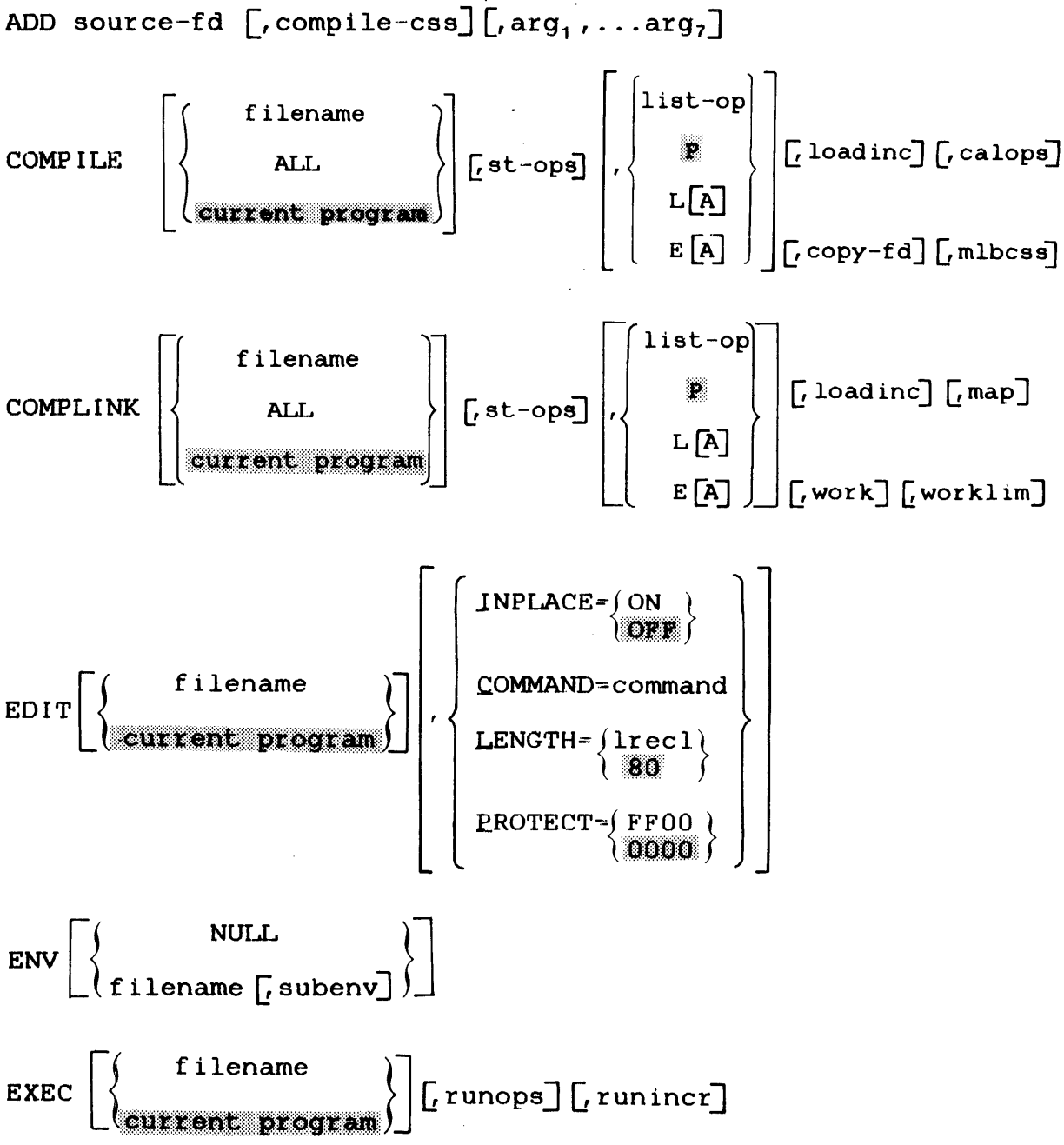


```
SPOOLFILE lu&lu1,pseudo-dev,FORM=formname [,{VFC   }]
                                           {IMAGE }

          [,{NOIMAGE}] [,{CHECKPOINT  }] [,COPIES=n]
            {NOVFC  }    {NOCHECKPOINT}

          [,{HOLD   }] [,{DELETE  }] [,PRIORITY=p]
            {RELEASE}    {NODELETE}

          [,BLOCK=bsize/isize]

START [{address         }] [,parameter1,...,parametern]
       {transfer address}

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

TASK [{taskid    }]
      {.BGROUND }

                  {CONTIGUOUS,fsize
                  {EC [/[{bsize}]] [/[{isize}]] [,{keys}]
                  {           {64}      {3}       {0000}
                  {INDEX [,[{lrecl}]] [/[bsize]] [/[isize]] [,{keys}]
                  {          {126}                            {0000}
TEMPFILE lu,      {NB [,[{lrecl}]] [/[bsize]] [/[isize]] ,{keys}
                  {          {126}                       {0000}
                  {LR [/[{bsize}]] [/[{isize}]] [,{keys}]
                  {           {64}      {3}       {0000}

VOLUME[{[voln] [/CSS]}]
       {  [*/CSS]    }

WFILE [fd,] lu
```

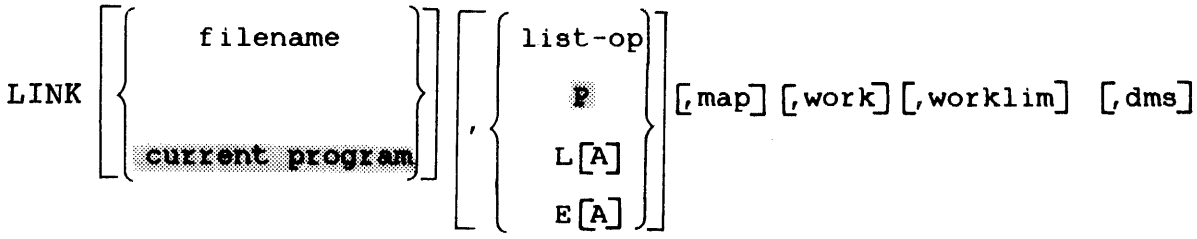


XDELETE fd_1 [,fd_2...,fd_n]

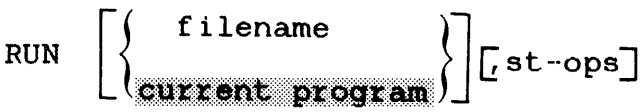
APPENDIX B
PROGRAM DEVELOPMENT COMMAND SUMMARY



LANGUAGE [language environment]



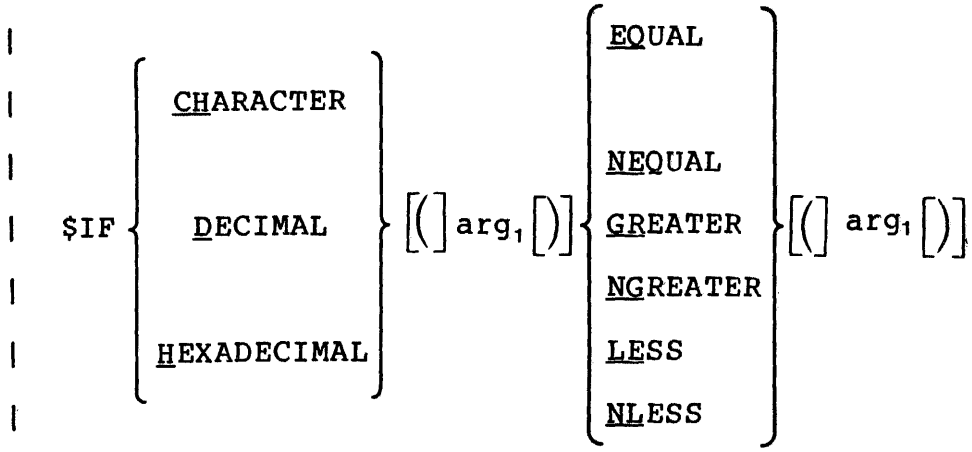
REMOVE fd



APPENDIX C
COMMAND SUBSTITUTION SYSTEM (CSS) COMMAND SUMMARY

```
%{char1char2₁ [char1char2₂ ...char1char2₄] %}
 {% new delimiter                          }

$BUILD {fd [,APPEND]}
       {lu          }
    .
    .
    .
$ENDB

$CLEAR

$CONTINUE

$COPY

$DEFINE {GVARIABLE } n}, [name] ,operator₁ [operator₂ ...operator_n]
        {LVARIABLE }

$ELSE

$ENDC

$EXIT

$FREE varname₁ [, ... ,varname_n]

$GLOBAL varname [({length})] [, ... ,varname [({length})]]
                 ({  B  })                    ({  B  })
```

$GOTO label [,REWIND]



$IFE n

$IFEXTENSION fd

$IFG n

$IFL n

$IFNE n

$IFNG n

$IFNL n

$IFNULL @n

$IFNNULL @n

$IFNX fd

$IFVOLUME fd

$IFX fd

$JOB [{PROCESSORTIME / CPUTIME}=maxtime]
[,classid=iocount$_1$] [,...,classid=iocount$_{32}$]
.
.
.
$TERMJOB

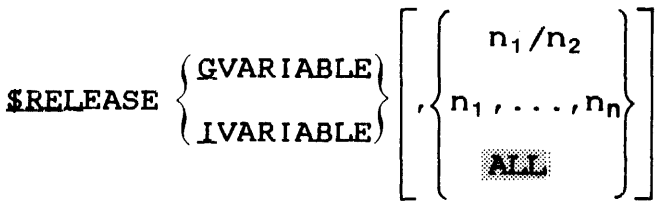
$LABEL label

$LOCAL varname [({length / 8})] [,...,varname [({length / 8})]]

$NOCOPY

$PAUSE

PRIOR n



$RELEASE { GVARIABLE / IVARIABLE } [, { n1/n2 / n1,...,nn / ALL }]

$SET varname=e

SET CODE n

$SKIP

$WAIT [{ n / 1 }]

$WRITE text [;]

APPENDIX D
MULTI-TERMINAL MONITOR (MTM) MESSAGE SUMMARY

ACCESS LEVEL ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Indicates that when memory access was attempted by the current program, the memory access level of the program status word (PSW) (bits 10-11) contained a lesser value than the access level field of the appropriate segment table entry (STE). The program address is XXXXXX. The memory fault address is given on Perkin-Elmer Series 3200 Processors.

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.

ADDRESS FAULT IN SVC AT XXXXXX

Indicates that the address of a supervisor call (SVC) parameter block or an address parameter in the parameter block points to a data structure that is outside the task taskid memory allocation or does not point to a data structure that is properly aligned.

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,

or the block size of an indexed file exceeds limit established at system generation (sysgen) time,

or 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 on-line.

ACCT-ERR

The account number specified is not valid.

APUO-ERR

OPTION NLPU is not valid for task linked with OPTION APUONLY.

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 Model 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 file control block (FCB).

ASGN-ERR TYPE=LU

An attempt was made to assign to a logical unit (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 file descriptor (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 exclusive write only (EWO) when only shared read 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 on-line.

BTCH-ERR

The batch capability was not started and is not available for a SUBMIT command.

BUFF-ERR

The expanded command substitution system (CSS) line overflowed CSS buffer size.

CLOS-ERR

Close failed for reason denoted by TYPE field.

| DECIMAL OVERFLOW ERROR AT XXXXXX
| NEXT INSTRUCTION AT XXXXXX

| Indicates that the result of load packed decimal string as a
| binary (LPB) instruction was too large to be stored as a
| binary number. The program address of the LPB instruction is
| XXXXXX. The instruction aborts and the next instruction is
| at XXXXXX. This fault occurs only on the Perkin-Elmer Series
| 3200 Processors.

DEF0-ERR

More than eight characters were specified for a keyword, a new variable name 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.

DEF5-ERR

Division by zero attempted.

DEF6-ERR

Arithmetic fault - result is greater than Y'0FFFFFFF'.

DEF7-ERR

A negative hexadecimal value was specified. Only positive values are allowed.

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 input/output (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 on-line.

DINC-ERR

The output from a DISPLAY, EXAMINE, or MODIFY command was terminated because the user task (u-task) was released or cancelled from the system console or by another task.

DUPLICATE USERNAME, FOREGROUND TASKNAME OR RESTRICTED USERID

Userid is already in use, restricted or matches the name of a foreground task.

END OF TASK n

Indicates that the task taskid has ended. The end of task code in decimal is n.

EXECUTE PRIVILEGE ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Indicates that an attempt was made by the current program to execute instructions for a segment that is execute-protected. The memory fault address is give on Perkin-Elmer Series 3200 Processors.

FD-ERR

The fd 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-FUNCTION RANGE ERROR AT XXXXXX

The floating-point instruction operand is not within the valid range for the function to be performed. The address of the faulting instruction is indicated by xxxxxx.

FLOATING POINT-UNDERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Results of floating point operation are too small to be represented; the instruction aborts. The next instruction is at XXXXXX. This error can only occur on the Perkin-Elmer Series 3200 processors.

FLOATING POINT-OVERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Floating point arithmetic procedure is too large to be represented; the instruction aborts. The next instruction is 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 u-task attempted to execute an illegal instruction at location XXXXXX.

ILLEGAL SVC-INSTRUCTION AT XXXXXX
SVC PARAMETER BLOCK AT XXXXXX

The u-task attempted to execute an illegal supervisor call (SVC) at location XXXXXX.

| INTERNAL REGISTER PARITY FAULT
| INSTRUCTION AT XXXXXX

| Indicates that a parity error machine malfunction is detected at location XXXXXX. This is an unrecoverable hardware-generated fault, which is possibly due to faulty external registers (REX). This fault occurs only on Model 3203 Processors.

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

Account number is invalid or unrecognized.

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 end of medium (EOM) or end of file (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

The lu is illegal or unassigned.

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 (LIB) is invalid. This error most frequently occurs when an attempt is made to load a task that was not built with Link.

LOAD-ERR TYPE=LOPT

Task options are incompatible with the system environment that attempted to load the task; i.e., attempt to load an executive task (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=PRES

A task has been loaded from the system console with the same name as a current USER ID.

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.

LPU#-ERR

The logical processing unit (LPU) number is out of range (allowed 0 to maximum set at sysgen).

LU-ERR

An lu specified in an assign statement is invalid.

LVL-ERR

The number of sysgen CSS levels was exceeded.

MAT PARITY FAULT
INSTRUCTION AT XXXXXX

Indicates that a memory address translator (MAT) parity error machine malfunction is detected at location XXXXXX. This is an unrecoverable hardware-generated fault, which is possibly due to faulty MAT circuitry or a bad chip. This fault occurs only on Model 3203 and 3205 Processors.

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 the following 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 the following message is displayed:

AFFECTED MEMORY PAGE MARKED OFF AT XXXXXX

MEMORY PARITY ERROR AT XXXXXX

An attempt was made to access nonexistent or bad memory on Model 7/32 and 8/32 machines.

MISSING PASSWORD

Password was omitted.

MISSING OPERAND

The entered command has a missing operand. Reenter the command specifying a value for n.

MNEM-ERR

The command mnemonic entered is unrecognizable or a nonprivileged user attempted to use a command that required privileged status.

NMPS-ERR

OPTION LPU and NLPU are only valid for the Model 3200MPS System.

NON EXISTENT SEGMENT ERROR (SST) AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Indicates that the current program has made a reference to a a nonexistant segment in the shared segment table (SST). The program address that caused the fault is XXXXXX. If the nonexistant segment is loaded, the instruction that caused the fault can be reexecuted. The memory fault address appears on the Perkin-Elmer Series 3200 Processors.

NO CMD-BUFF AVAILABLE

An attempt was made to send a message to a terminal for which all command buffers are occupied.

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.

NSPL-ERR

An attempt was made to use the SCTASK parameter of the LOAD command for a system using a spooler other than SPL/32.

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.

PAIR-ERR

The single quotation mark (') or double quotation mark (") symbols are not matched.

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.

PSM NOT SYSGENED

The entered command is rejected because the priority scheduling mechanism (PSM) was not system generated (sysgened).

READ PRIVILEGE ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Indicates that an attempt was made by the current program to read from a segment that is read-protected. The memory fault address is given for Perkin-Elmer Series 3200 Processors.

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.

REQS-ERR

The 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.

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 the user was not in the appropriate mode (e.g., CSS call in task-loaded mode).

SIGNON REQUIRED

An attempt was made to enter a command before signon,

or there was 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

The address of the SVC parameter block at XXXXXX was incorrect. 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.

TASK PAUSED

Indicates that the task taskid paused. The pause results from an SVC2 code 1 or the operator PAU SVC2 code 1 or the operator PAUSE command.

TIME-ERR

A task cannot be loaded because the user account central processing unit (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 on-line or the volume name is invalid.

WRITE PRIVILEGE ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Indicates that an attempt was made by the current program to write to a segment that is write-protected. The memory fault address is given for Perkin-Elmer Series 3200 Processors.

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
COMMAND SUBSTITUTION SYSTEM (CSS) MESSAGE SUMMARY

BUFF-ERR

indicates an expanded command line exceeds the CSS buffer. The task skips to $TERMJOB.

DBUF-ERR

indicates the operators of a $DEFINE command created a result that is greater than 110 characters or the command buffer size, whichever is smaller.

DEF0-ERR

indicates more than eight characters were specified for a keyword or a required name in the REQUIRED operator.

DEF1-ERR

indicates 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

indicates an empty additional keyword after a quote was used in a SEARCH operator specification.

DEF3-ERR

indicates the specified variable name is already in use.

DEF4-ERR

indicates the REQUIRED operator must be the last operator specified in a $DEFINE command.

DEF5-ERR

indicates a divide by zero was attempted.

DEF6-ERR

indicates an arithmetic fault; the result is greater than Y'0FFFFFFF'.

DEF7-ERR

indicates a negative hexadecimal value was specified. Only positive values are allowed.

FD-ERR

indicates an illegal or invalid file descriptor (fd), there is not enough space to build an fd, or required file support is not in the 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 end of file (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 the job is ended.

JOBS-ERR

indicates a second $JOB was found before a $TERMJOB was found.

KEYW-ERR

indicates 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 system generation (sysgen) time.

MNEM-ERR

indicates the command entered is not recognized. The task skips to $TERMJOB.

NOPR-ERR

indicates a required operand for a command was not specified.

PAIR-ERR

indicates the single quotation mark (') or double quotation mark (") 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

indicates 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 nonprivileged user.

TASK-ERR

indicates a task-related command was entered and there is no currently loaded task. The task skips to $TERMJOB.

%REP-ERR

indicates invalid replacement string definition or more than four replacement strings were 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 multi-terminal monitor (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

* ADD requires at least one argument

indicates a filename was not specified with the ADD command.

* CANNOT RUN -- filename.TSK DOES NOT EXIST

indicates a filename specified with the EXEC command does not exist in the environment.

* Compile css COMPext.CSS does not exist

indicates the filename entered with the ADD command contains a nonstandard extension.

* Current multimodule environment is edfname.EDF

indicates the ENV command was entered without a parameter, or the LIST command causes the contents of the current environment to be displayed.

* Deleting filename because it contains no records

indicates the EDIT command was entered, no text was allocated and the END command was entered. This file has been deleted.

* Enter name of file to be edited or * to just start EDITOR
* GIVE FILENAME=

indicates the EDIT command was entered with no filename. Entering a space bar and (CR) will enter the EDITOR with no file (in append mode).

* Environment edfname.EDF contains no files

indicates the LIST command was entered, but there are no filenames in the environment descriptor file (EDF).

* Environment name must have no extensions or .EDF

  indicates a filename was specified with the ENV command with an invalid extension.

* Execution of filename follows:

  indicates an image program was loaded and is executing.

* FILENAME already exists in environment edfname.EDF

  indicates an attempt was made to add an already existing filename to the EDF.

* FILENAME Compilation errors - listing on (device):

  indicates errors were encountered while compiling. These errors are listed on the specified device. FILENAME indicates the source file containing errors.

* FILENAME is not a valid file descriptor

  indicates an invalid filename form was specified in an ADD, COMPILE or REMOVE command.

* Filename must be specified or current program established
* before compilation

  indicates the COMPILE command was entered with no filename or current program.

* FILENAME not found in environment edfname.EDF

  indicates the filename was not found in the specified environment.

* Filename to be linked must be specified or there must be a
* current program for LINK to work on.

  indicates the LINK, EXEC or COMPLINK command was entered with no filename or current program.

* First argument filename is not permitted when in a multimodule
* environment. Environment name is always used.

  indicates a filename was specified where it was not required or allowed.

* Link errors -- listing on (device):

  indicates errors were encountered while linking. These errors are listed on the specified device.

* Must be in multimodule environment to use (command)

  indicates a development command such as COMPILE, COMPLINK, EXEC or LIST was entered without first setting the environment to a multimodule environment.

* Must have current program or specify file in order to run

  indicates the RUN command was entered when no current environment is set.

* Must specify extension or compile css name

  indicates the ADD command was entered along with a filename and no extension.

* New multimodule environment is edfname.EDF

  indicates a new multimodule environment has been allocated.

* No current environment

  indicates a filename was not specified with the ENV command.

* REMOVE requires one argument

  indicates a filename was not specified with the REMOVE command.

APPENDIX G
MULTI-TERMINAL MONITOR (MTM)/NON-MTM MONITOR TASK INTERFACE MESSAGES

G.1 $FOREGROUND TASK INTERFACE MESSAGES

The following messages are output to the system console:

xxxx-ERR SNDTID = sender task-id MSGE: received message

Where:

xxxx can be any of the following error statuses:

- PARM indicates bad syntax in terminal-dn.
- TNEX indicates the specified terminal-dn is not known by MTM.
- TNCM indicates the terminal is not in correct mode.
- TASE indicates a terminal assign error on an $END message (still assigned to FOREGROUND task?).
- DSTA indicates a duplicate $STA message for the same terminal-dn was received.
- DEND indicates a duplicate $END message for the same terminal-dn was received.
- MSTA indicates a missing $STA message.
- MEND indicates a missing $END message.

The following messages are output to the user console:

MNEM-ERR

interface is not available for normal MTM users.

MOSQ-ERR

mode sequence error occurred; the terminal is not in normal MTM mode.

NTSK-ERR

selected task is not in foreground or restricted task name.

SEQ-ERR

indicates task-loaded, task-executing, command substitution system (CSS) or batch mode.

SMGS-ERR

sends message error.

#MST-ERR

$STA message from FOREGROUND task is missing; the terminal was reassigned.

#MEN-ERR

$END message from FOREGROUND task is missing; the terminal was reassigned.

TASE-ERR

indicates a FOREGROUND task assign error.

TSPC-ERR

indicates a FOREGROUND task has no more space to add to the user's terminal; try again later.

G.2 HASP INTERFACE MESSAGES

MNEM-ERR

a nonprivileged user entered the $HASPxx command or no HASP-TUB was generated at MTM system generation (sysgen) time.

SEQ-ERR

terminal is in CSS, batch, task-loaded or task-executing mode.

NTSK-ERR

no such HASPxx taskid was found in the foreground.

USED-ERR

the selected HASPxx is currently being used by another MTM user.

TSPC-ERR

no HASP-TUB is available (there are more HASP tasks than were specified by SGN.$HSP at MTM sysgen time).

SMGS-ERR

indicates an error on the sending message to HASPxx.

APPENDIX H
CONTROL SUMMARY FOR BIDIRECTIONAL INPUT/OUTPUT CONTROL (BIOC) CRT DRIVER

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, 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

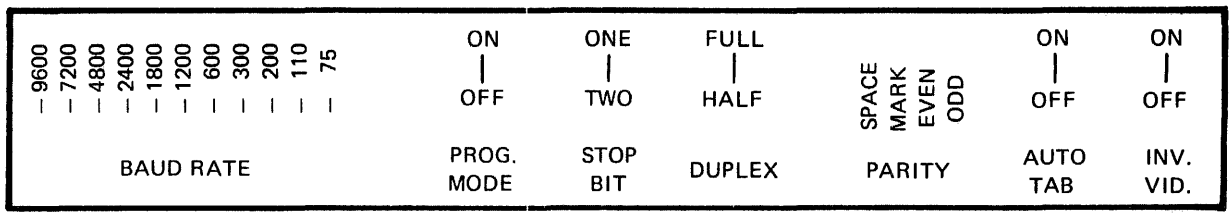


Figure H-1 Perkin-Elmer Model 1200 Mode Selectors

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 can 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 moves 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 (CR) 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, 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 CR. This will cause the same number of lines to appear; each CR will 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 CR. 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 move to the end of the line before storing the CR 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 carriage return/line feed (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-R 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 cannot 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) Move 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 move 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 can 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

Concurrent Computer Corporation

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, *etc.*

1. Publication number _____

2. Title of publication _____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

   _____

   _____

   _____

4. Was the publication easy to understand? If no, why not?

   _____

5. Were illustrations adequate? _____

   _____

   _____

6. What additions or deletions would you suggest? _____

   _____

   _____

7. Other comments: _____

   _____

   _____

From _____ Date _____

Position/Title _____

Company _____

Address _____

_____

_____

STAPLE 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

Concurrent
Computer Corporation

2 Crescent Place
Oceanport, NJ 07757

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT., HANCE AVE.

FOLD FOLD

STAPLE STAPLE