PERKIN-ELMER

OS/32
MULTI-TERMINAL MONITOR (MTM)

Reference Manual

48-043 F00 R00

The Perkin-Elmer Corporation, Computer Systems Division 2 Crescent Place, Oceanport, New Jersey 07757

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES

FIGURES

FIGURES (Continued)

TABLES

PREFACE

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

Chapter 1, which is reorganized, 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, and Chapter 3 explains the program development commands. Chapter 4 describes batch processing under MTM. Chapter 5 describes the command substitution system (CSS) and includes CSS commands. Chapter 6 describes spooling.

Appendix A summarizes the MTM user commands. Appendix B is a summary of the program development commands. Appendix C summarizes the CSS commands, and Appendix D is an MTM command message summary. Appendix E is a summary of CSS messages, and Appendix F is a summary of program development command messages.

This manual replaces S29-591. Revision R00 adds Chapter 3, describing the new program development commands. The signon CSS, USERINIT.CSS, is made more flexible. Vertical forms control (VFC) is added, and changes are made to several MTM user commands. A Help facility enables a user to access information about MTM and program development commands. For batch processing, the SUBMIT command is upgraded, and the batch signon requirements are simplified. Global and local variables are added to CSS, requiring four new commands: $FREE, $GLOBAL, $LOCAL, and $SET. Also, there is a reserved global variable for end of task codes, and there are reserved variables for assigning logical units in a program development environment. The $WAIT command is also added to CSS. Logical units can now be automatically assigned.

This revision applies to the OS/32 R06.1 software release and higher.

The following publications can be used in conjunction with this manual:

MANUAL TITLE	PUBLICATION NUMBER
OS/32 AIDS User's Guide	S29-374
OS/32 COPY User Guide	S29-676
OS/32 LINK Reference Manual	48-005
OS/32 EDIT User Guide	48-008
OS/32 Multi-Terminal Monitor (MTM) System Planning and Operator Reference Manual	48-023
OS/32 Operator Reference Manual	48-030
OS/32 System Support Utilities Reference Manual	48-031
OS/32 Supervisor Call (SVC) Reference Manual	48-038
OS/32 Application Level Programmer Reference Manual	48-039
32-Bit Systems User Documentation Summary	50-003

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

CHAPTER 1
GENERAL DESCRIPTION

1.1 INTRODUCTION

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

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

1.2 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:

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

Examples of dynamic functions are:

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

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

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

1.3 USER INFORMATION

Under MTM control, a terminal user can:

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

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

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

1.3.1 MTM Devices

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

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

1.3.2 Authorization

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

1.3.3 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.

1.3.4 Number of Terminal Users

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

1.4 MTM ENVIRONMENTS

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

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

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

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

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

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

- awaiting execution, or
- executing.

If a job already has completed execution, the returned status will be: no jobs found.

1.4.1 MTM Terminal Modes

An active terminal is defined to be in one of four 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 four modes:

- Command mode: No task is loaded, CSS procedure is not executing and BUILD is not in effect. All nontask-related commands are accepted. An "*" prompt is displayed in this mode.

- Task loaded mode: The task was loaded but was not started, or is paused. An "*" prompt is displayed in this mode.

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

- CSS mode: A CSS procedure is being built or executed. A "-" prompt is displayed in this mode. When CSS terminates, the terminal returns to command mode and a "*" prompt is output. When a CSS procedure is being built, a "B>" prompt is displayed.

1.4.1.1 Interactive Task to Terminal Mode

When a task issues an SVC1 I/O operation to an active terminal that is in task executing mode, MTM treats the I/O as a wait operation. This is of no concern for tasks that do SVC1 wait I/O. However, 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. Then MTM 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 TSW.

1.5 LOADING A TASK

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

1.6 MTM SPECIAL FEATURES

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

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

1.6.1 Command Substitution System (CSS)

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

1.6.2 The Help Facility

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

1.6.3 Program Development Commands

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

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

1.6.4 Spooling

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

1.6.5 Security and Access Protection of Disks

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

1.6.6 Signon CSS

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

1.7 CONVENTIONS

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

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

1.7.1 Prompt Conventions

A prompt is output to a terminal device to indicate that the MTM system is ready to accept input from the user. The 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 that input data be copied to a BUILD file.
-	Indicates that the system is ready to accept a command while an interactive task is active. 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.

1.7.2 Terminal Conventions

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

TABLE 1-2 TERMINAL CONVENTIONS

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

1.7.2.1 Using the Break Key

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

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

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

1.7.3 Command Conventions

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

1.7.4 File Conventions

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

1.7.4.1 Private Account Numbers

During the signon procedure a terminal user must supply the 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 private files has unrestricted access to those files and can update, execute, access, or delete as required. Furthermore, no other terminal user can gain access to another user's private files. However, to supply greater flexibility for file sharing, MTM supports the concept of group files.

1.7.4.2 Group Account Numbers

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

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

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

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

1.7.4.3 System Account Numbers

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

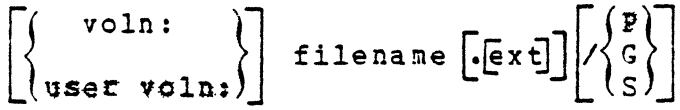
Within an MTM environment, the system operator is viewed as more privileged than terminal users with respect to file ownership. The system operator can allocate system files and can also designate an existing file to be made private, group, or system. Similar to a terminal user, the system operator uses the RENAME command to change file ownership.

1.7.4.4 File Descriptors

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

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

The format of the file descriptor is:



Parameters:

voln: is the name of the disk volume on which the file resides, or the name of a device. Voln can be from one to four characters. The first character must be alphabetic and the remaining, alphanumeric. This parameter need not be specified. If this parameter is not specified, the default volume (set with the VOLUME command) is used. When voln is not specified, the colon separating voln and filename must not be entered. Where voln refers to a device name, a colon must follow the device name, and neither the filename nor the extension is entered.

filename is the name of a file. A filename consists of from one to eight alphanumeric characters, the first of which must be alphabetic.

.ext is a 1- to 3-character alphanumeric string preceded by a period specifying the extension to a filename. If the period (.) and extension are omitted, a default extension appropriate to the particular command in which the fd appears is appended to the filename. 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 who created the file. All of the facilities for file manipulation are available to the owner of this file. No other user has access to this file unless it is also a group file. That is, the account number of the user who created the file 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 is a user's private file, is accessible to other terminal users for read only. 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.

The following are valid examples of file descriptors:

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

CHAPTER 2
MULTI-TERMINAL MONITOR (MTM) USER COMMANDS

2.1 INTRODUCTION

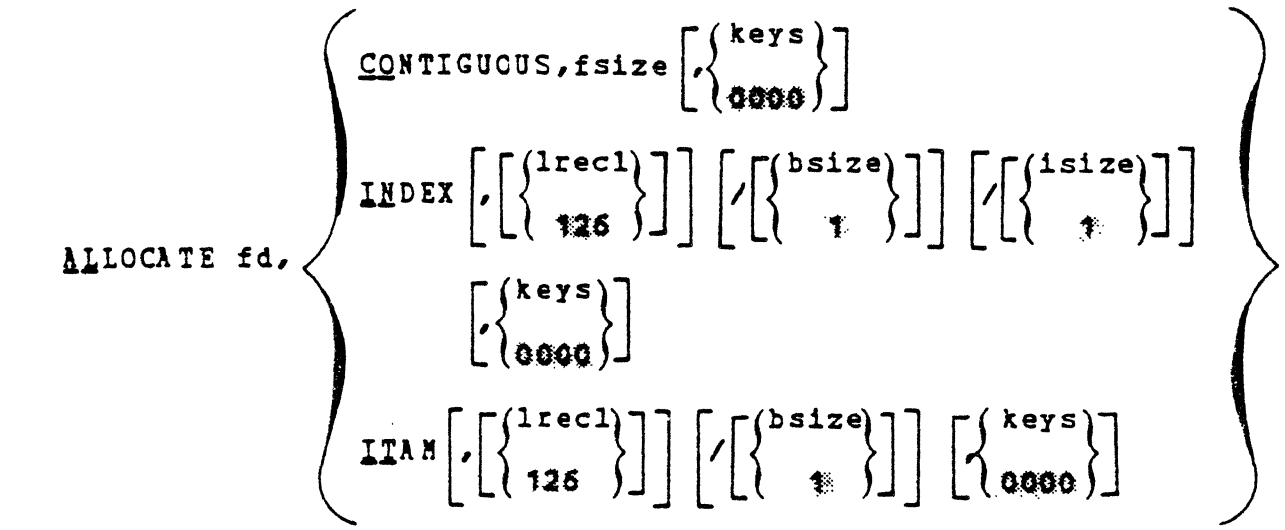
The following steps comprise a basic MTM terminal session:

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

2.2 ALLOCATE COMMAND

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

Format:



Parameters:

fd	is the file descriptor of the device or file to be allocated.
CONTIGUOUS	specifies that the file type to be allocated is contiguous.
fsize	is a decimal number indicating file size which is required for contiguous files. It specifies the total allocation size in 256-byte sectors. This size may be any value up to the number of contiguous free sectors existing on the specified volume at the time the command is entered.
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 ITAM device. It cannot exceed 65,535 bytes. Its default is 126 bytes. It may optionally be

followed by a slash (/) which delimits lrecl from bsize.

bsize — is a decimal number specifying the number of 256-byte sectors contained in a physical block to be used for buffering. This parameter cannot exceed the maximum block size established at sysgen time. If bsize is omitted, the default value is one sector. When the file type is ITAM, bsize is the buffer size in bytes.

isize — is a decimal number specifying the indexed block size. If isize is omitted, the default value is one sector. Like bsize, isize cannot exceed the maximum block size established at sysgen time.

ITAM — specifies that the device to be allocated is a communications device.

keys — specifies the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword, the left byte of which signifies the write key and the right byte, the read key. If this parameter is omitted, both keys default to 0.

Functional Details:

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

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

Examples:

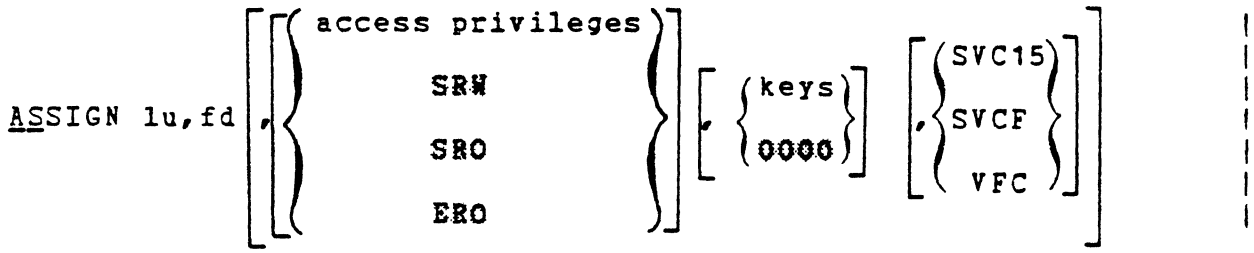
AL JANE.TSK,CO,64	Allocates, on the default user volume, a contiguous file named JANE.TSK whose total length is 64 sectors (16kb) with protection keys of 0.

AL M300:AJM.BLK,IN,132/4	Allocates, on volume M300, an indexed file named AJM.BLK with logical record length of 132 bytes, data block size of four sectors, and default isize of one sector. The protection keys default to 0. When this file is assigned, the system must have 2.25kb of available system space for buffers.
AL THISFILE,IN,256/4/2	Allocates, on the default user volume, an indexed file named THISFILE (blank extension) with a logical record length of 256 bytes, a data block size of four sectors, an index block size of two sectors, and protection keys of 0.
AL VOL1:AJM.OBJ,IN,126	Allocates, on volume VOL1, an indexed file named AJM.OBJ whose logical record length is 126 bytes. The buffer size and indexed block size default to one sector and the protection keys default to 0.
AL VO1:AJM.OBJ,IN,126//3	Allocates, on volume VO1, an indexed file named AJM.OBJ with logical record length of 126 bytes. The data block size defaults to one sector, the index block size is three sectors, and the protection keys default to 0.

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 number to which a device or file is to be assigned.
fd	is the file descriptor of the device or file to be assigned.
access privileges	are the desired access privileges. The default access privileges are: - SRW for contiguous and indexed files - SRO for private files within the group or system account - ERO for devices
keys	signifies the read/write protection keys of the file or device to be assigned.
SVC15 SVCF	signifies that the specified device is to be assigned for SVC 15 access. SVCF is the hexadecimal equivalent of SVC15 and can also be specified. This option pertains to communications devices only. If SVC 15 access is specified, vertical forms control cannot be specified.

| VFC | specifies the use of vertical forms control for the assigned lu. If this parameter is specified, SVC 15 access cannot be specified. If this parameter is omitted, there is no vertical forms control for the device assigned to the specified lu.

| | Full VFC support is in effect only if output is to a line printer.

NOTE

If the access privileges and keys parameters are omitted and VFC is specified, the positional commas belonging to the omitted parameters can be omitted.

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

Functional Details:

Access privileges can be one of the following:

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

| When the SVC 15 option is specified, only SRW, SREW, ERSW, and | ERW access privileges are accepted.

The DISPLAY LU command is used to determine the current access privileges of all assigned units. The command is rejected if the requested access privilege cannot be granted.

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

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

TABLE 2-1 ACCESS PRIVILEGE COMPATIBILITY

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

LEGEND

* compatible
\- incompatible

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

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

Some examples of protection using keys are:

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

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

The ASSIGN command can be entered in task loaded mode.

Examples:

AS 2,FILE.DAT,EWO,99AA	Assigns a disk file to lu 2. The EWO access privilege causes the file to be positioned at the end. It is conditionally protected with write and read keys of 99AA. New records are appended.

AS 2,TEST.JOB,VFC	Assigns a disk file to lu 2. Vertical forms control is in use. Access privileges and keys parameters are omitted along with their respective commas.
AS 2,TEST.JOB,,,VFC	Assigns a disk file to lu 2. Vertical forms control is in use. Access privileges and keys parameters are omitted but positional commas are specified.
AS 2,TEST.JOB,,VFC	Assigns a disk file to lu 2. Vertical forms control is in use. The positional comma belonging to the omitted access privileges parameter must be specified.
AS 2,TEST.JOB,SRO,VFC	Assigns a disk file to lu 2. Vertical forms control is in effect. The keys parameter, along with the positional comma, is omitted. The privilege is shared read only.

Invalid Examples:

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

2.4 BFILE COMMAND

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

Format:

BFILE fd, lu

Parameters:

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

Functional Details:

The BFILE command can be entered in task loaded mode.

Examples:

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

2.5 BIAS COMMAND

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

Format:

BIAS $\left\{\begin{matrix}\text{address} \\ * \end{matrix}\right\}$

Parameters:

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

Functional Details:

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

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

Example:

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

2.6 BREAK COMMAND

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

Format:

BREAK

Functional Details:

The BREAK command can be entered in task executing mode.

2.7 BRECORD COMMAND

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

Format:

BRECORD [fd,] lu

Parameters:

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

Functional Details:

The BRECORD command can be entered in task loaded mode.

Examples:

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

2.8 BUILD AND ENDB COMMANDS

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

Format:

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

Functional Details:

Lines entered from the terminal after the BUILD command are treated as data, and are copied to the specified device or file until an ENDB command is encountered. ENDB may be followed by other commands in the command line. Data following the ENDB command is treated as a command. If any data follows the BUILD command on the same line, it is treated as a comment and no action is taken. The BUILD command can be entered from the terminal only if 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 CPUTIME=utime/ostime

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

2.10 CLOSE COMMAND

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

Format:

$$\underline{CL}OSE \left\{ \begin{matrix} lu_1 \left[,lu_2,...lu_n\right] \\ \underline{A}LL \end{matrix} \right\}$$

Parameters:

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

Functional Details:

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

The CLOSE command can be entered in task loaded mode.

Examples:

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

```
------------
| 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 in task loaded mode. Executing this command causes the terminal mode to be switched from task loaded mode to task executing mode.

2.12 DELETE COMMAND

The DELETE command deletes a direct access file.

Format:

DELETE 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 mode, task loaded mode, and task executing mode.

2.13 DISPLAY ACCOUNTING COMMAND

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

Format:

DISPLAY ACCOUNTING [,{fd / user console}]

Parameter:

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

Functional Details:

The DISPLAY ACCOUNTING command displays this information:

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

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

2.14 DISPLAY DEVICES COMMAND

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

Format:

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 Details:

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

Example:

```
 D D
NAME  DN KEYS
NULL   0 0000
CON    2 0000
CR     4 0000
PRT   62 0000
PTRP  13 0000
PR     0 0000    SPOL
SPL    0 0000    SPOL
CRT1  30 0000
CRT2  14 0000
MAG1  85 0000
DSC1  C6 0000    MUD1     PROT     CDIR
DSC2  C7 0000    FIXD     RES      CDIR
DSC3  D6 0000    MTM      SYS
DSC4  D7 0000    FIX4
DSC5  E6 0000    OFF
D67A  FC 0000    V67A              CDIR
*
```

In the DISPLAY DEVICES output, columns 1, 2, and 3 contain the device name, device number (address), and keys, respectively. Column 4 is only defined for pseudo-print (spool), ITAM (communications), and direct access devices. The characters SPOL specify that the devices are pseudo-print devices used in spooling.

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

2.15 DISPLAY DFLOAT COMMAND

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

Format:

DISPLAY DFLOAT [,{fd / user console}]

Parameter:

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

Functional Details:

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

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

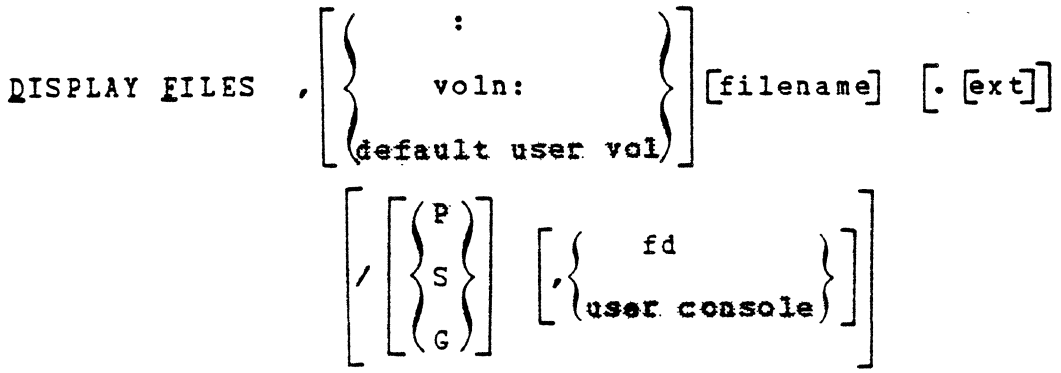
Example:

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

2.16 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

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

Parameters:

:	specifies that all files with the user account number be displayed regardless of what volume they reside on. Entering the colon with part of a filename limits the file search to filenames with the specified characters.
voln:	is a 1- to 4-character name of a disk volume. The first character must be alphabetic, the remaining alphanumeric. If voln is omitted, the default is the user volume.
filename	is a 1- to 8-character name of a file. The first character must be alphabetic, the remaining, alphanumeric.
ext	is a 1- to 3-character extension to the filename.

P	indicates that information is requested for a private file.
S	indicates that information is requested on a system file; default is private files only.
G	indicates that information is requested for a group file; default is private files only.
fd	is the file descriptor specifying the file or the device to which the display is output. If fd is omitted, the default is the user console.

Functional Details:

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

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

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

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

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.

****32.0- displays all files, eight characters long, whose last two characters are 32 and whose extension begins with an 0.

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

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

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

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

An example of the display produced by the DISPLAY FILES command is:

```
D F,FIXD:-.TSK/S

VOLUME= FIXD
FILENAME  EXT TYPE LENGTH KEYS START/NLR CREATED WRITTEN ACT
DSKSPACE  TSK   CO      9 **00        E* 6/09/79 6/09/79   0
OSCOPY    TSK   IN    256 **00        21 6/09/79 6/09/79   0
JFC       TSK   CO     81 **00       E3* 4/07/79 4/07/79   0
COBOL     TSK   CO     26 **00      134* 7/27/78 0/00/00   0


D F,TAS-/-


VOLUME= V67B
FILENAME  EXT TYPE LENGTH KEYS START/NLR CREATED WRITTEN ACT
TAS       CSS   IN     80 **00         1 6/09/79 6/09/79  13
TASKRT    TSK   CO    105 0000    218A* 4/07/79 4/07/79  22


D F,-.-/S


VOLUME= V67B
FILENAME  EXT TYPE LENGTH KEYS START/NLR CREATED WRITTEN ACT
INA1          IN    126 **00         0 6/09/79 6/09/79   0
INA2      CSS IN    100 **00         1 6/09/79 6/09/79   0
```

For contiguous files, TYPE is CO, LENGTH is the number of sectors allocated to the file in decimal, and START/NLR is the starting sector number in hexadecimal, followed by *.

For indexed files, TYPE is IN, length is the logical record length in decimal, and the START/NLR is the number of logical records in decimal.

ACT is the associated user's account number. It is the user's account number for private files, the group account number for group files and 0 for system files.

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

Examples:

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

D F,M-:TASK.5*	displays to the user's terminal the files named TASK that have one or two character extensions starting with the character 5. A separate display of these files is done for each online disk volume whose name starts with the letter M.
D F,-:TASK.-	displays to the user's terminal the files named TASK, with any extension. A separate display of these files is done for each online disk volume in the system.

2.17 DISPLAY FLOAT COMMAND

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

Format:

DISPLAY FLOAT [,{fd / user console}]

Parameter:

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

Functional Details:

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

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

Example:

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

2.18 DISPLAY LU COMMAND

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

Format:

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

Parameter:

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

Functional Details:

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

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


 1   M67A:RADPROC.CSS/000,SRO               200  100.0%
 3   CON:,SRW
 4   M67A:E2614586.001/000,SREW               1  100.0%
 5   CON:,SRW
 6   CON:,SRW
```

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

Example:

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

2.19 DISPLAY PARAMETERS COMMAND

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

Format:

DISPLAY PARAMETERS [,{fd / user console}]

Parameter:

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

Functional Details:

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

TABLE 2-2 DISPLAY PARAMETERS COMMAND FIELDS

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

TABLE 2-2 DISPLAY PARAMETERS COMMAND FIELDS (Continued)

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

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

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

TABLE 2-3 TASK OPTION BIT DEFINITIONS

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

TABLE 2-3 TASK OPTION BIT DEFINITIONS (Continued)

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

TABLE 2-3 TASK OPTION BIT DEFINITIONS (Continued)

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

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

TABLE 2-4 WAIT STATUS BIT DEFINITIONS

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

TABLE 2-4 WAIT STATUS BIT DEFINITIONS (Continued)

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

NOTE

Zero status indicates an active task.

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

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

Example:

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

```
*DISPLAY PARAMETERS

TASK      MTMUSER
CTSW      00001000
PSW          477F0
CLOC         F2B7C
STAT          2000
TOPT         10021
USSP          14F8
MUSP          2208
MXSP          3000
CTOP          24FE
UTOP          237C
UBOT             0
SLCC         F0000
NLU             15
MPRI           128
SVOL          M67A
```

2.20 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:

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

NOTE

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

Example:

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

2.21 DISPLAY TIME COMMAND

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

Format:

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

Parameter:

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

Functional Details:

The display has the following format:

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

or alternatively (by sysgen option):

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

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

2.22 DISPLAY USERS COMMAND

The DISPLAY USERS command displays the userid and terminal device names of all users currently signed on.

Format:

DISPLAY USERS [,{fd / user console}]

Parameter:

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

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

Example:

```
D U
ME-CT01:   STARTERI-CT02:   AVE-CT03:   JAW-CT04:
```

2.23 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.
ETM	displays the end of task message.
SVARIABLE	enables variable processing on a per user basis.

Functional Details:

The FNABLE command does not affect operator messages.

Variable support is included in the target system via the sysgen option SGN.VAR.

2.24 EXAMINE COMMAND

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

Format:

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

Parameters:

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

Functional Details:

Specifying only address1 causes the contents of memory at that location (as modified by any previous BIAS command) to be displayed. Specifying address1 and address2 causes all data from the first to the second address to be displayed.

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

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

Example:

```
BI  B100
EXA 100,10
```

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

2.25 FFILE COMMAND

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

Format:

FFILE [fd,] lu

Parameters:

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

Functional Details:

The FFILE command can be entered in task loaded mode.

Examples:

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

2.26 FRECORD COMMAND

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

Format:

FRECORD [fd,] lu

Parameters:

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

Functional Details:

The FRECORD command can be entered in task loaded mode.

Examples:

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

2.27 HELP Command (MUST BE IN EDIT FOR HELP * or HELP MNEMONIC)

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

Format:

HELP [{mnemonic / *}] → MUST BE IN EDIT

Parameters:

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

Functional Details:

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

Examples:

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

Example:

```
HELP*
ADD               AL(LOCATE)        AS(SIGN)          BF(ILE)
BI(AS)            BR(ECORD)         BU(ILD)           CAL
CA(NCEL)          CL(OSE)           COBOL             COMPILE
COMPLINK          CO(NTINUE)        DE(LETE)          D(ISPLAY)
EDIT              ENA(BLE)          ENDB              ENV
EXA(MINE)         EXEC              FF(ILE)           FORT
FORTO             FR(ECORD)         HELP              LINK
LIST              L(OAD)            LOG               MACRO
ME(SSAGE)         MOD(IFY)          P(AUSE)           PRE(VENT)
PRI(NT)           PUN(CH)           REMOVE            REN(AME)
REP(ROTECT)       REW(IND)          RPG               RUN
RW                RVOL(UME)         SEN(D)            SIGNOF(F)
S(IGNON)          ST(ART)           T(ASK)            TE(MPFILE)
V(OLUME)          WF(ILE)           XAL(LOCATE)       XDE(LETE)
SUP(MIT)          INQ(UIRE)         PUR(GE)
FOR HELP ON  ANY  OF  THE  ABOVE  COMMAND  MNEMONICS,  TYPE  HELP
MNEMONIC

* HELP AS

ASSIGN:  THE  ASSIGN  COMMAND  ASSIGNS  A  DEVICE,  FILE  OR
COMMUNICATIONS DEVICE TO ONE OF A TASK'S LOGICAL UNITS.


FORMAT:


   (AS) SIGN LU, FD, , ACCESS PRIVILEGES, KEYS, SVC15
```

2.28 INIT COMMAND

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

Format:

INIT fd [, {segsize increment}]

Parameters:

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

Functional Details:

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

The INIT command can be entered in command mode.

Examples:

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

2.29 LOAD COMMAND

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

Format:

LOAD [taskid,] fd [,segsize increment]

Parameters:

taskid	specifies the name of the task to be loaded.
fd	specifies the file or device the task is being loaded from.
segsize increment	specifies amount of memory in kb (above the memory size) that the task needs for processing. When a task is built (via Link), the OPTION WORK=n command adds additional memory to a task. The size field in the LOAD command overrides the amount of memory specified by Link. The size is accepted in .25kb increments.

Functional Details:

In order to maintain CSS compatibility, a background (.BG) task also can be loaded into memory. Any valid taskid can be entered but will be ignored.

If a task is loaded from a direct access device, the system first searches the user volume or the specified volume under the user's account. If the file is not found in the search, the system automatically looks for the file on the system volume in the system account. If only the fd is 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.

NOTE

The LOAD command loads the task file <fd> into the terminal user's segment. The TASK and the OPTION commands are ignored if the task is currently loaded.

Examples:

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

2.30 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 through 65,535 specifying the number of lines after which the user log file is to be checkpointed. If this parameter is omitted, the default is 15 lines. If n is specified as 0, no checkpointing will occur.

Functional Details:

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

Checkpointing is only meaningful for indexed files on disk. The LOG command can be entered in command mode, task loaded mode, and task executing mode.

Example:

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

2.31 MESSAGE COMMAND

The MESSAGE command sends a message to a specified user.

Format:

TO MTM TERMINAL

MESSAGE { userid / .OPERATOR } message

TO CONSOLE

Parameters:

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

Functional Details:

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

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

Example:

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

```
ME AVE HELLO MTM USER
```

The format of the message received is:

```
TK-HELLO MTM USER
```

2.32 MODIFY COMMAND

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

Format:

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

Parameters:

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

Functional Details:

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

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

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

Examples:

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

2.33 OPTIONS COMMAND

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

Format:

OPTIONS [{AFPAUSE / AFCONTINUE}] [,{SVCPAUSE / SVCCONTINUE}] [,NRESIDENT]

Parameters:

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

Functional Details:

The OPTIONS command can be entered in task loaded mode.

Example:

```
OPT   AFC,SVCC
```

| PAUSE |

2.34 PAUSE COMMAND

The PAUSE command pauses the currently running task.

Format:

PAUSE

Functional Details:

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

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

2.35 PREVENT COMMAND

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

Format:

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

Parameters:

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

Functional Details:

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

| PRINT |

2.36 PRINT COMMAND

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

Format:

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

Parameters:

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

Functional Details:

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

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

2.37 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 ccard punch driver does not supportt VFC.

Functional Details:

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

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

2.38 RENAME COMMAND

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

Format:

RENAME oldfd,newfd

Parameters:

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

Functional Details:

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

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

Example:

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

2.39 REPROTECT COMMAND

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

Format:

REPROTECT fd,new keys

Parameters:

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

Functional Details:

Unconditionally protected files can be conditionally reprotected or unprotected.

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

| REWIND and RW |

2.40 REWIND AND RW COMMANDS

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

Format:

REWIND [fd,] lu

or

RW [fd,] lu

Parameters:

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

Functional Details:

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

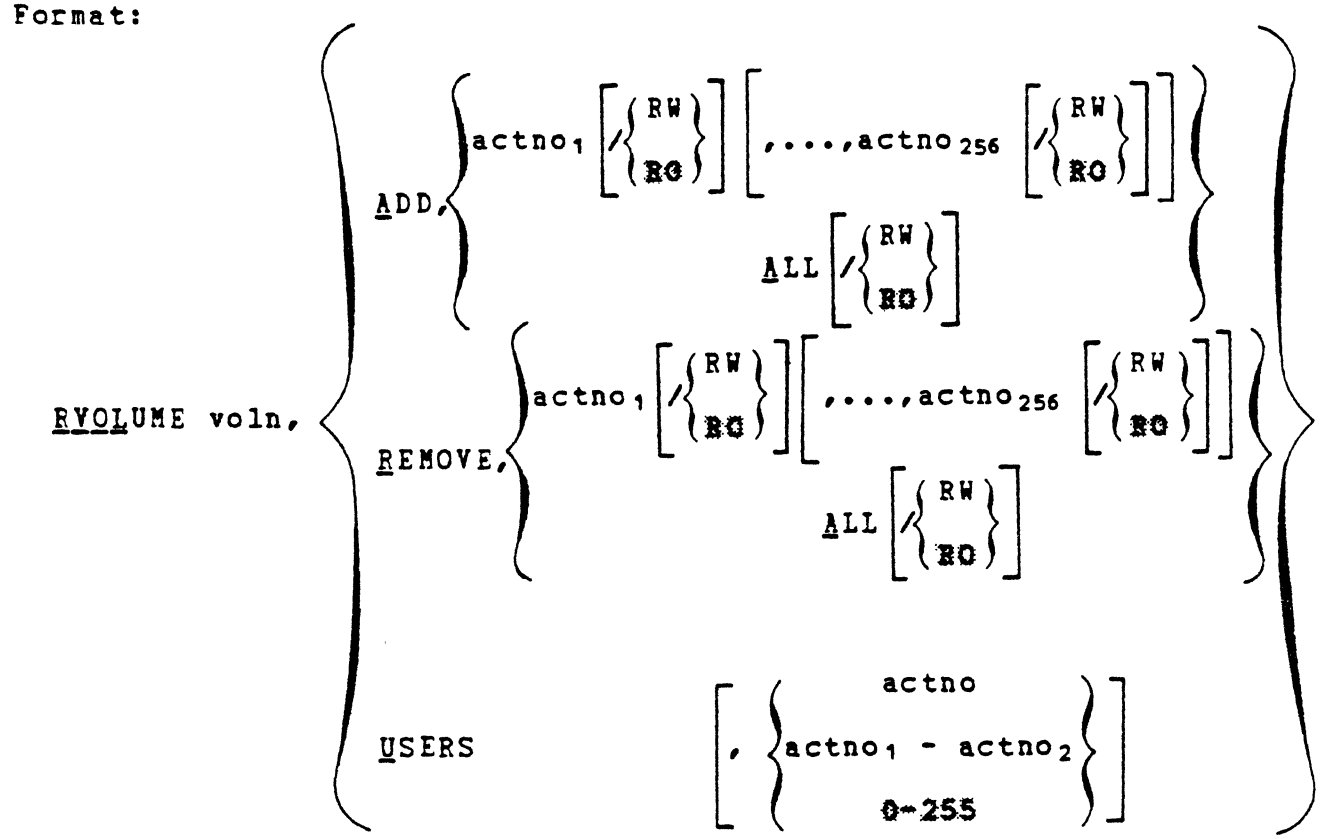
Examples:

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

2.41 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 through 255 indicating the accounts allowed/disallowed access to the restricted disk. If ALL is specified, accounts 0 through 255 have access to the restricted disk.

RW — indicates that the specified account has read/write access to the restricted disk. If this argument is omitted, only the default is read.

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.

Once a restricted disk is dismounted by the system operator, any accounts that once had access no longer have access privileges.

Examples:

```
RVOL FIXD,U
  4/RW    20/RW    77/RW    82RW
RVOL FIXD,A,ALL
RVOL FIXD,U
   0/RO    1/RO    2/RO    3/RO    4/RO    5/RO    6/RO
   7/RO    8/RO    9/RO   10/RO   11/RO   12/RO   13/RO
  14/RO   15/RO   16/RO   17/RO   18/RO   19/RO   20/RO
    .
    .
    .
 252/RO  253/RO  254/RO  255/RO
RVOL FIXD,R,ALL
RVOL FIXD,U
   82/RW
RVOL FIXD,A,ALL/RW
RVOL FIXD,U
    0/RW    1/RW    2/RW    3/RW    4/RW    5/RW    6/RW
    7/RW    8/RW    9/RW   10/RW   11/RW   12/RW   13/RW
   14/RW   15/RW   16/RW   17/RW   18/RW   19/RW   20/RW
    .
    .
    .
 252/RW  253/RW  254/RW  255/RW
RVOL FIXD,R,ALL
RVOL FIXD,U
  82/RW
```

```
------------
|   SEND   |
------------
```

2.42 SEND COMMAND

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

Format:

SEND message[;]

Parameters:

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

Functional Details:

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

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

The SEND command can be entered in task executing mode.

Example:

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

The following is received by the task:

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

2.43 SIGNOFF COMMAND

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

Format:

SIGNOFF

Functional Details:

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

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

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

2.44 SIGNON COMMAND

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

Format:

```
SIGNON userid,actno,password [,ENVIRONMENT= { fd          }]
                             {NULL[:]}
        [,CPUTIME=maxtime]
        [,classid=iocount1 [,...,classid=iocount32]]
```

Parameters:

userid	is a 1- to 8-character alphanumeric string specifying the terminal user's identification.
actno	is a 3-digit decimal number between 1 and 250 specifying the terminal user's account number.
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 at signon time. If the entire keyword parameter is omitted, MTM searches all online disks for the signon CSS procedure USERINIT.CSS/P. The system account, on the system volume, is searched last. If USERINIT.CSS is found, MTM calls the CSS and executes the routine. If it is not found, MTM enters command mode.

CPUTIME= maxtime is a decimal number specifying the maximum CPU time to which the job is limited. If this parameter is omitted, the default established at sysgen time is used. If 0 is specified, no limits are applied. The parameter can be specified as:

```
mmmm:ss
hhhh:mm:ss
ssss
```

classid= is one of the 4-character alphanumeric mnemonics specified at sysgen time associated with each specified device or file class.

iocount is a decimal number specifying the maximum number of I/O transfers associated with a particular device class to which the job is limited. If this parameter is omitted, the default established at sysgen time is used. If 0 is specified, no limits are applied to that class.

Functional Details:

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

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

Examples:

```
SIGNON ME,12,PASSWD

SIGNON ME,118,SWDOC,ENV=NULL

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

2.45 START COMMAND

The START command initiates execution of a dormant task.

Format:

$$\underline{\text{ST}}\text{ART}\left[\left\{\begin{matrix}\text{address}\\ \text{transfer address}\end{matrix}\right\}\right]\ [,\text{parameter}_1,\ldots,\text{parameter}_n]$$

Parameters:

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

Functional Details:

The START command can be entered in task loaded mode.

Examples:

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

2.46 TASK COMMAND

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

Format:

TASK [{ taskid | .BGROUND }]

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
```

2.47 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:

$$\underline{TE}\text{MPFILE lu,} \left\{ \begin{array}{l} \underline{CO}\text{NTIGUOUS,fsize} \\ \underline{IN}\text{DEX} \left[\left[\left\{ \begin{array}{c} \text{lrecl} \\ 126 \end{array} \right\} \right] \right] \left[\left[\left\{ \begin{array}{c} \text{bsize} \\ 1 \end{array} \right\} \right] \right] \left[\left[\left\{ \begin{array}{c} \text{isize} \\ 1 \end{array} \right\} \right] \right] \end{array} \right\}$$

Parameters:

lu	is a decimal number specifying the lu number to which a temporary file is to be assigned.
CONTIGUOUS	specifies that the file type to be allocated is contiguous.
fsize	is a decimal number specifying the total allocation size in 256-byte sectors. This size can be any value up to the number of contiguous free sectors existing on the specified volume at the time the command is entered.
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.
bsize	is a decimal number specifying the number of 256-byte sectors contained in a physical block to be used for buffering. If bsize is omitted, the default value is 1. bsize cannot exceed the maximum block size established at sysgen time.

isize	is a decimal number specifying the index block size in 256-byte sectors. If isize is omitted, the default value is 1. isize cannot exceed the maximum block size established at sysgen time.

Functional Details:

A temporary file is allocated on the temporary volume.

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

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

Examples:

TE 2,CO,64	Allocates, on the temporary volume, a contiguous file with a total length of 64 sectors (16kb) and assigns it to the loaded task's lu 2.
TE 14,IN,126	Allocates, on the temporary volume, an index file with a logical record length of 126 bytes. The buffer size and index block size default to one sector. The file is assigned to lu 14 of the loaded task.

| VOLUME |

2.48 VOLUME COMMAND

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

Format:

VOLUME [voln]

Parameter:

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

Functional Details:

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

The default user volume is initially set to the system volume when the user signs on. The VOLUME command can be entered in command mode, task executing mode and task loaded mode.

Example:

```
VOL
USR=MTM    SYS=MTM    SPL=M678    TEM=M301    RVL=MTM
```

2.49 WFILE COMMAND

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

Format:

WFILE [fd,] lu

Parameters:

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

Functional Details:

The WFILE command can be entered in task loaded mode.

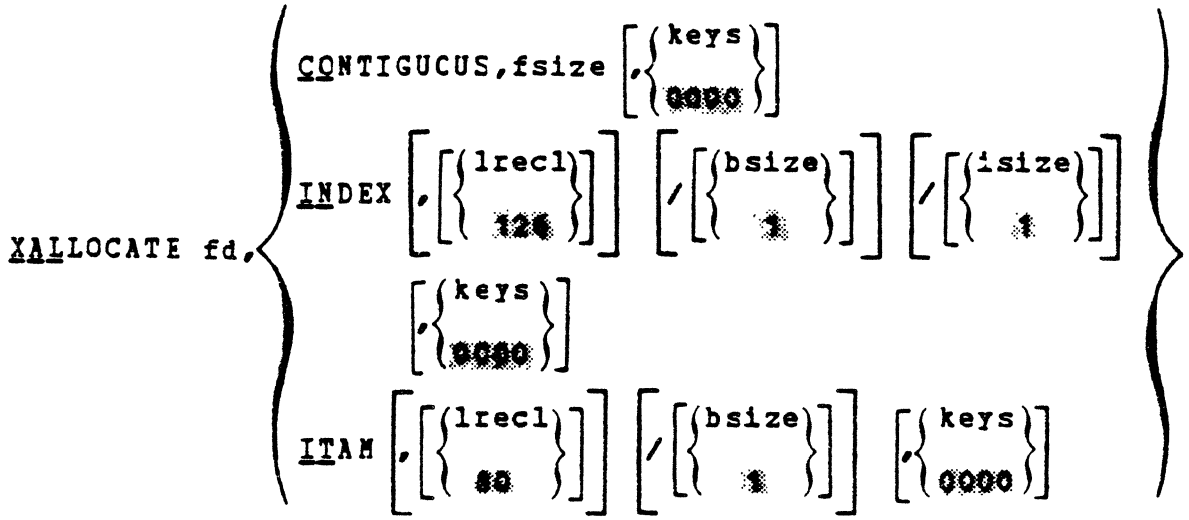
Examples:

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

2.50 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.
INDEX	specifies that the file type to be allocated is indexed.
lrecl	is a decimal number specifying the logical record length of an indexed file or communications device. It cannot exceed 65,535 bytes. Its default is 126 bytes. It may optionally be followed by a slash (/) which delimits lrecl from bsize.

bsize — is a decimal number specifying the number of 256-byte sectors contained in a physical block to be used for buffering. This parameter cannot exceed the maximum block size established at sysgen time. If bsize is omitted, the default value is one sector.

isize — is a decimal number specifying the indexed block size. If isize is omitted, the default value is one sector. Like bsize, isize cannot exceed the maximum block size established at sysgen time.

ITAM — specifies that the device to be allocated is a communications device.

keys — specifies the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword, the left byte of which signifies the write key and the right byte the read key. If this parameter is omitted, both keys default to 0.

Functional Details:

The XALLOCATE command is different from the ALLOCATE command. If the fd to be allocated is a device name instead of a filename, a DFL-ERR TYPE=VOL occurs. If a filename is specified, the XALLOCATE command reallocates the file. If fd is an existing file, it is deleted and reallocated. If fd does not exist, it is allocated. This command permits the user to have CSS procedures that allow a file or device name to be specified.

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

2.51 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').

Example:

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

CHAPTER 3
PROGRAM DEVELOPMENT

3.1 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.

3.2 CREATING A SOURCE PROGRAM

To enter 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 with the user-specified filename and program-supplied filename extension, and the editor is loaded and started.

Table 3-1 lists the program development language command syntax and program-supplied filename extensions.

TABLE 3-1 PROGRAM DEVELOPMENT LANGUAGE COMMANDS

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

TABLE 3-1 PROGRAM DEVELOPMENT LANGUAGE COMMANDS (Continued)

LANGUAGE	COMMAND SYNTAX	PROGRAM DEVELOPMENT FILENAME EXTENSIONS
CCBCL	CCBCL [[voln:] filename]	.CBL
REPORT PROGRAM GENERATOR	RPG [[voln:] filename]	.RPG
PASCAL	PAS [[voln:] filename]	.PAS

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

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

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

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

Example:

```
*FORT PROG1
** NEW PROGRAM
-EDIT
-G PROG1.FTN
-O TA = \,7,73
-C CCM = CCN:
```

```
    .
    .
    .
(edit session)
    .
    .
    .
>SAVE*
>END
-WORK FILE = M67B: PROG1.000/P
-RENUMBERED INPUT FILE AVAILABLE, M67B: PROG1.FTN/P
```

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

```
** NEW PPOGRAM
-EDIT
>
```

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

Example:

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

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

3.2.1 Creating a Data File

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

Example:

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

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

3.3 EXECUTING A PROGRAM

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

Example:

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

This example assumes that PROG1.FTN already exists as the current program. The EXEC command loads and runs the current program, PROG1.FTN, and displays a zero end of task code.

3.4 MODIFYING A PROGRAM

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

Example:

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

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

3.5 RE-EXECUTING A MODIFIED PROGRAM

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

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

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

Example:

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

This example assumes that PROG1.FTN already exists. The EXEC command, entered with PROG1, compiles, links, and then executes the image program. A zero end of task code is displayed after each process.

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

Example:

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

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

```
** CURRENT PROGPAM NOT SPECIFIED
```

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

3.6 EXECUTING MULTIPLE PROGRAMS AS A SINGLE PROGRAM

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

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

Example:

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

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

The extenstion to the EDF filename is program supplied. If you enter it, an error is generated. The user-specified or default volume is searched for ALLPROG. If it is not found, an empty file named ALLPROG is allocated, and the message, NEW ENVIRONMENT, is displayed. The EDF is now ready to receive the fds of the multiple source files. The program development ADD command is used to add source program fds to the the multi-module environment.

Example:

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

The multi-module environment is created and an EDF, ALLPROG, is allocated via the ENV command. The ADD command adds the fds, PPOG1.FTN and PROG2.CBL, to the multi-module environment.

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

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

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

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

Example:

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

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

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

```
** SYNTAX ERROR
```

Not all program development commands are available in both language and multi-module environments. Table 3-2 shows the commands that are available in the environments.

TABLE 3-2 PROGRAM DEVELOPMENT COMMAND AVAILABILITY

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

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

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

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

Example:

```
*ENV ALLPROG
*ADD PROG1.FTN
*LIST
** CURRENT ENVIRONMENT = ALLPROG
-PROG2.CBL
-PROG1.FTN
*EDIT PRCG1.FTN
-EDIT PROG1.FTN
   .
   .
   .
(edit session)
   .
   .
   .
>SAVE*
>END
*EXEC
-PERKIN-ELMER OS/32 LINKAGE EDITOR 03/242 R00-01P3
-END OF TASK CODE = 0
** EXECUTION CF ALLPRCG FOLLOWS:
-END OF TASK CODE = 0
** EXECUTION OF PROG2 FOLLOWS:
-END OF TASK CODE = 0
** EXECUTICN CF PROG1 FCLLOWS:
-END OF TASK CODE = 0
>
```

The multi-module environment is accessed via the ENV command and the EDF name, ALLPROG. PROG1.FTN is added to the multi- module environemnt. The LIST command displays the filenames remembered in the EDF. The Edit command accesses the editor to modify PROG1. When the edit session is ended,the EXEC command executes PROG2 and PROG1, displaying a zero end of task code after each successful execution.

3.7 HOW TO RECOVER FROM ERRORS

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

Example:

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

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

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

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

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

3.8 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 3-3 shows the default variable names, their settings, and lu assignments.

TABLE 3-3 PROGRAM DEVELOPMENT DEFAULT VARIABLE SETTINGS AND LOGICAL UNIT ASSIGNMENTS

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

Before running a program, ensure that the default variable and lu settings are appropriate. To change any of the variable settings, use the $SET command. The input device can be changed from the console (default) to a pre-allocated file.

Example:

```
*$SET @SSYSIN = FILE.IN
```

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

Example:

```
*$SET @SSYSPRT = FILE.OUT
```

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

You can also redefine global variables via the $GLOBAL command and then set their values via the $SET command. You can then assign their lus via the ASSIGN command.

Example:

```
*$GLOBAL @SSYSFILE
*$SET @SSYSFILE = FILE.IN
*AS 1,FILE.IN
```

The $GLOBAL command can also be used to assign lu 0 through lu 15 by defining the @lu variable and setting the variable to the desired device or file.

Example:

```
*$GLOBAL @LU1
*$SET @LU1=CON:
```

User-defined variables and their lu assignments supercede all other variable names and device lu assignments. They remain in effect until you change them, sign off, or free them via the $FREE command.

Example:

```
*AS1, FILE.IN
*$FREE @LU1
```

The $GLOBAL command can also be used to specify task start parameters to be used with the RUN or EXEC command.

Example:

```
*$GLOBAL @SOPT(32)
*$SET @SOPT = 'CROSS, SQUEZ'
```

In this example, start parameters are contained in the variable, @SOPT(32), and supercede any other start parameters.

3.9 PROGRAM DEVELOPMENT COMMANDS

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

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

3.9.1 ADD Command

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

Format:

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

Parameters:

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

Functional Details:

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

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

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

```
** SYNTAX ERROR
```

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

```
** EXTENSION OMITTED
```

The cssprod parameter must be used if the extension of the specified file differs from the language extensions listed in Table 3-1. If this parameter is omitted when you are using a nonstandard extension, the following messages are displayed:

```
** NONSTANDARD FXTENSION
** ALTERNATE CSS REQUIRED
```

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

3.9.2 COMPILE Command

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

Language Format:

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

Multi-Module Format:

```
COMPILE  [ { voln:      } ] [ { filename        } ]
         [ { user voln: } ] [ { ALL             } ]
                            [ { current program } ]
```

Parameters:

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

Functional Details:

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

If the environment is null when you enter the COMPILE command, this message is displayed:

```
** ENVIRONMENT NOT SET
```

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

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

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

```
** file.fd NOT FOUND
```

The COMPILE command functions are illustrated in Figures 3-1 and 3-2.

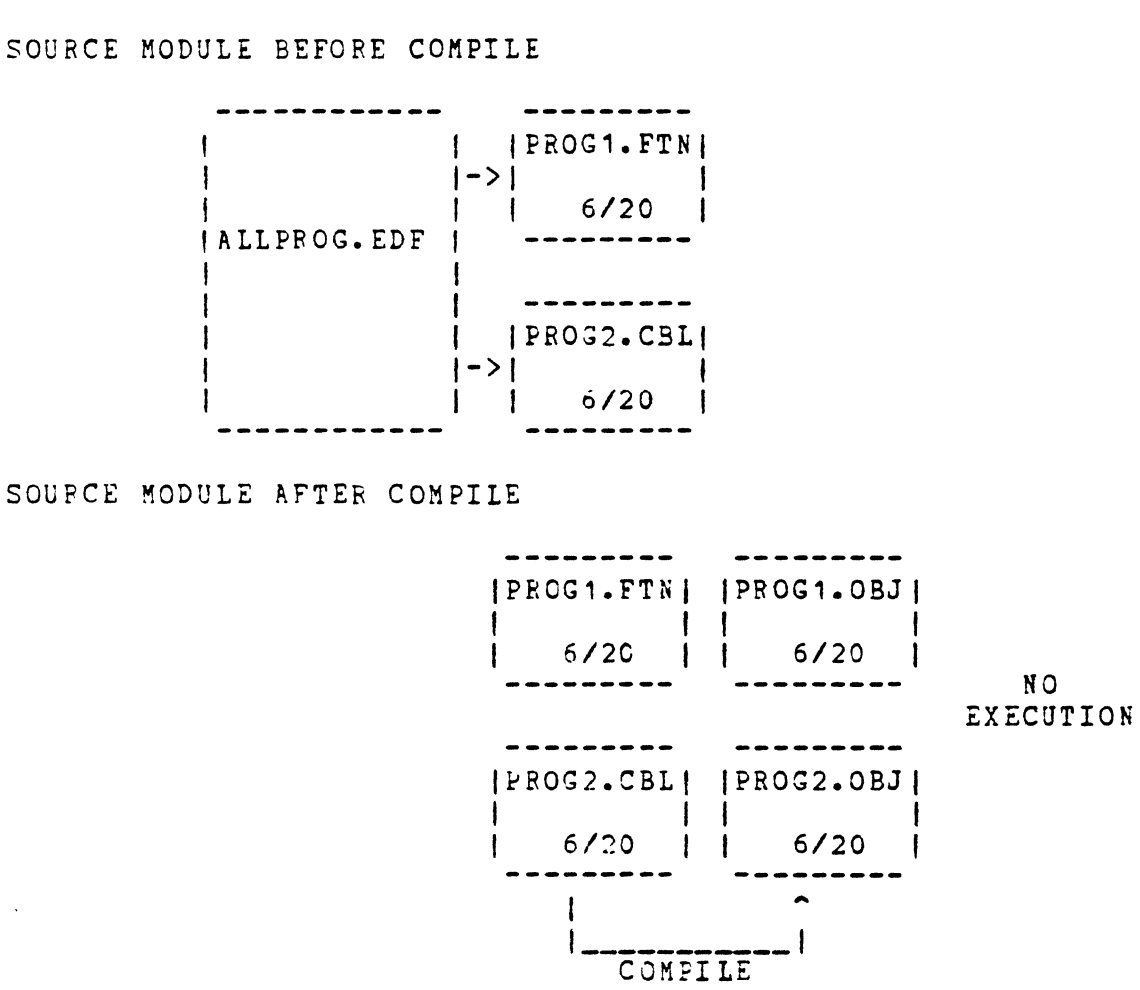


Figure 3-1 COMPILE Command Functions in the Language Environment

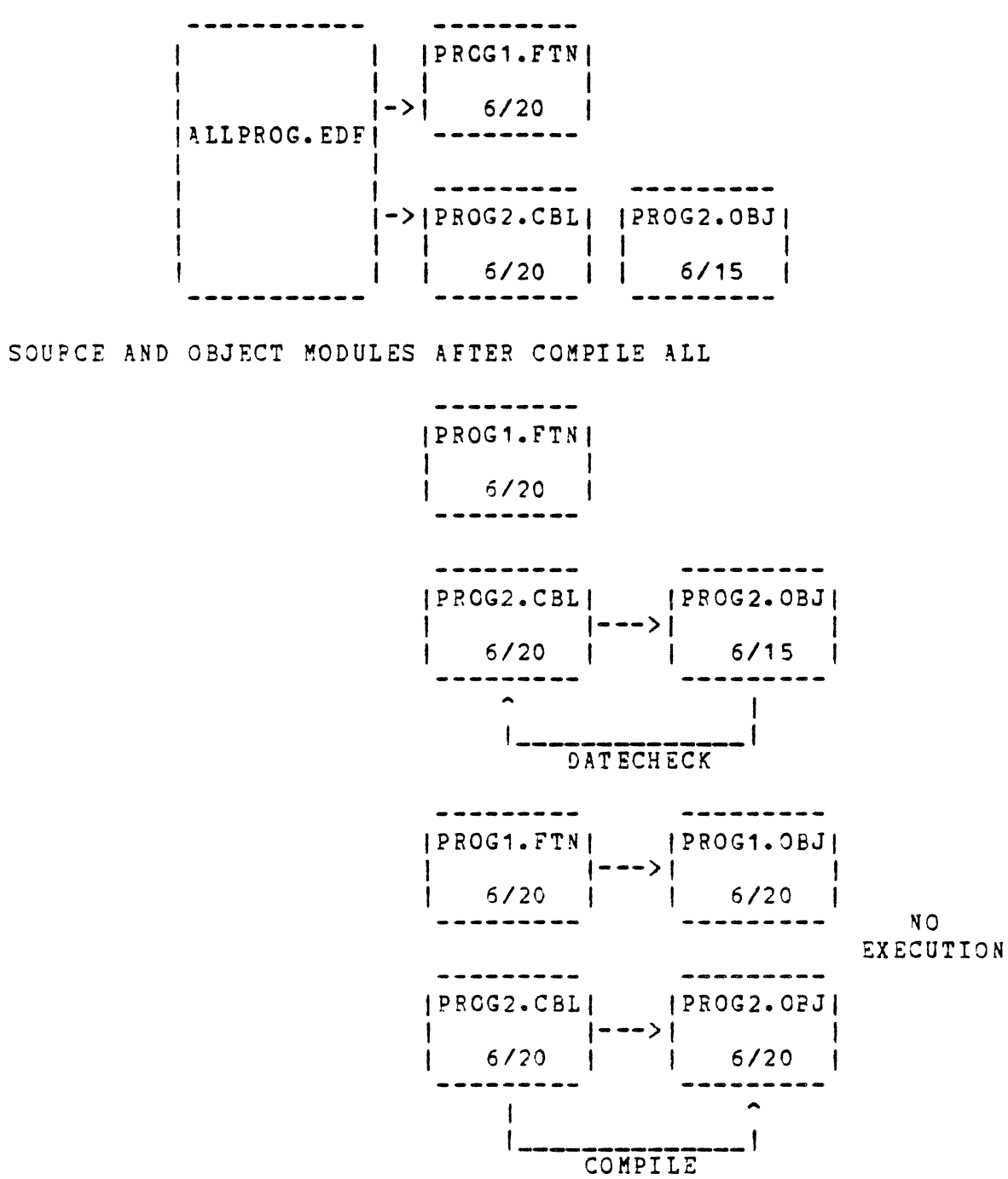


Figure 3-2 COMPILE Command Functions in the Multi-Module Environment

3.9.3 COMPLINK Command

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

Language Format:

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

Multi-Module Format:

```
COMPLINK
```

Parameters:

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

Functional Details:

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

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

```
** fd NOT FOUND
```

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

** TOO MANY ARGUMENTS

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

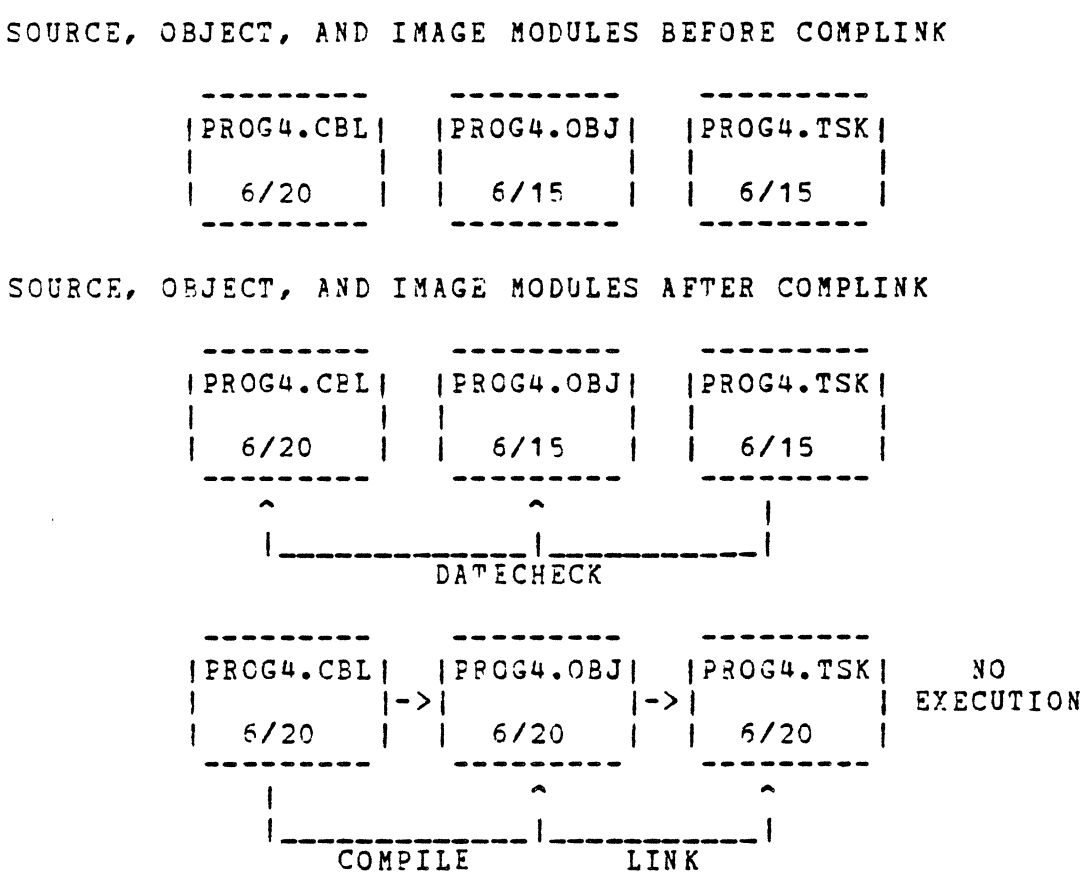


Figure 3-3 COMPLINK Command Functions in the Language Environment

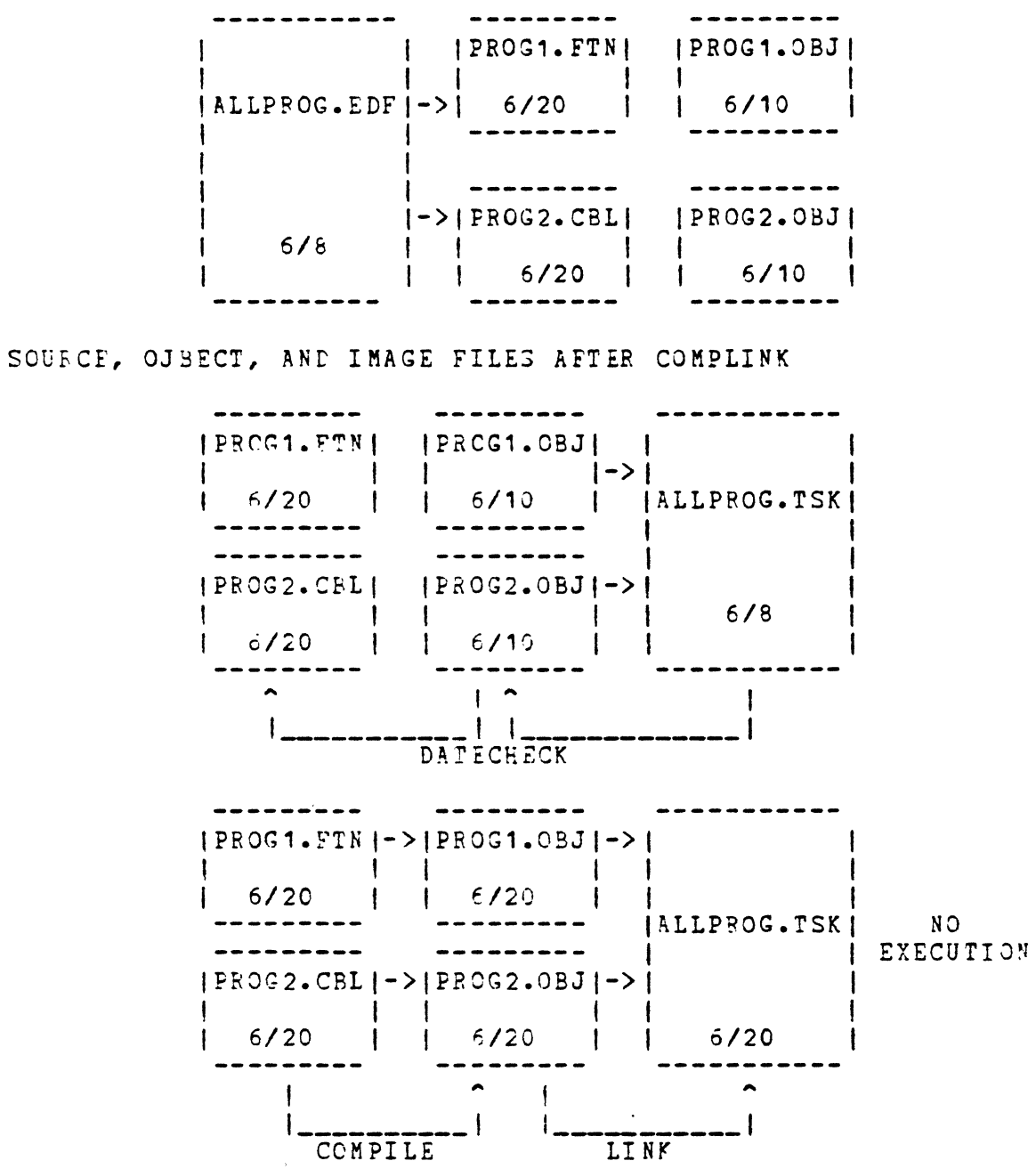


Figure 3-4 COMPLINK Command Functions in the Multi-Module Environment

3.9.4 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:

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

Parameters:

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

Functional Details:

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

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

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

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

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

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

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

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

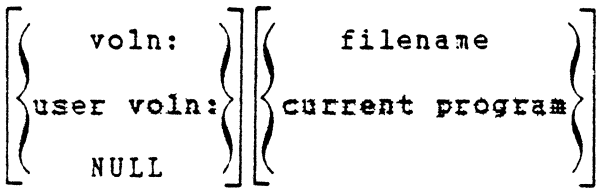
For informatin on the Edit commands, see Section 1.6.2, or the OS/32 Edit User Guide.

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

3.9.5 ENV Command

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

Multi-Module Format:



Parameters:

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

Functional Details:

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

```
** SYNTAX ERROR
```

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

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

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

```
** NO CURRENT ENVIRONMENT
```

```
----------
|  EXEC  |
----------
```

3.9.6 EXEC Command

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

Format:

```
                  [ { voln:      } ] [ { filename        } ]
EXEC              [ {user voln:} ] [ {current program} ]  [,"start parameters"]
```

Parameters:

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

Functional Details:

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

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

Start parameters can also be specified by setting a variable, @SOPT(32) with the SGLOBAL command.

Example:

```
*SGLOBAL @SOPT(32)
```

Start parameters of up to 32 characters can be specified in the SOPT variable, and they supercede all other start parameters. If the start parameters are null and SGLOBAL @SOPT (32) is null the task is started without start parameters.

EXEC command functions are shown in Figures 3-5 and 3-6.

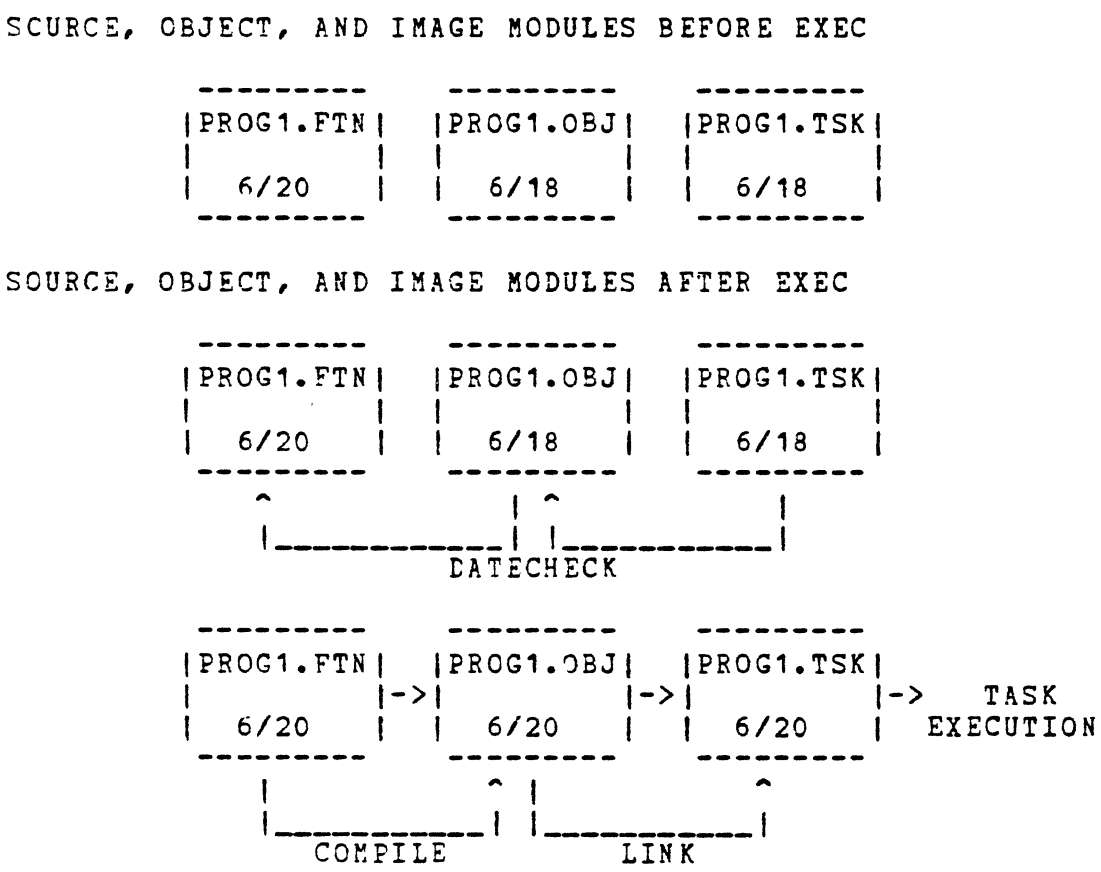


Figure 3-5 EXEC Command Functions in the Language Environment

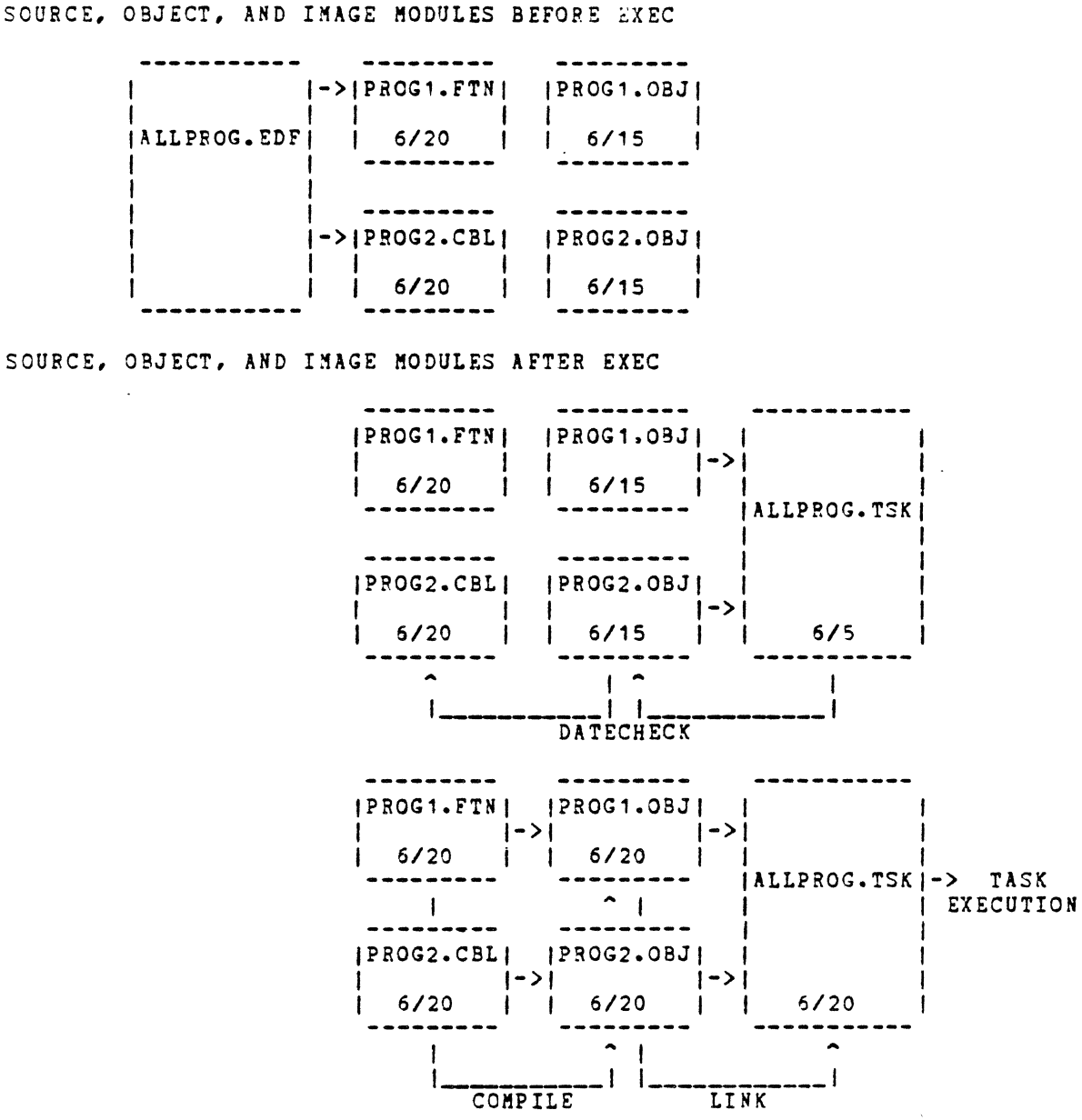


Figure 3-6 EXEC Command Functions in the Multi-Module Environment

3.9.7 LINK Command

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

Language Format:

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

Multi-Module Format:

```
LINK
```

Parameters:

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

Functional Details:

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

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

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

```
** NO ENVIRONMENT SPECIFIED
```

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

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

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

3.9.7.1 Link Sequences

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

Example:

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

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

```
>ESTABLISH TASK
>INCLUDE current program
>INCLUDE LIBRARY SYSV:F7RTL.OBJ/S
>OP DFLOAT, FLOAT, WORK=3072
>BUILD filename.TSK
>END
```

The LINK command functions are shown in figures 3-7 and 3-8.

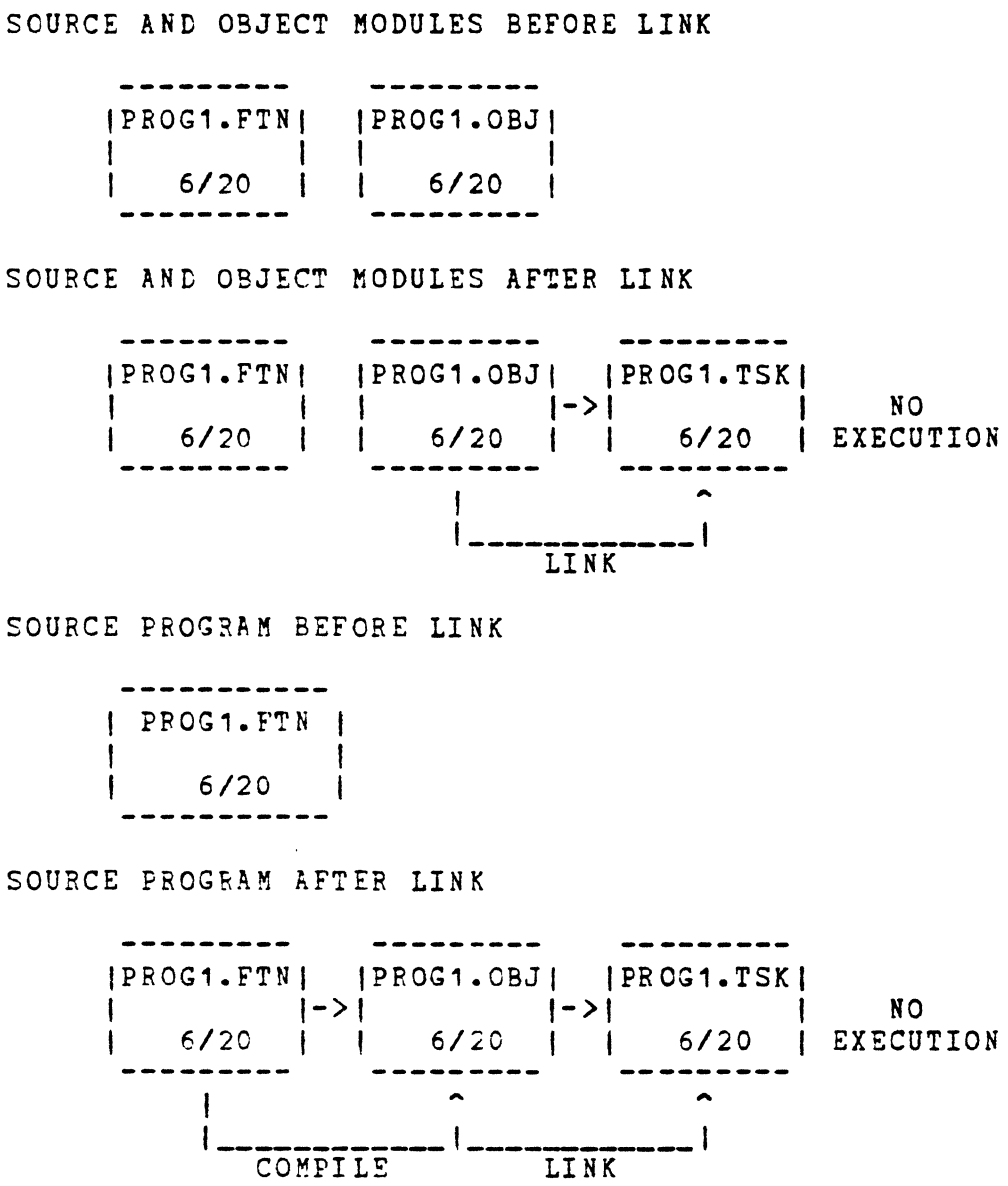
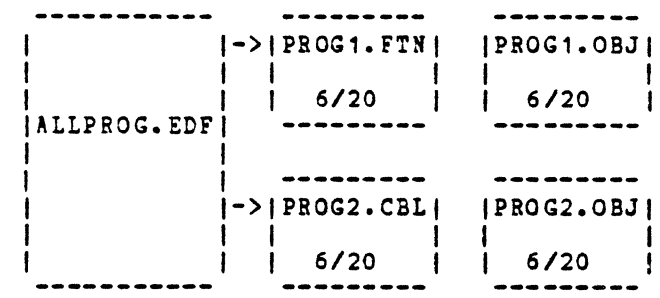
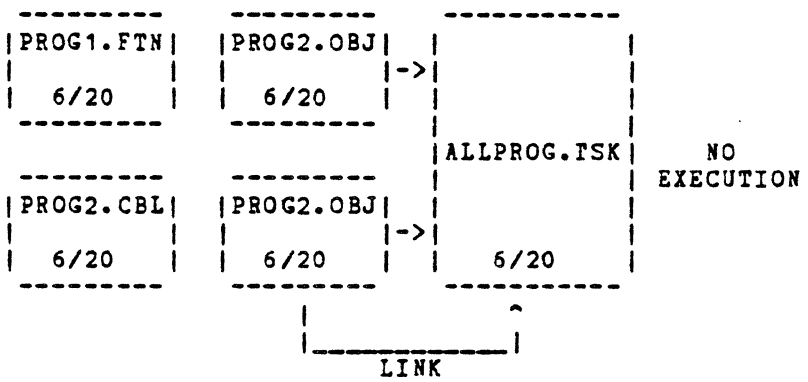


Figure 3-7 LINK Command Functions in the Language Environment

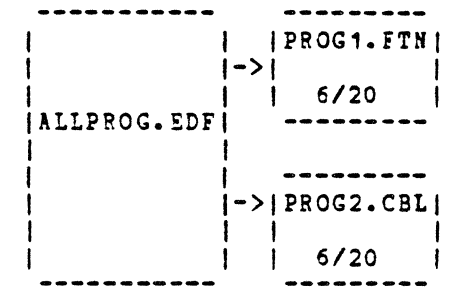
SOURCE AND OBJECT PROGRAMS BEFORE LINK



SOURCE AND OBJFCT PROGRAMS AFTER LINK



SOURCE MODULE BEFORE LINK



SOURCE MODULE AFTER LINK

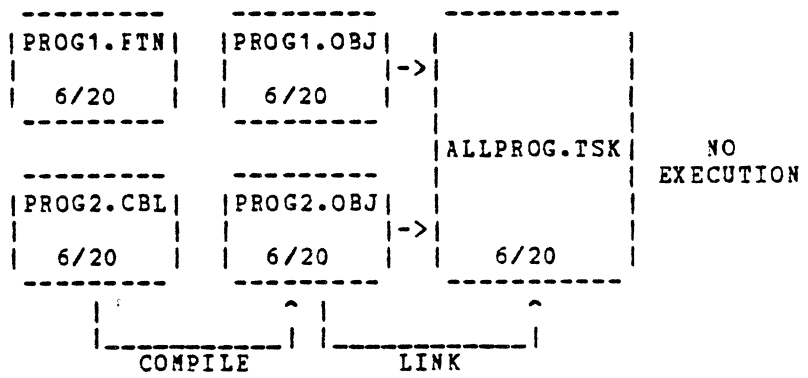


Figure 3-8 LINK Command Functions in the Multi-Module Environment

3.9.8 LIST Command

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

Format:

```
LIST
```

Functional Details:

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

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

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

```
** ENVIRONMENT EMPTY
```

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

```
** TOO MANY ARGUMENTS
```

| REMOVE |

3.9.9 REMOVE Command

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

Format:

REMOVE fd

Parameters:

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

Functional Details:

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

** FILENAME NOT IN ENVIRONMENT

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

** SYNTAX ERROR

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

** ENVIRONMENT EMPTY

3.9.10 RUN Command

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

Format:

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

Parameters:

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

Functional Details:

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

```
** fd NONEXISTENT
```

See Section 3.9.6 for more information on start parameters.

Figures 3-9 and 3-10 illustrate the RUN command functions.

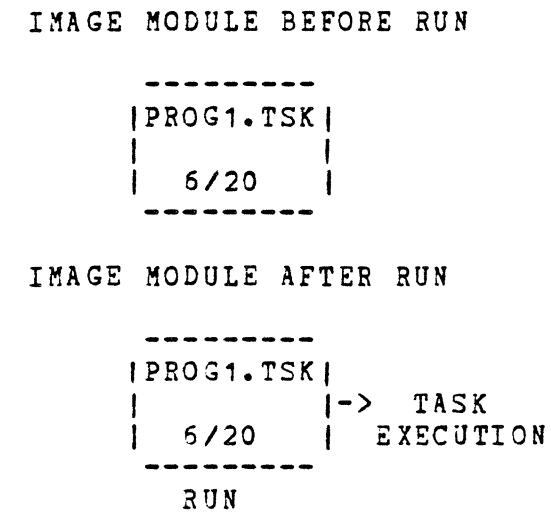


Figure 3-9 RUN Command Function in the Language Environment

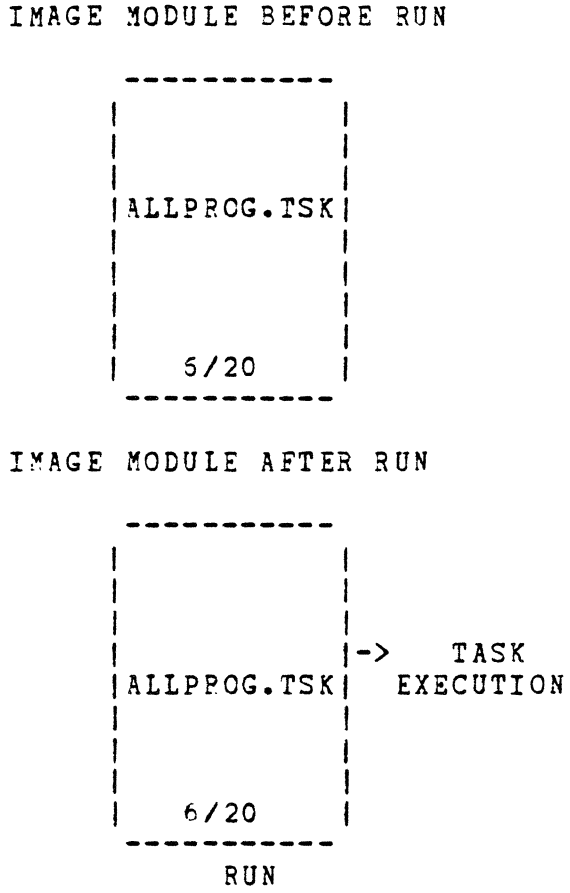


Figure 3-10 RUN Command Function in the Multi-Module Environment

Table 3-4 explains the functions of the commands used to compile, link, and run a program.

TABLE 3-4 PROGRAM DEVELOPMENT COMMANDS THAT COMPILE, LINK, AND EXECUTE

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

3.10 SAMPLE PROGRAM DEVELOPMENT SESSIONS

This section presents coding examples using the program development commands.

```
*FORT TEST                          Create language
** NEW PROGRAM                     environment with the
-EDIT                               FORTRAN language command.

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

```
$GLOBAL @SSYSIN,@SSYSOUT,@SSYSLIST
*SSET @SSYSIN=CON:                    Define and set global variables.
*SSET @SSYSOUT=CON:
*SSET @SSYSLIST=PR:
*EXEC TEST                            Execute TEST.FTN.
-FORTRAN:TEST                         Compile TEST.FTN.
** COMPILE ERRORS, LISTING ON PR:     Compilation errors in TEST.


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

*EXEC                                 Execute current program.
   .
   .
   .
(compilation sequence)                Compile.
   .
   .
   .
-FORTRAN - TEST
-END OF TASK CODE=0                   Sucessful compilation.
-LINK - TEST                          Linkedit.
   .
   .
   .
(link sequence)
   .
   .
   .
-END OF TASK CODE=0                   Successful link.
** EXECUTION OF TEST FOLLOWS:         Run.
   .
   .
   .
(execution sequence)
   .
   .
   .
-END OF TASK CODE=0

*EXEC                                 Successful execution. Re-execute.
** EXECUTION OF TEST FOLLOWS:         Ensure program is compiled
                                      and linked.
   .                                  Compile, link unnecessary.  Object
   .                                  and image up-to-date.
   .
(execution sequence)
   .
   .
   .

*RUN
-END OF TASK CODE = 0                 Successful execution.

** EXECUTION OF TEST FOLLOWS:         Rerun, compile, or link.
   .
   .
   .
```

```
(execution sequence)
   .
   .
   .
-END OF TASK CODE=0

*EXEC NEWPROG                          Execute NEWPROG.
** FILE NEWPROG.FTN NOT FOUND          System finds NEWPROG.MAC. Cannot
                                       find NEWPROG.FTN. Specify MACRO
                                       command to access NEWPROG.MAC.
*MACRO

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

*EDIT                                  Edit current program.
   .
   .
   .
(edit session)
   .
   .
   .
SAVE*
>END

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


                                       Create multi-module environ-
                                       ment with ENV command.
```

```
*ENV BIGTASK                          BIGTASK.EDF allocated.
** NEW ENVIRONMENT
*ADD SUB.CAL                          Add 3 fds to EDF.
*ADD MACRTY.CAL
*ADD FTOR.FTN
*LIST                                 List all programs in EDF.
** CURRENT ENVIRONMENT=BIGTASK.EDF
-SUB.CAL
-MACRTY.CAL
-FTOR.FTN
*ADD SUBFUNC.FTN                      Add 2 more fds 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
*EDITYSUB                             Make changes to YSUB.MAC.
   .
   .
   .
(edit session)
   .
   .
   .
SAVE*
>END                                  Create multi-module environment.

*ENV BIGTASK
*EXEC                                 Execute modules remembered in
-FORTRAN - FTOR.FTN                   BIGTASK.
-FORTRAN - SUBFUNC.FTN                FTOR.FTN and YSUB.object
-MACRO - YSUB.MAC                     modules are outdated.
-CAL - MACRTY.CAL
-LINK - BIGTASK                       Link BIGTASK.
   .
   .
   .
(link sequence)
   .
   .
   .
** 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                          Create multi-module environment.
*EXEC
-MACRO:YSUB.MAC                       YSUB.MAC object is outdated. Expand,
                                      assemble, and linkedit.
-CAL - YSUB.MAC
-LINK - BIGTASK
    .
    .
    .
(link sequence)
    .
    .
    .
** EXECUTION OF BIGTASK FOLLOWS:
    .
    .
    .
(execution sequence)
    .
    .
    .
-END OF TASK CODE=0
```

CHAPTER 4
MULTI-TERMINAL MONITOR (MTM) BATCH PROCESSING

4.1 INTRODUCTION

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

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

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

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

4.2 BATCH COMMANDS

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

To submit a batch job a user must have created a batch job file on disk. This file must have a SIGNON command as the first record, and a SIGNOFF command as the last record. The only valid commands to be used between the SIGNON and SIGNOFF commands are MTM user commands (Chapter 2), program development commands (Chapter 3), batch processing conmmands, and calls to a CSS file (Chapter 5). A batch job file is not a CSS. Therefore, CSS commands, with the exception of $IF..., $ELSE, and $ENDC, are invalid. Any command that can be used at a terminal can be used in the batch job file.

Example of a single batch job file:

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

Example CSS to build a batch job file and submit the job:

```
** ASM.CSS     [MODULE]
**
**      @ 1  (MODULE TO BE ASSEMBLED)
**
**      EXAMPLE: ASM EXIN
**
$BU     @1.JOB
SIGNON @1.JOB
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
```

4.2.1 INQUIRE Command

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

Format:

```
INQUIRE [fd][ ,fd1 | user console ]
```

Parameters:

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

Functional Details:

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

Possible responses to the INQUIRE command are:

```
JOB  fd  NOT FOUND

JOB  fd  EXECUTING

JOB  fd  WAITING  BEHIND=n

NO JOBS WITH YOUR ACCOUNT
```

Examples:

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

4.2.2 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:

```
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 a second LOG command is entered, the old log is closed and the new one is initiated.
COPY	specifies that all output is written to both the terminal and the log device.
NOCOPY	specifies that all output, except messages, is written to the log device and not the terminal. Messages from other users and the operator are written to both the terminal and the log device.
n	is a decimal number from 0 through 65,535 specifying the number of lines after which the log file is to be checkpointed. If this parameter is omitted, the default is 15 lines. If n is specified as 0, no checkpointing occurs.

Functional Details:

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

Checkpointing is only meaningful for indexed files on disk.

Example:

```
LOG PR:
```

| PURGE |

4.2.3 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.

Example:

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

4.2.4 SIGNOFF Command

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

Format:

SIGNOFF

Functional Details:

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

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

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

4.2.5 SIGNON Command

SIGNON must be the first command in a batch job.

Format:

```
SIGNON userid,actno,password [,ENVIRONMENT= { fd       }]
                             { NULL[:] }
        [,CPUTIME=maxtime]
        [,classid=iocount1 [,...,classid=iocount32]]
```

Parameters:

userid	is a 1- to 8-character alphanumeric string specifying terminal user identification.
actno	is a 3-digit decimal number from 1 through 250 specifying the terminal user's account number. If this parameter is omitted, the password parameter should also be omitted. MTM will use the account number of the user submitting the batch job.
password	is a 1- to 12-character alphanumeric string specifying the terminal user's password. This parameter should be omitted if the actno parameter is omitted. MTM will use the password of the user submitting the job.
ENVIRONMENT=	fd is the file descriptor specifying the file that will establish the user's environment at signon time.
	NULL specifies that the signon CSS procedure, USERINIT.CSS, should be ignored and the user will establish the environment at signon time. If the entire keyword parameter is omitted, MTM searches all online disks for the signon CSS procedure, USERINIT.CSS/P. The system volume, system account, is searched last. If USERINIT.CSS is found, MTM calls the CSS and executes the routine. If it is not found, MTM enters command mode.

CPUTIME= maxtime is a decimal number specifying the maximum CPU time to which the batch job is limited. If this parameter is omitted, the default established at sysgen is used. If 0 is specified, no limits are applied. The parameter can be specified as:

```
mmmm:ss
hhhh:mm:ss
ssss
```

classid= is one of the 4-character alphanumeric mnemonics, specified at sysgen, associated with each specified device or file class.

iocount is a decimal number specifying the maximum number of I/O transfers associated with a particular device class to which the batch job is limited. If this parameter is omitted, the default established at sysgen is used. If 0 is specified, no limits are applied to that class.

Functional Details:

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

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

Examples:

```
SIGNON  ME

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

S  ME,CPUTIME=120

S  ME,ENV=NULL,CPUTIME=120

S  ME,ENV=XYZ
```

| SUBMIT |

4.2.6 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 submit file the user created to submit a batch job. If this parameter is omitted, the batch submit file remains on the user volume.
PRIORITY=	priority is a decimal number from 10 through 249 specifying the priority at which a batch job will run. If this parameter is omitted, a batch job will run at the default batch priority (two priorities lower than the priority at which MTM runs) 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 the default priorities established at MTM sysgen. The u-task priorities are established at link time and can be reset with the PRIORITY parameter of the SUBMIT command. Interactive tasks run at one priority lower than MTM. Batch jobs run at two priorities lower than MTM. If the MTM sysgen priority is set to equal 1 (128), interactive jobs will run at priority +1 (129), or one lower than MTM; batch jobs will run at priority +2 (130), two 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 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 no u-task priority is specified with the SUBMIT command, the batch job runs at the default priority or the link priority, whichever is lower.

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

Example:

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

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

Submit the job from the terminal for batch processing:

```
SUBMIT TEST.JOB
```

Submit a batch submit 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
```

4.3 BATCH JOB SUBMISSION USING THE SPOOLER

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

4.4 ERROR HANDLING

Any error that occurs in a batch job 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, and the end of task code is 255. The job will continue at the command following the START; i.e., the next task will be loaded. The task end of task code can be tested by subsequent commands in the batch stream to determine if the task completed normally.

4.5 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 JCB=fd
```

CHAPTER 5
COMMAND SUBSTITUTION SYSTEM (CSS)

5.1 GENERAL DESCRIPTION

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

- the ability to switch the command input stream to a file or device;
- a set of logical operators to control the precise sequence of commands;
- parameters that can be passed to a CSS file so that general sequences can be written to take on specific meaning when the parameters are substituted; and
- the ability for one CSS file to call another, in the manner of a subroutine, so complex command sequences can be developed.

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

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

5.2 CALLING A CSS FILE

A CSS file is called and executed from the terminal by specifying the file descriptor (fd) of the CSS file. Any valid fd can be used. If the leading characters of a CSS fd are the same as a command, MTM assumes a command:

Example:

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

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

Example:

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

5.3 USE OF PARAMETERS

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

Example:

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

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

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

Example:

A CSS file ROG consists of:

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

It is called as follows:

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

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

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

Example:

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

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

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

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

Example:

Given the CSS call:

```
CSS1 arg1,arg2
```

and assuming that in file CSS1 there is another call:

```
CSS2 arg3,arg4
```

the following references can be made in CSS2:

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

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

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

Example:

A CSS file consists of:

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

If this file is called from the card reader (CR:), then lu 1 is assigned to the card reader (CR:). Likewise, a call from the magnetic tape (MAG1:) results in:

```
AS 1,MAG1:
```

5.4 USE OF VARIABLES

MTM and batch users can allocate a specified number of pseudo device variables to be used within a CSS. The maximum number of variables that can be defined is established at sysgen time. See the OS/32 Multi-Terminal Monitor (MTM) System Planning and Operator Reference Manual.

5.4.1 Types of Variables

There are two types of pseudo device variables:

- Global variables
- Local variables

Global variables exist from signon to signoff or until they are freed via the $FREE command. Local variables can be used only within the CSS levels in which they are defined. When a particular CSS level is exited, all local variables defined within it are freed.

5.4.2 Naming Variables

A variable name can consist of one through eight characters and is preceded by the commercial @ sign. The character following the @ sign must be alphabetic; the remaining characters can be alphanumeric.

Examples:

```
@A

@B19

@ABC123
```

5.4.3 Defining Variables

All variables must be defined by name using the $GLOBAL and $LOCAL commands. To set a variable to a specific value, use the $SET command.

5.4.4 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.

5.5 COMMANDS EXECUTABLE WITHIN A CSS FILE

All of the MTM supported commands can be used in a CSS file, as well as a number of commands specifically associated with the CSS facility.

All of the CSS commands start with the $ character except for the SET CODE command. The $ indicates where a CSS was used.

The CSS commands entered within a CSS file are described in the following sections. Refer to Appendix E for CSS message descriptions.

NOTE

If a task is started when CSS is running, CSS becomes dormant until the task is terminated. Execution of the CSS stream will resume after the task terminates.

5.5.1 $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:

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

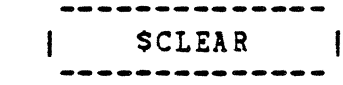
Parameters:

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.
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.



5.5.2 $CLEAR Command

The $CLEAR command terminates a CSS stream, closes all CSS files, and deactivates CSS.

Format:

$CLEAR

Functional Details:

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

5.5.3 $CONTINUE Command

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

Format:

$CONTINUE

$COPY and $NOCOPY

5.5.4 $COPY and $NOCOPY Commands

The $COPY and $NOCOPY commands control the listing of CSS commands on the terminal or log device (if from batch). $COPY initiates the listing and all subsequent commands are copied to the terminal before being executed. The $NOCOPY command deactivates the listing, but is itself listed. The $COPY command is an aid in debugging CSS job streams.

Format:

$COPY

$NOCOPY

5.5.5 $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

| $FREE |

5.5.6 $FREE Command

The $FREE command frees one or more pseudo variables.

Format:

```
$FREE varname1 [,...,varnamen]
```

Parameters:

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

Example:

```
$FREE @A
```

5.5.7 $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:

$GLOBAL varname [({length / 8})][,...,varname[({length / 8})]]

Parameters:

varname	is a 1- to 8-character name (the first character is alpabetic) 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)
```

5.5.8 $JOB and $TERMJOB Commands

The $JOB and $TERMJOB commands set the boundaries of a CSS job. The $JOB command indicates the start, and the $TERMJOB command indicates the end of a CSS job that contains all the user CSS commands and tasks.

Format:

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

Parameters:

CPUTIME= maxtime is a decimal number specifying the maximum CPU time to which the CSS routine is limited. If this parameter is omitted, the default established at MTM sysgen is used. If 0 is specified, no limits are applied.

classid= is one of the 4-character alphanumeric mnemonics specified at MTM sysgen that is associated with each specified device or file class.

iocount is a decimal number specifying the maximum CPU time to which the CSS routine is limited. If this parameter is omitted, the default established at sysgen time is used. If 0 is specified, no limits are applied to that class.

Functional Details:

The $JOB and $TERMJOB commands are not necessary in a CSS procedure. However, they help prevent errors in one CSS job from affecting other CSS jobs. If a CSS job contains an error, the statements remaining in that job are skipped until a $TERMJOB command is found. The next command executed is the first command found after a $TERMJOB command. If the next command is a $JOB command signifying the start of a new CSS job, it could be skipped because the system is looking for a $TERMJOB that signifies the end of the CSS job containing the error.

The CSS job containing an error is aborted, and the end of task code is 255. The $JOB command resets the end of task code to 0 for the next CSS job.

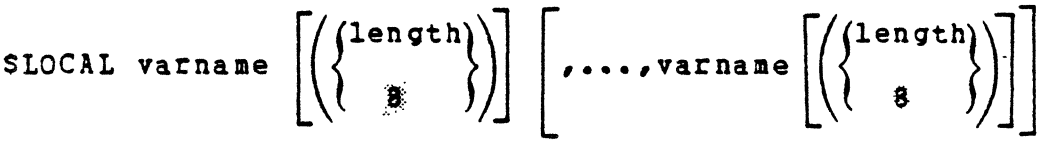
Interactive jobs have no default limits established at sysgen time. However, the user can specify CPU time and I/O transfer limits for a particular job through the $JOB command.

Any limits in the $JOB command found in a batch stream are ignored if limits were already specified in the SIGNON command.

5.5.9 $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)
```

5.5.10 $PAUSE Command

The $PAUSE command suspends execution of a CSS procedure.

Format:

$PAUSE

Functional Details:

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

| $SET |

5.5.11 $SET Command

The $SET command establishes the value of a named pseudo device variable.

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 = @A1'.MAC'

$SET @A = @1

$SET @A = 'A''B'
```

5.5.12 SET CODE Command

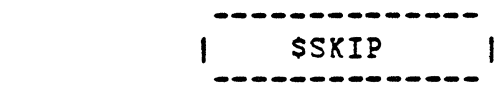
The SET CODE command modifies the end of task code of the currently selected CSS task.

Format:

SET CODE n

Parameter:

n	is a decimal number from 1 through 254.



5.5.13 $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

5.5.14 $WAIT Command

The $WAIT command suspends execution of a CSS for a specified period of time.

Format:

$WAIT [n]

Parameter:

n	is a decimal number from 1 through 900 specifying the number of seconds CSS execution will be suspended. If this parameter is omitted, the default is 1 second.

Functional Details:

The $WAIT command will only function from a CSS routine.

When the $WAIT command is entered and the user does not want to wait the specified time, a $CONTINUE command can be entered.

```
--------------
|   $WRITE   |
--------------
```

5.5.15 $WRITE Command

The $WRITE command writes a message to the terminal or log device for both interactive and batch jobs.

Format:

$WRITE text [;]

Functional Details:

The message is output to the terminal or log device. It begins with the first nonblank character after $WRITE and ends with a semicolon or carriage return. The semicolon is not printed.

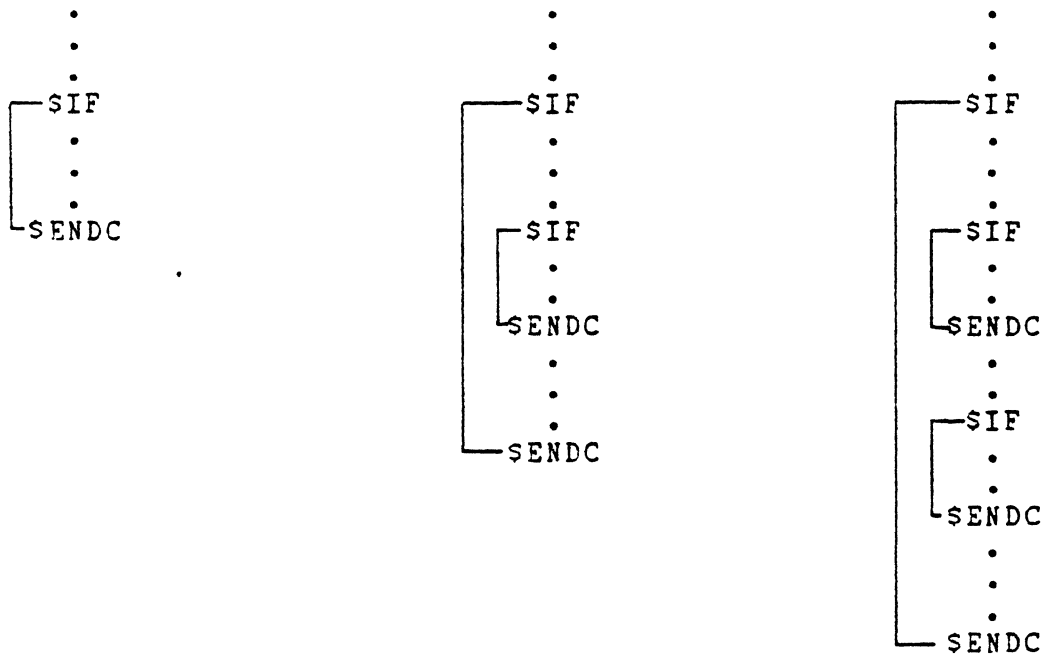
5.6 LOGICAL IF COMMANDS

The logical IF commands all start with the three characters, $IF, and allow one argument; e.g., $IFE 225, $IFX B.CSS, $IFNULL @1.

Each logical IF command establishes a condition that is tested by the CSS processor. If the result of this test is true, commands up to a corresponding $ELSE or $ENDC command are executed. If the result is false, these same commands are skipped.

The $ENDC command delimits the range of a logical IF; however, nesting is permitted so each $IF must have a corresponding $ENDC.

In the following examples, the ranges of the various logical IF commands are indicated by brackets:



There is no restriction on the depth of nesting. Logical IF commands are used within a CSS file. However, they differ from previous CSS commands in that each one tests a specific built-in, defined condition rather than causes a specific action.

The logical IF commands fall into three categories:

- End of task code testing
- File existence testing
- Parameter existence testing

5.6.1 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.
EOT (SVC 3,n)	When any task terminates by executing the end of task program command (SVC 3,n), the end of task code for that task is set to n.
CANCEL	When a task is cancelled, the end of task code is set to 255.

The six commands available for testing the end of task code of the currently selected CSS task are as follows:

$IFE n	Test end of task code equal to n
$IFNE n	Test end of task code not equal to n
$IFL n	Test end of task code less than n
$IFNL n	Test end of task code not less than n
$IFG n	Test end of task code greater than n
$IFNG n	Test end of task code not greater than n

In all cases, if the results of the test are "false", CSS skips commands until the corresponding $ELSE or $ENDC. If a CSS attempts to skip beyond EOF, a command error is generated.

5.6.2 File Existence Testing Commands

There are two commands dealing with file existence:

$IFX fd	Test fd for existence
$IFNX fd	Test fd for nonexistence

If the result of the test is false, CSS skips to the corresponding $ELSE or $ENDC command. If a CSS attempts to skip beyond EOF, an error is generated.

5.6.3 Parameter Existence Testing Commands

There are two commands dealing with the existence of parameters:

$IFNULL	@n	Test @n null
$IFNNULL	@n	Test @n 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:

$IFN @$_1$@$_2$@$_3$

In effect, this tests the logical OR of @1, @2, and @3 for nullity. If any of the three is present, the test result is false.

| $ELSE |

5.6.4 $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

5.7 $GOTO AND $LABEL COMMANDS

The $GOTO command is used to skip forward within a CSS procedure. The $LABEL is used to define the object of a $GOTO.

Format:

```
$GOTO  label

$LABEL label
```

Parameters:

Label	is from one to eight alphanumeric characters, the first of which must be alphabetic.

Functional Details:

The $GOTO command causes all subsequent commands to be ignored until a $LABEL command with the same label as the $GOTO command is encountered. At that point, command execution resumes.

The $GOTO cannot branch into a logical IF command range but can branch out from one.

An example of an illegal $GOTO is:

```
$IF      Condition
$GOTO    OUTIF
  .
  .
  .
$ENDC
$IF      Condition
$LABEL   OUTIF
```

The $LABEL occurs within an IF block (the second IF condition) that was not active when $GOTO was executed.

The following is valid, however:

```
$IF       Condition
$GOTO     OUTIF
   .
   .
   .
$ENDC
$IF       Condition
   .
   .
   .
$ENDC
$LABEL     OUTIF
```

5.8 $IFEXTENSION COMMAND

The $IFEXTENSION command is used to test for the existence of an extension for a given fd. If the extension exists, subsequent commands are executed up to the next $ELSE or $ENDC command. If an extension does not exist, subsequent commands are skipped up to the next $ELSE or $ENDC command.

Format:

$IFEXTENSION fd

Parameter:

fd	is the file descriptor to be tested to determine if an extension is included.

Functional Details:

$IFEX (with no fd) is always considered false.
$IFNEX (with no fd) is always considered true.

| $IFVOLUME |

5.9 $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.

5.10 LOGICAL IF COMMANDS COMPARING TWO ARGUMENTS

The following logical IF commands are used to compare two arguments. They differ from the other logical IF commands in that they do not test specific built-in conditions but, rather, test conditions provided by the user. These commands are available only with MTM.

```
$IF . . . EQUAL
$IF . . . NEQUAL
$IF . . . GREATER
$IF . . . NGREATER
$IF . . . LESS
$IF . . . NLESS
```

For each of the logical commands, two arguments are compared according to the mode. There are three valid modes:

- Character
- Decimal
- Hexadecimal

For character mode, the comparison is left-to-right and is terminated on the first pair of characters that are not the same. If one string is exhausted before the other, the short string is less than the long string. If both strings are exhausted at the same time, they are equal. For character mode, the arguments can be enclosed in double quotes if they contain blanks. The quotes are not included in the compare.

For decimal and hexadecimal mode, the comparison is performed by comparing the binary value of the numbers.

If after comparing the arguments for each of the commands, the condition is determined to be true, subsequent commands are executed up to the corresponding $ELSE and $ENDC. If the condition is false, commands are skipped up to the corresponding $ELSE or $ENDC.

```
--------------
|    SIF     |
--------------
```

5.10.1 $IF...EQUAL, $IF...NEQUAL Commands

The $IF...EQUAL command is used to determine if two arguments are equal, while the $IF...NEQUAL is used to determine if two arguments are not equal.

Format:

$$\$IF \left\{ \begin{matrix} \underline{C}HARACTER \\ \underline{D}ECIMAL \\ \underline{H}EXADECIMAL \end{matrix} \right\} arg_1 \ \underline{E}QUAL \ arg_2$$

$$\$IF \left\{ \begin{matrix} \underline{C}HARACTER \\ \underline{D}ECIMAL \\ \underline{H}EXADECIMAL \end{matrix} \right\} arg_1 \ \underline{N}EQUAL \ arg_2$$

5.10.2 $IF...GREATER, $IF...NGREATER Commands

The $IF...GREATER command is used to determine if arg1 is greater than arg2. The $IF...NGREATER command is used to determine if arg1 is not greater than arg2.

Format:

$$\$IF \left\{ \begin{matrix} \underline{C}HARACTER \\ \underline{D}ECIMAL \\ \underline{H}EXADECIMAL \end{matrix} \right\} arg_1 \ \underline{G}REATER \ arg_2$$

$$\$IF \left\{ \begin{matrix} \underline{C}HARACTER \\ \underline{D}ECIMAL \\ \underline{H}EXADECIMAL \end{matrix} \right\} arg_1 \ \underline{N}GREATER \ arg_2$$

5.10.3 $IF...LESS, $IF...NLESS Commands

The $IF...LESS command is used to determine if arg1 is less than arg2. The $IF...NLESS command is used to determine if arg1 is not less than arg2.

Format:

$$\$IF \left\{ \begin{array}{l} \underline{C}HARACTER \\ \underline{D}ECIMAL \\ \underline{H}EXADECIMAL \end{array} \right\} arg_1 \ \underline{LE}SS \ arg_2$$

$$\$IF \left\{ \begin{array}{l} \underline{C}HARACTER \\ \underline{D}ECIMAL \\ \underline{H}EXADECIMAL \end{array} \right\} arg_1 \ \underline{NLE}SS \ arg_2$$

CHAPTER 6
SPOOLING

6.1 INTRODUCTION

The OS/32 package comes with a spooler task for both input and output spooling. At system generation (sysgen) time, the spool option must be included and the pseudo print devices declared in order to incorporate the spooling facility.

6.2 INPUT SPOOLING

The input spooling feature enables a batch stream of cards such as source programs, operator commands, command substitution system (CSS) files, or other user data to be copied onto a disk file for immediate or subsequent processing. There are two control commands available for input spooling:

```
/@INPUT

/@SUBMIT
```

In all cases, each deck of cards to be copied must end with a control card with the /@ appearing in columns 1 and 2.

6.2.1 Input Card

The /@INPUT card copies all the data between the /@INPUT and the /@ cards to a disk file. The resulting file can be explicitly assigned and read by the user in order to access the spooled information.

Cards to be copied must be preceded by a control card with the format:

```
/@INPUT  fd/actno ,DELETE
```

Parameters:

fd	is the file descriptor of the disk file in the form of voln:filename.ext. The only required field is filename. If voln is omitted, the default spool volume is used.
actno	is the account number the terminal user signs on with.
DELETE	specifies that if a file with the same name and account number already exists, that file is deleted and reallocated.

CAUTION

IF THE WRONG ACCOUNT NUMBER IS ENTERED,
THE USER MIGHT DELETE ANOTHER USER FILE.

Example:

A certain task (TEST.TSK) requires five input data records in order to execute. The following statements place these five input statements on a disk file named TEST.DTA (associated with account number 12) and delete and reallocate TEST.DTA if it already exists:

```
/@IN TEST.DTA/12,D
4 INPUT TEST
122736
545627
889710
632192
/@
```

6.2.2 Submit Card - Adding Batch Jobs to the Batch Queue

The Spooler can also be used to submit batch jobs to the multi-terminal monitor (MTM). This is done through the /@SUBMIT command which copies card files to the disk and submits the file as a batch job. The commands 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 card with the format:

Format:

```
/@SUBMIT  fd/actno ,DELETE
```

Parameters:

fd	is the name of the command file; i.e., the batch job, that is to be placed on the batch queue.
actno	is the account number the terminal user signs on with.
DELETE	specifies that if a file with the same name and account number exists, that file is to be deleted and reallocated.

Each deck of cards must end with a control card with /@ appearing on columns 1 and 2.

Refer to the OS/32 System Support Utilities Reference Manual for more information on the Spooler.

There are two ways to submit a batch job using the Spooler.

Method 1:

First a CSS file is copied from a card file to a disk file named TEST.CSS (associated with 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,D
LO TEST
AS 1,TEST.DTA
AS 3,PR:
AS 5,MAG1:
START
/@
```

The CSS file TEST.CSS now can be executed by the batch job TEST.JOB. 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 this job as follows:

```
/@SUBMIT TEST.JOB/12,D
SIGNON ME,12,PASSWD
LOG PR:
TEST.CSS
SIGNOFF
/@
```

Method 2:

Only one step is required to build the file TEST.JOB and submit it as a batch job. The commands are executed in sequence. If the file TEST.JOB already exists on the disk, it is deleted and reallocated. After this batch job completes, the file TEST.JOB remains on the disk:

```
/@SUBMIT TEST.JOB/12,D
SIGNON ME,12,PASSWD
LOG PR:
LO TEST
AS 1,TEST.DTA
AS 3,PR:
AS 5,MAG1:
START
SIGNOFF
/@
```

6.3 OUTPUT SPOOLING

Output spooling allows more than one task to be assigned to one or more print or punch devices simultaneously. Data to be printed or punched is written to disk files where it is then copied by the Spooler to the available print or punch devices on a task priority basis.

To make use of the output Spooler, assign any logical units (lu) to be printed or punched to one or more pseudo devices. As soon as the lu is closed, the Spooler automatically will print or punch the results. Printing or punching may be delayed because of a backlog to the device.

There is no limit to the number of tasks or logical units that can be assigned to a pseudo device. After the user makes an lu assignment, the following occurs internally: the operating system automatically intercepts all assignments to a pseudo device and allocates an indexed file called a spool file on the spool volume. Subsequent output calls cause data to be written to this file and not to the device. The Spooler supports both image and formatted writes.

When the lu assigned to the spool file is closed, the filename, task name, and priority are placed into the Spooler print or punch queue. The queue is maintained as a file on the spool volume. If there is an entry on the queue, the output Spooler begins printing or punching and stays active as long as there is something on the queue. Files are spooled and output on a task priority basis. The user must ensure that sufficient disk space is available to accommodate output spooling. The user task is responsible for handling end of medium (EOM) status while writing to spool files within their own standard I/O error recovery routines.

Printing multiple copies of a disk file or punching multiple copies of a card deck is accomplished through use of the Spooler. To print or punch a disk file using the Spooler, issue a command through MTM from the terminal. This is done with the PRINT and PUNCH commands. See Sections 2.36 and 2.37.

If the device specified in a PRINT or PUNCH command does not support printed output or output punching respectively, the output will be generated in the way that is supported on the specified device.

For print files, a header page precedes each file printed. The header page has the format:

```
USERID

ACCOUNT NUMBER

TIME OF DAY

DATE
```

When a file is directed to a card punch file, each output record is 80 bytes in length. A header card precedes the punched output; a trailer card terminates the punched output. Header suppression is not supplied.

Example:

To list and punch a file named TEST.CSS in account number 12 on the volume MTM using the 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
```

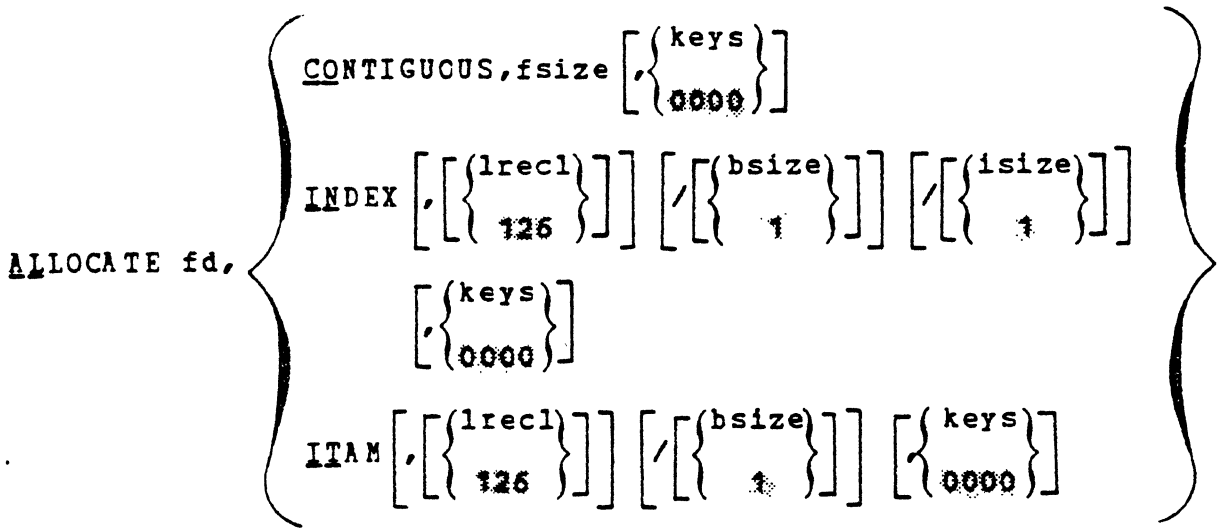
6.4 SPOOLING ERRORS

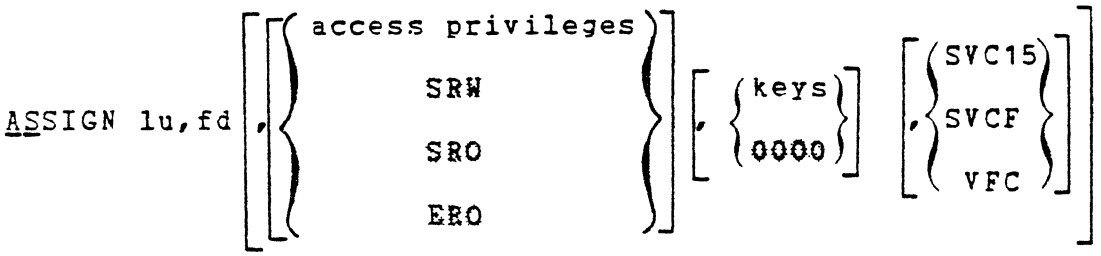
The following message is generated by the operating system in response to a spooler command.

```
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.

APPENDIX A
MTM COMMAND SUMMARY





PFILE fd, lu

BIAS {address / *}

BREAK

BRECORD [fd,] lu

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

CANCEL

CLOSE {lu1 [,lu2,...lun]}
      {ALL              }

CONTINUE [address]

DELETE fd1 [,fd2,...,fdn]

DISPLAY ACCOUNTING [,{fd          }]
                     {user console}

DISPLAY DEVICES [,{fd          }]
                  {user console}

DISPLAY DFLOAT [,{fd          }]
                 {user console}

DISPLAY FILES ,[{:                }] [filename] [.[ext]]
               [{voln:            }]
               [{default user vol }]

              [/ [{P}] [,{fd          }]]
              [  [{S}]   {user console}  ]
              [  [{G}]                   ]

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 / $VARIABLE}

EXAMINE address$_1$ [{,n / /address$_2$ / ,1}] [,{fd / user console}]

FFILE [fd,] lu

FRECORD [fd,] lu

HELP [{mnemonic / *}]

INIT fd [,{segsize increment / 1}]

```
INQUIRE [fd { ,fd1          }]
            { user console }

LOAD [taskid,] fd [,segsize increment]

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

MESSAGE {userid   } message
        {.OPERATOR}

MODIFY address,[{data1}] [,data2,...,datan]
               [{0    }]

OPTIONS [{AFPAUSE    }] [,{SVCPAUSE    }] [,NRESIDENT]
        [{AFCONTINUE }]    {SVCCONTINUE }

PAUSE

          {MESSAGE  }
          {PROMPT   }
PREVENT   {ETM      }
          {SVARIABLE}

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

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

PURGE fd

RENAME oldfd,newfd

REPROTECT fd,new keys
```

REWIND [fd,] lu

or

RW [fd,] lu

RVOLUME voln, { ADD, { actno$_1$ [/{RW / RO}] [,...,actno$_{256}$ [/{RW / RO}]] / ALL [/{RW / RO}] } | REMOVE, { actno$_1$ [/{RW / RO}] [,...,actno$_{256}$ [/{RW / RO}]] / ALL [/{RW / RO}] } | USERS [, {actno / actno$_1$ - actno$_2$ / 0-255}] }

SEND message[;]

SET LOG [fd] [,[{NOCOPY / COPY}]] [,[[{n / 15}]]]

SIGNOFF

SIGNON userid,actno,password [,ENVIRONMENT= {fd / NULL[:]}]
[,CPUTIME=maxtime]
[,classid=iocount$_1$ [,...,classid=iocount$_{32}$]]

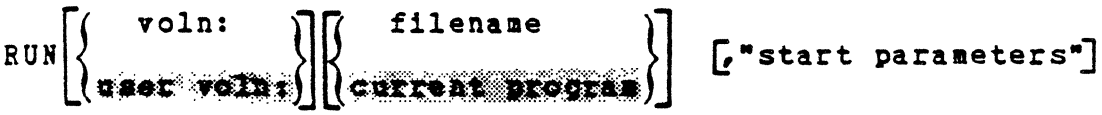
START [{address / transfer address}] [,parameter$_1$,...,parameter$_n$]

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

|        [( taskid   )]
| TASK   [{          }]
|        [( .BGROUND )]

                  ( CONTIGUOUS,fsize
TEMPFILE lu,      {
                  ( INDEX [,[( lrecl )]] [/[( bsize )]] [/[( isize )]]
                            [{       }]    [{       }]    [{       }]
                            [( 126   )]    [(   1   )]    [(   1   )]

VOLUME [voln]

WFILE [fd,] lu

                 ( CONTIGUCUS,fsize [,( keys )]
                 (                  [ {      }]
                 (                  [ ( 0000 )]
                 (
                 ( INDEX [,[( lrecl )]] [/[( bsize )]] [/[( isize )]]
                 (         [{       }]    [{       }]    [{       }]
XALLOCATE fd,    {         [( 126   )]    [(   1   )]    [(   1   )]
                 (
                 (       [,( keys )]
                 (       [ {      }]
                 (       [ ( 0000 )]
                 (
                 ( ITAM [,[( lrecl )]] [/[( bsize )]] [,( keys )]
                 (        [{       }]    [{       }]   [ {      }]
                 (        [( 126   )]    [(   1   )]   [ ( 0000 )]

XDELETE fd1 [,fd2...,fdn]
```

APPENDIX B
PROGRAM DEVELOPMENT COMMAND SUMMARY

```
ADD fd [,cssprod]

COMPILE [{voln:      }][{filename       }]
        [{user voln: }][{current program}]

COMPILE [{voln:      }][{filename       }]
        [{user voln: }][{ALL            }]
                       [{current program}]

COMPLINK [{voln:      }][{filename       }]
         [{user voln: }][{current program}]

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

ENV  [{voln:      }][{filename       }]
     [{user voln: }][{current program}]
     [{NULL       }]

EXEC [{voln:      }][{filename       }] [,"start parameters"]
     [{user voln: }][{current program}]

LINK [{voln:      }][{filename       }]
     [{user voln: }][{current program}]

LIST
```

REMOVE fd


RUN
voln:
user voln:
filename
current program
["start parameters"]

APPENDIX C
CSS COMMAND SUMMARY

$BUILD{fd / lu} [,APPEND]
    .
    .
    .
$ENDB

$CLEAR

$CONTINUE

$COPY

$NOCOPY

$ELSE

$ENDC

$EXIT

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

$GLOBAL varname [({length / 8})][,...,varname[({length / 8})]]

$GOTO label

$LABEL label

SIF { CHARACTER / DECIMAL / HEXADECIMAL } arg_1 EQUAL arg_2

SIF { CHARACTER / DECIMAL / HEXADECIMAL } arg_1 NEQUAL arg_2

SIF { CHARACTER / DECIMAL / HEXADECIMAL } arg_1 GREATER arg_2

SIF { CHARACTER / DECIMAL / HEXADECIMAL } arg_1 NGREATER arg_2

SIF { CHARACTER / DECIMAL / HEXADECIMAL } arg_1 LESS arg_2

SIF { CHARACTER / DECIMAL / HEXADECIMAL } arg_1 NLESS arg_2

| SIFE n

SIFEXTENSION fd

| SIFG n

$IFL n

$IFNE n

$IFNG n

$IFNL n

$IFNULL @n

$IFNNULL @n

$IFVOLUME fd

$IFX fd

$IFNX fd

$JOB
CPUTIME=maxtime
[,classid=iocount$_1$][,...,classid=iocount$_{32}$]
.
.
.
$TERMJOB

$LOCAL varname [({length / 8})] [,...,varname [({length / 8})]]

$PAUSE

$SET varname=e

SET CODE n

SSKIP

SWAIT [{n / 1}]

SWRITE text [;]

APPENDIX D
MTM MESSAGE SUMMARY

ACCESS PRIVILEGE ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a valid segment in an invalid mode; i.e., store into a write protected segment; execute instructions from an execute protected segment; load from a read protected segment.

ACCT-ERR

The account number specified is not between 0 and 255.

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 3200 Series Machines.

ALLO-ERR TYPE=NAME

A desired filename currently exists on the specified volume.

The block size of an indexed file exceeds limit established at sysgen time.

For an indexed file, a zero logical record length or data block size was specified.

ALLO-ERR TYPE=TYPE

The volume specified is not a direct access device.

ALLC-ERR TYPE=VOL

The volume name specified, or the name it defaulted to, is not the name of any of the disks currently online.

ACCT-ERR

The account number specified is not between 0 and 255.

ARGS-ERR

The amount of space between CTOP and UTOP is insufficient for placement of START command arguments by the command processor.

ARITHMETIC FAULT AT XXXXXX

A fixed or floating point error was detected at address xxxxxx, or an attempt was made to divide by zero. This only occurs on Perkin-Elmer Models 7/32 and 8/32 Machines.

ASGN-ERR

The assign failed for reason denoted by TYPE field.

ASGN-ERR TYPE=BUFF

An attempt was made to assign a file when there was insufficient system space available to accommodate the FCB.

ASGN-ERR TYPE=LU

An attempt was made to assign to an lu that is greater than the maximum lu number specified at Link time.

ASGN-ERR TYPE=NAME

An assignment is being directed to a nonexistent file.

ASGN-ERR TYPE=PRIV

A file, currently assigned to an lu with a given privilege, is assigned to another lu; e.g., an assignment of ERW is directed towards a file currently assigned for SRO.

ASGN-ERR TYPE=PROT

The file being assigned to is unconditionally protected (read and/or write keys=X'FF') or the read/write keys specified in the ASSIGN command do not correspond to those associated with the file, and the file is conditionally protected (read and/or write keys not X'00' or X'FF').

ASGN-ERR TYPE=SIZE

An indexed file is being assigned and there is not enough room on the disk to allocate a physical block.

ASGN-ERR TYPE=SPAC

An assign is refused because the available task system space was exceeded.

ASGN-ERR TYPE=TGD

An attempt was made to assign a trap generating device.

ASGN-ERR TYPE=VOL

Volume name specified or defaulted to is not the name of any of the disks currently online.

BTCH-ERR

The batch capability was not started and is not available for a SUBMIT command.

BUFF-ERR

The expanded CSS line overflowed CSS buffer size.

CLOS-ERR

Close failed for reason denoted by TYPE field.

DELE-ERR TYPE=ACCT

An attempt was made to delete a file not on the user's private account.

DEL-ERR TYPE=ASGN

An attempt is being made to delete a file that is currently assigned, or is being processed by the CSS processor.

DELE-ERR TYPE=BUFF

There is insufficient memory available in system space to perform a delete function.

DELE-ERR TYPE=DU

An attempt was made to delete a file from a device that is not on line.

DELE-ERR TYPE=IO

An I/O error was encountered while attempting to delete a file.

DELE-ERR TYPE=NAME

File with a specified name was not found.

DELE-ERR TYPE=PROT

An attempt is being made to delete a file with nonzero protection keys.

DELE-ERR TYPE=TYPE

The volume name specified or defaulted to is not a direct access device.

DELE-ERR TYPE=VOL

Nonexistent file is assigned to a task.

DUPLICATE USERNAME

Userid is already in use.

FD-ERR

The file descriptor is syntactically incorrect or invalid, or a program on the disk is being loaded without enough system space.

fd IS NOT A CONTIGUOUS FILE

The INIT command can only be used to initialize contiguous files.

FILE voln: filename. ext/acct NOT ENTERED ONTO PRINT QUEUE

A spool file was closed but the spooler task was not loaded or started.

FIXED POINT-ZERO DIVIDE ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

An attempt was made to divide by zero. Current instruction aborted, and next instruction at address xxxxxx.

FIXED POINT-OVERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Fixed point arithmetic result is too large to be represented. Instruction aborts. Next instruction at xxxxxx.

FLOATING POINT-UNDERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Results of floating point operation are too small to be represented. Instruction aborts. Next instruction at xxxxxx.

FLOATING POINT-OVERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Floating point arithmetic procedure is too large to be represented. Instruction aborts. Next instruction at xxxxxx.

FLOATING POINT-ZERO DIVIDE ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

An attempt was made to perform a floating point divide by zero.

FORM-ERR

The command format is invalid.

GOTO-ERR

A $LABEL that is terminating the range of the $GOTO is branching into an IF group.

| ILLEGAL INSTRUCTION AT XXXXXX

| The user task attempted to execute an illegal instruction at
| location XXXXXX.

| ILLEGAL SVC-INSTRUCTION AT XXXXXX
| SVC PARAMETER BLOCK AT XXXXXX

| The user task attempted to execute an illegal SVC at location
| XXXXXX.

| INVALID SEGMENT ADDRESS ERROR AT XXXXXX
| MEMORY FAULT ADDRESS=XXXXXX

| An attempt was made to access a memory location not within a
| valid mapped segment; i.e., an attempt to access a memory
| location outside of the task space.

| INVALID ACCOUNT

| Invalid or unrecognized account number.

| INVALID PASSWORD

| Password is invalid.

I/O-ERR

A device/file being accessed by MTM is returning a nonzero I/O status.

I/O-ERR TYPE=DU

The device is unavailable.

I/O-ERR TYPE=EOM I/O-ERR TYPE=EOF

The device reached an EOM or EOF before completing the operation.

I/O-ERR TYPE=FUNC

An invalid operation is being directed toward a device; e.g., attempting to write to a read-only device.

I/O-ERR TYPE=LU

An illegal or unassigned lu.

I/O-ERR TYPE=PRTY

A parity or other recoverable error occurred.

I/O-ERR TYPE=UNRV

An unrecoverable error occurred.

JOBS-ERR

A $JOB statement was encountered following another $JOB statement but prior to a $TERMJOB statement.

JOB NOT FOUND

The fd of job to be purged is invalid.

LOAD-ERR TYPE=ASGN

Load could not be accomplished because the specified fd is already exclusively assigned.

LOAD-ERR TYPE=DU

Attempt was made to load from an unavailable device.

LOAD-ERR TYPE=I/O

An I/O error was generated during the load operation.

LOAD-ERR TYPE=LIB

The data in the loader information block is invalid. This error most frequently occurs when an attempt is made to load a task which was not built with Link.

LOAD-ERR TYPE=LOPT

Task options are incompatible with the system environment that attempts to load the task; i.e., attempt to load an e-task under MTM where e-task loading under MTM is not enabled.

LOAD-ERR TYPE=MEM

A load was attempted without a large enough segment.

LOAD-ERR TYPE=MTCB

The maximum number of tasks specified at sysgen time was exceeded.

LOAD-ERR TYPE=NOFP

A task requiring floating point support is being loaded, and the required floating point option is not supported in the system.

LOAD-ERR TYPE=SEG

A task requiring a task common area (TCOM) and/or a run time library (RTL) is being loaded. The TCOM/RTL is not in the system and cannot be loaded.

LOAD-ERR TYPE=ROIO

There is an I/O error on the roll volume.

LOAD-ERR TYPE=RVOL

There is a roll file allocation or assignment error.

LU-ERR

An lu specified in an assign statement is invalid.

LVL-ERR

The number of sysgen CSS levels was exceeded.

| MEMORY ERROR ON DATA FETCH AT XXXXXX
| MEMORY FAULT ADDRESS=XXXXXX

| Attempt was made to retrieve or to load data from a failing memory area on Perkin-Elmer 3200 machines. If affected memory is within task space and the operating system has memory diagnostic support, the affected page is automatically marked off, and this message is displayed:

| AFFECTED MEMORY PAGE MARKED OFF AT XXXXXX

MEMORY ERROR ON INSTRUCTION FETCH AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

A Perkin-Elmer Model 3200 machine attempted to execute an installment from an area of memory that is failing. If affected memory is within task space and the operating system has memory diagnostic support, the affected page is automatically marked off, and this message is displayed:

AFFECTED MEMORY PAGE MARKED OFF AT XXXXXX

MEMORY PARITY ERROR AT XXXXXX

Attempt made to access nonexistent or bad memory on Model 7/32 and 8/32 machines.

MISSING PASSWORD

Password omitted.

MNEM-ERR

The command mnemonic entered is unrecognizable.

NOFP-ERR

No floating point support exists in the system.

NON EXISTENT SEGMENT ERROR (PST) AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a memory location greater than the maximum valid program address; i.e., an attempt to access a memory location outside of the task space.

NOPR-ERR

A command was entered that required more parameters than specified in the command line.

PACKED FORMAT-SIGN ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An illegal sign digit was detected in a packed decimal number at xxxxxx for Perkin-Elmer 3200 Model 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 Model 3200 machines only.

PARM-ERR

A command was entered with invalid or missing parameters.

PRIV-ERR

The access privilege mnemonic is syntactically incorrect, or an MTM user without access privileges tried to access a restricted file.

RENM-ERR TYPE=NAME

A filename already exists in the volume directory.

RENM-ERR TYPE=PRIV

The file/device cannot be assigned for ERW (required to perform the rename) because the file/device is currently assigned to at least one lu.

The protection keys of the file to be renamed are not X'0000'.

REPR-ERR TYPE=PRIV

The file/device cannot be assigned for ERW (required to carry out the reprotection) because the file/device is currently assigned to at least one lu.

ROLL-ERR

The task is currently rolled out.

SEGMENT LIMIT ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a memory location within a valid mapped segment, but the page number in the segment is greater than the largest valid page number for the segment; i.e., an attempt to access a memory location outside of the task space.

SEQ-ERR

A command was entered out of sequence. Task was not loaded when a BUILD command was entered.

SIGNON REQUIRED

Attempt to enter a command before signon or a mistake in the SIGNON command.

SKIP-ERR

An attempt was made to skip beyond the end of a CSS job.

SPAC-ERR

Task exceeds established maximum system space.

SVC ADDRESS ERROR-INSTRUCTION AT XXXXXX
SVC PARAMETER BLOCK AT XXXXXX

Incorrect address of SVC parameter block at xxxxxx. The SVC parameter block must be on a fullword boundary.

SVC6-ERR TYPE=ARGS

There is insufficient room between UTOP and CTOP to contain the start option string.

SVC6-ERR TYPE=DORM

A command was issued to a specified task that is dormant.

SVC6-ERR TYPE=NMSG

The task currently executing at the terminal could not receive a message trap.

SVC6-ERR TYPE=PRES

The Spooler was not loaded when it was requested.

SVC6-ERR TYPE=QUE

Spooler is dormant.

TASK-ERR

A task-related command was entered and there is no currently loaded task.

TIME-ERR

A task cannot be loaded because the user account CPU limit expired.

UNDEFINED DATA FORMAT FAULT AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An undefined data format/alignment fault was detected at xxxxxx for Perkin-Elmer 3200 Series machines.

USER-ERR

An invalid userid was entered in a MESSAGE command.

VOLN-ERR

The volume specified is not online or the volume name is invalid.

xxxx ERROR ON fd SECTOR n

An I/O error occurred while attempting to initialize sector n of file fd. xxxx is the type of error; it may be unrecoverable I/O, recoverable I/O, or device unavailable.

APPENDIX E
CSS MESSAGE SUMMARY

BUFF-ERR

indicates an expanded command line exceeds the CSS buffer. The task skips to $TERMJOB.

FD-ERR

indicates not enough space to build an fd, or required file support is not in system. The task skips to $TERMJOB.

FORM-ERR

indicates a command syntax is invalid. The task skips to $TERMJOB.

GOTO-ERR

indicates a $LABEL occurred inside an IF block that was not active at the time of the $GOTO command. The task skips to $TERMJOB.

I/O-ERR

indicates an EOF was found while skipping to $ENDC, an EOF was found before a $ENDB while building a file, or a $TERMJOB was found while skipping to $ENDC within a job. The task skips to $TERMJOB, or end of task code is 255 and job is ended.

JOBS-ERR

indicates a second $JOB was found.

LVL-ERR

indicates the CSS levels required exceed the number established at sysgen time.

MNEM-ERR

indicates the command entered is not recognized. The task skips to $TERMJOB.

PARM-ERR

indicates a command was entered with invalid or missing parameters.

SEQ-ERR

indicates a command was entered out of sequence.

TASK-ERR

indicates a task-related command was entered and there is no currently loaded task. The task skips to $TERMJOB.

@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 TOC LONG

indicates that the length of the defined variable is greater than 32.

@XXXX-VARIABLE ERROR, DOES NOT EXIST

indicates an attempt to set or free the value of a nonexistent variable. Also, during CSS execution, a variable definition is required.

@XXXX-VARIABLE ERROR, DEFINITION DOES NOT EXIST

indicates an attempt to set the value of a variable to the value of a second nonexistent variable.

@SYSCODE-VARIABLE ERROR, UNABLE TO ACCESS PAGE-FILE

indicates that at signon time MTM was unable to access the variable page file.

VARIABLE ERROR, VARIABLE PROCESSING NOT SUPPORTED

indicates that one of the following variable related commands was entered into a system that does not support variable processing:

- $FREE
- $GLOBAL
- $LOCAL
- $SET

VARIABLE ERROR, VARIABLE PROCESSING DISABLED

indicates that one of the following variable related commands was entered into a system with variable processing support that is disabled:

- $FREE
- $GLOBAL
- $LOCAL
- $SET

APPENDIX F
PROGRAM DEVELOPMENT MESSAGE SUMMARY

** ALTERNATE CSS REQUIRED

The fd entered with the ADD command contains a nonstandard extension, and the cssprod parameter was not specified.

** COMPILE ERROR - LINK NOT EXECUTED

In a complink process, a compilation error was found, and the process aborted before the link procedure began.

** COMPILE ERRORS, LISTING ON PR:

Errors were encountered while compiling. These errors are listed on the specified pr:.

** CURRENT ENVIRONMENT - filename

The ENV command, entered without a filename, causes the name of the current environment to be displayed.

** CURRENT PROGRAM NOT SET

A filename was not specified, or no current program exists.

** EDIT - filename.ext

In the multi-module environment, the EDIT command was entered without a filename. The fd of the current source program is displayed.

** ENVIRONMENT EMPTY

The LIST command was entered, but there are no file descriptors in the EDF.

** EXTENSION OMITTED

A filename entered with the ADD or REMOVE command did not contain the required extension.

** EXECUTION OF filename FOLLOWS:

An image program is loaded and is executing.

** FILE fd NOT FOUND

The specified filename cannot be found in the language environment.

** fd NONEXISTENT

A specified fd does not exist in the environment.

** FILENAME CONFLICT - ENTRY NOT ADDED

An attempt was made to add an already existing fd to the EDF.

** FILENAME NOT IN ENVIRONMENT

An fd specified with the REMOVE command does not exist in the EDF.

** LANGUAGE ENVIRONMENT NOT SET

A development command such as EDIT, COMPILE, COMPLINK, or EXEC was entered without first setting the language environment.

** LINK ERRORS - EXECUTION ABORTED

Program execution aborted when a link error was encountered

** NEW ENVIRONMENT

An empty EDF has been allocated.

** NEW PROGRAM

An empty source file is allocated in the language environment.

** NO CURRENT EDF

The ENV command was entered without an EDF name, or there is no current EDF.

** NON-STANDARD EXTENSION

An attempt was made to add an fd with a nonstandard language extension to the EDF without specifying a cssprod parameter.

** NOT IN MULTI-MODULE ENVIRONMENT

A command that is only meaningful in a multi-module environment was specified in a language environment.

** SOURCE FILE NOT FOUND

The specified source file cannot be found.

** SYNTAX ERRCR

An fd was not specified with the ADD or REMOVE command.

** TASK fd NOT FOUND

The specified task cannct not be found.

** TOO MANY ARGUMENTS

Arguments were specified in a multi-module environment.