PERKIN ELMER

OS/32 MULTI-TERMINAL MONITOR (MTM)

Reference Manual

48-043 R00

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

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

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

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

TABLE OF CONTENTS

CHAPTERS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES

TABLES

PREFACE

This manual contains information on the Perkin-Elmer Multi-Terminal Monitor (MTM). It is written for the MTM user but could be helpful to the system operator and system programmer.

Chapter 1, which is reorganized, is a general description of the MTM system containing general information on MTM system requirements, MTM features, and various conventions. Chapter 2 describes MTM user commands, and Chapter 3 contains program development commands. Chapter 4 describes batch processing under MTM. Chapter 5 describes the command substitution system (CSS) and includes all CSS commands. Chapter 6 describes spooling.

Appendix A summarizes the MTM user commands. Appendix B summarizes the program development commands. Appendix C summarizes the CSS commands and Appendix D summarizes the terminal user command messages. 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 a chapter describing the new program development commands. The signon CSS, USERINIT.CSS, is made more flexible. Vertical forms control is added, and various changes are made to several MTM user commands. A Help facility enables a user to access information on how to use 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. Alsc, 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 R06.0 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
32-Bit Systems Software User Documentation Summary	48-015
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

For further information on the contents of all Perkin-Elmer 32-bit software manuals, see the 32-Bit Systems Software 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, the application of MTM becomes limitless, supporting a mixture of terminal users such as clerks, 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 the functions performed when MTM is operating; i.e., dynamic functions.

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 functicns would be performed and maintained by the customer's system support group responsible for making computing facilities available to system users. The dynamic functions would be perfcrmed by a system operator during system operation. The functions performed by a system operator 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 as well as 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 situated either locally or remotely. The interactive terminals for local users are directly connected to the computer without requiring 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) 550
- VDU 1100
- VDU 1200
- VDU 1250
- Perkin-Elmer SIGMA 10 terminal
- M33 Teletype
- M35 Teletype
- Non-editing 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 contrcls 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 an interactive 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 currently loaded, CSS procedure is not currently executing and BUILD is not in effect. All nontask-related commands are accepted. An "*" prompt is used in this mode.
- Task Loaded Mode: The task was loaded but was not started, or is paused. An "*" prompt is used in this mode.
- Task Executing Mode: A task was started and is executing. If started from CSS, CSS mode is suspended. A "-" prompt is used 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 currently being built or executed. A "-" prompt is used in this mode. When CSS terminates, the terminal returns to command mode and a "*" prompt is output. When a CSS procedure is currently being built, a "B>" prompt is displayed.

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 discs
- Signon CSS

1.6.1 Command Substitution System (CSS)

A terminal user can build a command stream on a disc file. 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. They 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 their currentness.

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. The Spooler can be used to submit a batch job stream to MTM. The job then runs unattended and output is directed toward the Spooler.

1.6.5 Security and Access Protection of Discs

Privately owned discs 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 disc can restrict or enable access of the disc 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 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 discs 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 used on the terminal devices are shown in Table 1-1.

TABLE 1-1 PROMPT CONVENTIONS

PROMPT	USE
*	Indicates MTM system is ready to accept another command.
>	Indicates a request for input data.
B>	Requests 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 interactively 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 CRT which uses the number sign (#). Basic communications support both # and CTRL X for line deletion for asychronous 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.

1.7.2.1 Using the Break Key

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

If an 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 command entered 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 when entering multiple commands on the same line and the command line 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 line of data preceded by a command beginning with an asterisk is considered to be a comment.

1.7.3.1 Statement Syntax Conventions

Following is a list of syntax conventions used in this manual. An example of each syntax convention follows the description.

Capital letters must be entered exactly as shown.

$\underline{\text{SIGNOF}}\text{F}$

Lowercase letters represent parameters or denote information provided by the user.

n

Underlining points out the mnemonic of the entry and means the underlined portion must be entered.

$\underline{\text{P}}\text{AUSE}$

An ellipsis represents an indefinite number of parameters or a range of parameters.

$$\underline{\text{S}}\text{IGNON userid }[\text{,actno,password}]\left[\text{,}\underline{\text{ENV}}\text{IRONMENT=}\begin{Bmatrix}\text{fd}\\ \text{NULL }[:]\end{Bmatrix}\right]$$

$$[\text{,}\underline{\text{CPUTI}}\text{ME=maxtime}]$$

$$[\text{,classid=iocount}_1\ [\text{,...,classid=iocount}_{32}]]$$

Shading represents a default option.

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

Braces represent required parameters from which one must be chosen.

$$\underline{ME}\text{SSAGE }\left\{\begin{matrix}\text{userid}\\ \underline{\text{.O}}\text{PERATOR}\end{matrix}\right\}\text{ message}$$

Brackets represent an optional parameter.

$$\underline{LO}\text{G }\left[\text{fd}\left[,\left\{\begin{matrix}\underline{N}\text{OCCPY}\\ \underline{C}\text{OPY}\end{matrix}\right\}\right]\right]\left[,\left\{\begin{matrix}\text{n}\\ \text{15}\end{matrix}\right\}\right]$$

Commas separate parameters and substitute missing positional parameters.

$$\begin{aligned}&\underline{S}\text{IGNON userid }[\text{,actno,password}]\left[,\underline{ENV}\text{IRONMENT=}\left\{\begin{matrix}\text{fd}\\ \text{NULL }[:]\end{matrix}\right\}\right]\\ &\qquad[,\underline{CPUTI}\text{ME=maxtime}]\\ &\qquad[\text{,classid=iocount}_1\ [,\ldots,\text{classid=iocount}_{32}]]\end{aligned}$$

Braces inside brackets represent optional parameters from which one can be chosen.

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

Commas preceding braces inside brackets must be entered if one of the optional parameters is chosen.

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

A comma inside brackets must be entered if the optional parameter is chosen.

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

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

An equal sign associates a parameter with its keyword.

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

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 value of the group acccunt number. The grcup 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 nc 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:

- Disc volume name or device name
- Filename
- File extension
- Account number

The format of the file descriptor is:

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

Parameters:

voln: is the name of the disc 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. Voln need not be specified. If voln 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 does 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.
SA 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:

$$\text{\underline{AL}LOCATE fd,}\left\{\begin{array}{l} \text{\underline{CO}NTIGUOUS,fsize}\left[,\left\{\begin{array}{c}\text{keys}\\ \text{0000}\end{array}\right\}\right] \\ \text{\underline{I}NDEX}\left[,\left[\left\{\begin{array}{c}\text{lrecl}\\ 126\end{array}\right\}\right]\right]\left[/\left[\left\{\begin{array}{c}\text{bsize}\\ 1\end{array}\right\}\right]\right]\left[/\left[\left\{\begin{array}{c}\text{isize}\\ 1\end{array}\right\}\right]\right] \\ \qquad\left[,\left\{\begin{array}{c}\text{keys}\\ \text{0000}\end{array}\right\}\right] \\ \text{\underline{I}TAM}\left[,\left[\left\{\begin{array}{c}\text{lrecl}\\ 80\end{array}\right\}\right]\right]\left[/\left[\left\{\begin{array}{c}\text{bsize}\\ 1\end{array}\right\}\right]\right]\left[\left\{\begin{array}{c}\text{keys}\\ \text{0000}\end{array}\right\}\right] \end{array}\right\}$$

Parameters:

fd — is the file descriptor of the device or file to be allocated.

CONTIGUOUS — specifies 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 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.

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 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,CC,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 V01:AJM.OBJ,IN,126//3 — Allocates, on volume V01, 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.

Messages:

ALLO-ERR	Allocation failed for reasons denoted by TYPE field.
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
NOPR-ERR	Required operand missing
PARM-ERR	Invalid parameter

2.3 ASSIGN COMMAND

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

Format:

```
                 ┌ ┌ ⎧access privileges⎫ ┐                         ┐
                 │ │ ⎪                 ⎪ │                ┌ ⎧SVC15⎫ ┐│
                 │ │ ⎪       SRW       ⎪ │  ┌   ⎧keys⎫ ┐  │ ⎪     ⎪ ││
    ASSIGN lu,fd │,│ ⎨                 ⎬ │  │ , ⎨    ⎬ │  │,⎨SVCF ⎬ ││
                 │ │ ⎪       SREW      ⎪ │  └   ⎩0000⎭ ┘  │ ⎪     ⎪ ││
                 │ │ ⎪                 ⎪ │                └ ⎩ VFC ⎭ ┘│
                 └ └ ⎩       SRO       ⎭ ┘                         ┘
```

NOTE

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

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 if desired.

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 — is the desired access privileges. The default access privileges are:

- SRW for contiguous files and devices
- SREW for indexed files
- SRO for files within the group or system account

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.

Functional Details:

Access privileges can be one of the following:

SRC	sharable read-only
ERO	exclusive read-cnly
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

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 has assigned 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 form of protection is called dynamic 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. For example, 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	SRC	SRW	SWO	EWO	SREW	ERW
ERSW	-	-	-	-	*	-	-	-
ERO	-	-	-	-	*	*	-	-
SRO	-	-	*	*	*	*	*	-
SRW	-	-	*	*	*	-	-	-
SWO	*	*	*	*	*	-	-	-
EWO	-	*	*	-	-	-	-	-
SREW	-	-	*	-	-	-	-	-
ERW	-	-	-	-	-	-	-	-

*	compatible
-	incompatible

The keys format is a 4-digit hexadecimal number. The left two digits signify the write prctection 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 protecticn 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, conditicnally 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 may be entered in task loaded mode.

Examples:

`AS 2,FILE.DAT,EWO,99AA`

Assigns a disc 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 disc 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 disc 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,,OOFF,VFC — Assigns a disc 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 disc file to lu 2. Vertical forms control is in use. The keys parameter, along with the positional comma, is omitted.

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.

Messages:

ASGN-ERR	The assign failed for reason denoted by the TYPE field.
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error
PRIV-ERR	Invalid access privilege mnemonic
TASK-ERR	No task loaded

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 may 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.

Messages:

ASGN-ERR — The file or device could not be assigned for the reason noted in the TYPE field.

FORM-ERR — Command syntax error.

I/O-ERR — An I/O error was encountered.

LU-ERR	An invalid file descriptor was encountered.
MNEM-ERR	Incorrect command name
TASK-ERR	No task loaded.

2.5 BIAS COMMAND

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

Format:

$$\underline{\text{BI}}\text{AS} \begin{Bmatrix} \text{address} \\ * \end{Bmatrix}$$

Parameter:

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 0.

* is the starting address of the currently loaded 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
```

Messages:

FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error
TASK-ERR	No task loaded

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 may be entered in task executing mode.

Messages:

MNEM-ERR	Incorrect command name
SEQ-ERR	Break was entered when no user task I/O was pending to the MTM terminal.

2.7 BRECORD COMMAND

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

Format:

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

Parameters:

fd — is the file descriptor of the device or file to be backspaced one record.

lu — is the lu to which the file is assigned. If lu is specified without fd, the operation is performed on the lu regardless of what is assigned tc it.

Functional Details:

The BRFCORD ccmmand may 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.

Messages:

ASGN-ERR — The file or device could not be assigned for the reascn noted in the TYPE field.

FORM-ERR — Command syntax error.

I/O-ERR — An I/O error was encountered.

LU-ERR	An invalid file descriptor was encountered.
MNEM-ERR	Incorrect command name.
TASK-ERR	No task loaded.

2.8 BUILD AND ENDB COMMANDS

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

Format:

```
       (fd)
BUILD  {  } [,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 ccpied to it. When the ENDB is encountered, the temporary file is assigned to the specified lu of the loaded task. This form cf 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 following BUILD are not treated as commands, but as data, and are copied to the specified device or file until an ENDB is encountered. The ENDB command must be the first command in the ccmmand line, but it need not start in column 1. ENDB must be followed by other commands in the command line. The BUILD command must be the last command on an input line. Further data appearing on that line is treated as comment and causes no action to be taken. The BUILD command may be entered from the terminal only if CSS is not active. It may be entered in command mode, task loaded mode, and task executing mode.

Example:

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

Messages:

ASGN-ERR	Output file or device could not be assigned.
FD-ERR	Invalid file descriptor or no indexed file support
FORM-ERR	Command syntax error
LU-ERR	The lu specified was already assigned (only valid when lu operand is specified).
MNEM-ERR	Incorrect command name
PARM-ERR	Invalid parameter
SEQ-ERR	CSS file active, when BUILD was entered from user's terminal, or BUILD command entered with lu operand and task was not dormant.
TASK-ERR	No task loaded

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 may be entered in task loaded mode and task executing mode.

Messages:

FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
TASK-ERR	No task loaded

2.10 CLOSE COMMAND

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

Format:

$$\underline{\text{CL}}\text{OSE}\begin{Bmatrix} lu_1\,[,lu_2,\ldots,lu_n] \\ \underline{\text{A}}\text{LL}\end{Bmatrix}$$

Parameters:

lu — decimal numbers signifying the logical units to be unassigned.

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 may 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.

Messages:

CLOS-ERR	Close failed for reason denoted by TYPE field
FORM-ERR	Command syntax error

MNEM-ERR	Incorrect command name
NOPR-ERR	Required argument missing
PARM-ERR	Invalid cr missing parameter
SEQ-ERR	Task nct dormant or paused
TASK-ERR	No task loaded

2.11 CONTINUE COMMAND

The CONTINUE command causes a task that has been paused 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 may be entered in task loaded mode. Executing this command causes the terminal mode to be switched from task loaded mode to task executing mode.

Messages:

FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
SEQ-ERR	Task not paused
TASK-ERR	No task loaded

2.12 DELETE COMMAND

The DELETE command deletes a direct access file.

Format:

$\underline{DE}$LETE $fd_1 [,fd_2,...,fd_n]$

Parameter:

fd	identifies the file(s) to be deleted.

Functional Details:

The file being deleted must not be currently assigned to any lu of any task. A file can only be deleted 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 may be deleted.

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

Messages:

DELE-ERR TYPE=	Delete failed for reason denoted by TYPE field
NAME	File with the specified name was not found
PROT	Protecticn keys for the specified file are other than 0
PRIV	Specified file is assigned to a task
VOL	Nonexistent file is assigned to a task
TYPE	Incorrect device type was specified
BUFF	Insufficient memory available in system space to delete an indexed file
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name

2.13 DISPLAY ACCOUNTING COMMAND

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

Format:

```
                         ┌ ⎧      fd      ⎫ ┐
    DISPLAY ACCOUNTING   │,⎨              ⎬ │
                         └ ⎩ user console ⎭ ┘
```

Parameter:

fd is the file descriptor to which the accounting information is displayed.

Functional Details:

The DISPLAY ACCOUNTING command displays this information:

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

The DISPLAY ACCOUNTING command may 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.

Messages:

ASGN-ERR	Output device could not be assigned
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O error encountered on output device/file

MNEM-ERR Incorrect command name

PARM-ERR Invalid parameter

SEQ-ERR No task has yet been run during the current terminal session.

```
 ------------
| DISPLAY    |
| DEVICES    |
 ------------
```

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 display is output to the terminal.

Functional Details:

The DISPLAY DEVICES command may 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) 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 discs, column 5 contains the characters PROT. For MTM users, if the disc is write-protected, column 5 contains the characters SYS. If the disc is restricted, column 5 contains the characters RES. If the secondary directory option is enabled, the last column contains the characters CDIR.

Messages:

ASGN-ERR	Optional fd could not be assigned
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O error encountered on output device or file
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error

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 display is output to the terminal.

Functional Details:

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

The DISPLAY DFLOAT command may 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
```

Messages:

ASGN-ERR	Optional fd could not be assigned; e.g., fd is already assigned for exclusive use
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O errcr encountered on output file or device.
MNEM-ERR	Incorrect command name
NOFP-ERR	Task not built with the DFLOAT option at Link time
PARM-ERR	Operand syntax error
TASK-ERR	No task loaded

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:

$$\text{\underline{D}ISPLAY \underline{F}ILES}\ \left[,\left[\begin{Bmatrix}\text{voln:}\\ \text{default user vol}\end{Bmatrix}\right]\right][\text{filename}]\ \ [.\ [\text{ext}]]$$

$$\left[/\left[\begin{Bmatrix}\text{P}\\ \text{S}\\ \text{G}\end{Bmatrix}\right]\ \left[,\begin{Bmatrix}\text{fd}\\ \text{user console}\end{Bmatrix}\right]\right]$$

Parameters:

voln: is a 1- to 4-character name of a disc volume. The first character must be alphabetic, the remaining alphanumeric. If voln is omitted, the default user volume is assumed.

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 display is output to the user terminal.

Functional Details:

A hyphen (-) in the command format requests that all files starting with the characters preceding 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.

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.

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-/-


VCLUME= V67B
FILENAME  EXT TYPE LENGTH KEYS START/NLR CREATED WRITTEN ACT
TAS       CSS  IN      80 **00         1 6/09/79 6/09/79  18
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 disc 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 disc volume in the system.

Messages:

ASGN-ERR	Optional fd could not be assigned
FORM-ERR	Command syntax error
I/O-ERR	I/O error encountered on output device or file
MNEM-ERR	Incorrect command name
NOPR-ERR	Required operand missing on output device or file
PARM-ERR	Operand syntax error

2.17 DISPLAY FLOAT COMMAND

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

Format:

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

Parameter:

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

Functional Details:

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

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

Example:

```
D FL
0,2  00000000   00000000
4,6  00000000   00000000
8,A  00000000   0000000C
C,E  00000000   0000000C
```

Messages:

ASGN-ERR — Optional fd could not be assigned; e.g., fd is already assigned for exclusive use

FD-ERR — Invalid file descriptor

FORM-ERR	Command syntax error
I/O-ERR	I/O error encountered on output file or device
MNEM-ERR	Incorrect command name
NOFP-ERR	Specified task not established with the FLOAT option at Link time
PARM-ERR	Operand syntax error
TASK-ERR	No task loaded

```
 ------------
|  DISPLAY   |
|    LU      |
 ------------
```

2.18 DISPLAY LU COMMAND

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

Format:

```
                   ┌ ⎧       fd        ⎫ ┐
    DISPLAY LU     │,⎨                 ⎬ │
    -       -      └ ⎩  user conscle   ⎭ ┘
```

Parameter:

fd — is an cptional file descriptor specifying the file or device to which the assigned logical units are to be displayed. If fd is omitted, the display is output to the user's terminal.

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   CCN:,SRW
     6   CON:,SRW


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

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

Example:

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

Messages:

ASGN-ERR	Optional fd could not be assigned; e.g., fd is already assigned for exclusive use
FD-ERR	Invalid optional file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O error detected on output device or file
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error
TASK-ERR	No task loaded

2.19 DISPLAY PARAMETERS COMMAND

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

Format:

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

Parameter:

fd — is an cptional file descriptor specifying the file or device to which the display is output. If fd is omitted, the display appears at the user ccnsole.

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

FIELC	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
MUSP	xxxxx	Maximum used system space
MXSP	xxxxx	Maximum allowed system space
CTCP	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	Cefault volume id

The addresses displayed as CTOP, UTOP, UBOT, and SLOC are not physical addresses, but are addresses within the task's own program space. CLOC may te a program space address or a physical address in a system subroutine being executed on behalf of the task. NLU is given in decimal. SVOL is the ASCII system volume id. The fields CTOP, UTCP, UBCT, 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
5	0400 0000	0 - Prompt disabled 1 - Prompt enabled
6	0200 0000	0 - I/O interpreted without vertical forms control 1 - All I/O interpreted with vertical forms control
7	0100 0000	0 - No extended SVC 1 parameter blocks used (excludes communications I/O) 1 - Extended SVC 1 parameter blocks used
8	0800 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

TABLE 2-3 TASK OPTION BIT DEFINITIONS (Continued)

BIT	MASK	MEANING
24	0000 0080	0 - NCRCLL 1 - RCLL
25	0000 0040	0 - Nc overlay 1 - Use overlay
26	0000 0020	0 - Accounting disabled 1 - Accounting enabled
27	0000 0010	0 - Task can issue intercept call 1 - Task cannot issue intercept call
28	0000 0008	0 - No account privileges 1 - File account privileges
29	0000 0004	0 - Bare disc assign not allowed 1 - Bare disc assign allowed
30	0000 0002	0 - Nct universal 1 - Universal
31	0000 0001	0 - Nc keychecks 1 - Dc keychecks (e-tasks only)

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	Ccnsole wait (paused)
19	0000 1000	Lcad wait
20	0000 0800	Dormant

TABLE 2-4 WAIT STATUS BIT DEFINITIONS (Continued)

BIT	MASK	MEANING
21	0000 0400	Trap wait
22	0000 0200	Time of day wait
23	0000 0100	Suspended
24	0000 0080	Interval wait
25	0000 0040	Terminal wait
26	0000 0020	Roll pending wait
27	0000 0010	Intercept initialization (MTM)
28	0000 0008	Intercept termination (MTM)
29	0000 0004	System resource connection wait
30	0000 0002	Accounting wait

NOTE

Zero status indicates the task is active.

CTSW is given in hexadecimal. For a definition of the status portion of the TSW, see the OS/32 Applications 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:

```
D P
TASK      MTMUSER
CTSW     00001000
PSW         477F0
CLOC       F2B7C
STAT         2000
TOPT        10021
USSP         14F8
MUSP         2208
MXSP         3000
CTOP         24FE
UTOP         2370
UBOT            0
SLOC        F0000
NLU            15
MPRI          128
SVOL         M67A
```

Messages:

ASGN-ERR	Optional fd could not be assigned
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O error detected on output device or file
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error
TASK-ERR	No task loaded

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:

```
                       ┌ ┌     fd       ┐ ┐
    DISPLAY REGISTERS  │,┤              ├ │
                       └ └ 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 tc the user console.

Functional Details:

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

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
```

Messages:

ASGN-ERR — Optional fd could not be assigned; e.g., fd is already assigned for exclusive use

FD-ERR — Invalid file descriptor

FORM-ERR	Command syntax error
I/O-ERR	I/O error detected on output device or file
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error
TASK-ERR	No task loaded

NOTE

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

2.21 DISPLAY TIME COMMAND

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

Format:

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

Parameter:

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

Functional Details:

The display has the following format:

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

or alternatively (by sysgen option):

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

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

Messages:

ASGN-ERR	Optional fd could not be assigned
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error

I/O-ERR	I/C error encountered on output device or file
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error

2.22 DISPLAY USERS COMMAND

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

Format:

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

Parameter:

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

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

Example:

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

Messages:

ASGN-ERR	Output device could not be assigned
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O error encountered on output device or file
MNEM-ERR	Incorrect command name
PARM-ERR	Invalid parameter

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:

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

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.

$VARIABLE enables variable processing on a per user basis.

Functional Details:

The ENABLE command does not affect operator messages.

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

Messages:

FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error

2.24 EXAMINE COMMAND

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

Format:

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

Parameters:

address indicates the starting and ending addresses in memory whose contents are to be displayed in hexadecimal. All addresses specified are rounded down to halfword boundaries by the system.

n is a decimal number specifying the number of halfwords to be displayed. If n is omitted, one halfword is displayed.

fd is the file descriptor specifying the file or device to which the contents of memory are displayed. If omitted, the display is output to the user's console.

Functional Details:

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

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

Any memory that may be accessed by the loaded task may 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 B000 EXA 100,10	Examines 10 halfwords starting at relative address 100 (absolute address B100) within the task.

Messages:

FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
I/O-ERR	I/O error detected on output device or file
MNEM-ERR	Incorrect command name
NCPR-ERR	Required cperand missing
PARM-ERR	Operand syntax error (an attempt to examine memory reserved for MAC, or marked off and possibly nonexistent)
ROLI-ERR	Task currently rolled out
TASK-ERR	No task lcaded

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 may 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 lu4, to forward space one filemark.

Messages:

ASGN-ERR — The file or device could not be assigned for the reason noted in the TYPE field

FORM-ERR — Command syntax error

I/O-ERR — I/C error encountered on the specified device or file

LU-ERR — Logical unit not a legal decimal number or greater than maximum lu for the task

MNEM-ERR — Incorrect command name

TASK-ERR — No task loaded

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.
```

Messages:

ASGN-ERR — The file or device could not be assigned for the reason noted in the TYPE field.

FORM-ERR — Command syntax error

I/C-ERR — I/O error encountered on the specified device or file.

LU-ERR Logical unit not a legal decimal number or greater than the maximum lu specified for the task.

MNEM-ERR All commands on the same line following an erronecus command are ignored.

TASK-ERR There was no task loaded.

2.27 HELP Command

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

Format:

$$\text{HELP}\begin{Bmatrix}\text{mnemonic}\\ *\end{Bmatrix}$$

Parameters:

mnemonic — is any valid MTM mnemonic or program development command.

* — causes a list of all MTM and program development commands to be displayed to the list device.

Functional Details:

The HELP command is inplemented 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.

2.28 INIT COMMAND

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

Format:

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

Parameters:

fd — is the file descriptor of any unassigned, unprotected, contiguous file.

segsize increment — is the size of the buffer space used. The default is 1kb.

Functional Details:

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

The INIT command can be entered in command mode.

Examples:

```
INIT DATA.FIL            Initializes the file DATA.FIL.

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

Messages:

ASGN-ERR — An error occurred when an attempt was made to assign the file to be initialized or the task file for the File Manager Support Utility. See the ASSIGN command description for error information.

FD-ERR
Invalid file descriptor was specified

fd IS NOT A CONTIGUOUS FILE
INIT can only be used to initialize contiguous files.

FORM-ERR
Invalid segment size increment was specified

LOAD-ERR
An error occurred when an attempt was made to load the File Manager Support Utility. See the LOAD command for error information on loading a task.

MNEM-ERR
An incorrect command name was specified.

SEQ-ERR
Another CSS procedure is active. A second INIT command cannot be executed until the first has completed.

xxxx ERRCR 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/C, or device unavailable.

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) 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.

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.

Messages:

FD-ERR	File descriptor syntax error
FORM-ERR	Command syntax error
LOAD-ERR	Load failed for reason noted in TYPE field
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error
SEQ-ERR	Task already loaded

2.30 LOG COMMAND

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

Format:

$$\text{LOG}\left[\text{fd}\left[,\begin{Bmatrix}\underline{\text{N}}\text{OCOPY}\\ \underline{\text{C}}\text{OPY}\end{Bmatrix}\right]\right]\left[,\begin{Bmatrix}\text{n}\\ 15\end{Bmatrix}\right]$$

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:

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

Example:

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

Messages:

ASGN-ERR	Log file could not be assigned
FD-ERR	File descriptor syntax error
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name

2.31 MESSAGE COMMAND

The MESSAGE command sends a message to a specified user.

Format:

```
         (userid   )
MESSAGE <          > message
         (.OPERATOR)
```

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
```

Message:

USER-ERR	Userid not currently signed on

2.32 MODIFY COMMAND

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

Format:

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

Parameters:

address — is the halfword boundary address at which the contents of memory are to be modified.

data — is a data field consisting of zero to four hexadecimal digits that represent a halfword to be written into memory starting at the locaticn 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 (mcdified 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 u-task has write access can be modified. Only the impure segment can be modified for an e-task.

Examples:

BI 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.

Messages:

ADRS-ERR	Address nct halfword boundary aligned or not in a writable segment of the loaded task
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
NCPR-ERR	Missing cperand
ROLL-ERR	Task currently rolled out
TASK-ERR	No task loaded

2.33 OPTIONS COMMAND

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

Format:

$$\underline{\text{C}}\text{PTICNS}\left[\begin{Bmatrix}\underline{\text{AFP}}\text{AUSE}\\ \underline{\text{AFC}}\text{ONTINUE}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\underline{\text{SVCP}}\text{AUSE}\\ \underline{\text{SVCC}}\text{ONTINUE}\end{Bmatrix}\right]\left[,\underline{\text{NON}}\text{RESIDENT}\right]$$

Parameters:

AFFAUSE 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 tc 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 CPTICNS command can be entered in task loaded mode.

Example:

```
CPT   AFC,SVCC
```

Messages:

FCRM-ERR	Command syntax error
PARM-ERR	Operand syntax error
TASK-ERR	No currently selected task

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 may be entered in task loaded mode and task executing mode.

Messages:

FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
SEQ-ERR	Task paused
SVC6-ERR POS=DORM	Task dormant
TASK-ERR	No task lcaded

2.35 PREVENT COMMAND

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

Format:

$$\text{\underline{PRE}VENT}\left\{\begin{array}{l}\text{\underline{M}ESSAGE}\\ \text{\underline{PR}OMPT}\\ \text{\underline{ET}M}\\ \text{\$\underline{VAR}IABLE}\end{array}\right\}$$

If a terminal user has not input either of these commands then 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.

$VARIABLE disables variable processing on a per user basis.

Functional Details:

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

Messages:

FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error

2.36 PRINT COMMAND

The PRINT command sends the data to be printed to a pseudo device (spool print queue) to be subsequently sent to the device that will actually print the file.

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. 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.

Functional Details:

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

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

Messages:

FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error

MNEM-ERR	Incorrect command name
SVC 6-ERR TYPE=NMSG	Spooler did not receive message
SVC 6-ERR TYPE=PRES	Spooler not loaded
SVC 6-ERR TYPE=QUE	Spooler is dormant

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. 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.

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.

Messages:

FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error

MNEM-ERR	Incorrect command name
SVC 6-ERR TYPE=NMSG	Spooler did not receive message
SVC 6-ERR TYPE=PRES	Spooler not loaded
SVC 6-ERR TYPE=QUE	Spooler is dormant

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 new file descriptor to which the file is renamed.

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 may 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.
```

Messages:

ASGN-ERR
TYPE=PRIV
PCS=fd
	File descriptor currently assigned or the RENAME directed to the system console

FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error

MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error

RENM-ERR
TYPE=NAME
POS=fd
	File renamed to existing file

2.39 REPROTECT COMMAND

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

Format:

```
    REPRCTECT 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.

Messages:

ASGN-ERR	Reprotect failed for reason noted in TYPE field
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
NOPR-ERR	Keys nct specified
PARM-ERR	Operand syntax error
REPR-ERR	Reprotect failed for reason noted in TYPE field

ASGN and REPR error TYPE field responses:

NAME	File descriptor does not exist on specified volume
PRIV	File descriptor currently assigned
VOL	Volume or device does not exist

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.

Messages:

ASGN-ERR	File or device could not be assigned for the reason ncted in the TYPE field
FORM-ERR	Command syntax error encountered
I/O-ERR	I/O error encountered on the specified device or file
LU-ERR	Invalid file descriptor encountered
MNEM-ERR	Incorrect command name
TASK-ERR	No task loaded

ASGN error TYPE field responses:

NAME	File descriptor does not exist on specified volume
PRIV	File or device already assigned
VOL	Volume or device does not exist

2.41 RVOLUME COMMAND

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

Format:

$$\text{\underline{RVOL}UME voln,}\left\{\begin{array}{l} \text{\underline{A}DD,}\left\{\begin{array}{c} \text{actno}_1\left[/\left\{\begin{array}{c}\text{RW}\\ \text{RO}\end{array}\right\}\right]\left[,\ldots,\text{actno}_{256}\left[/\left\{\begin{array}{c}\text{RW}\\ \text{RO}\end{array}\right\}\right]\right]\\ \text{\underline{A}LL}\left[/\left\{\begin{array}{c}\text{RW}\\ \text{RO}\end{array}\right\}\right]\end{array}\right\}\\ \text{\underline{R}EMOVE,}\left\{\begin{array}{c} \text{actno}_1\left[/\left\{\begin{array}{c}\text{RW}\\ \text{RO}\end{array}\right\}\right]\left[,\ldots,\text{actno}_{256}\left[/\left\{\begin{array}{c}\text{RW}\\ \text{RO}\end{array}\right\}\right]\right]\\ \text{\underline{A}LL}\left[/\left\{\begin{array}{c}\text{RW}\\ \text{RO}\end{array}\right\}\right]\end{array}\right\}\\ \text{\underline{U}SERS}\quad\left[,\left\{\begin{array}{c}\text{actno}\\ \text{actno}_1 - \text{actno}_2\\ \text{0-255}\end{array}\right\}\right]\end{array}\right\}$$

Parameters:

voln — is the volume name of the restricted disc.

ADD — indicates that the specified accounts will have access to the restricted disc.

actno — is a decimal number from 0 through 255 indicating the accounts allowed/disallowed access tc the restricted disc. If ALL is specified, accounts 0 through 255 have access to the restricted disc.

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

RO indicates that the specified account has read only access to the restricted disc.

REMOVE indicates that the specified accounts are disallowed access to the restricted disc. If ALL is specified, all accounts having access to the restricted disc are disallowed access with the exception of the owner's account.

USERS displays all accounts having access to the restricted disc along with the access privileges.

Functional Details:

A disc marked on as a SYSTEM disc is treated as a restricted disc. Account number 255 is the owner.

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

If an owner enters a REMOVE parameter specifying his own account, he will be denied access to the disc; but as 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 disc, accessing private, group, and system files is exactly the same as accessing files on any other disc.

For a user with RO access to a restricted disc, accessing group and system accounts is the same as accessing files on any other disc. 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 disc is dismounted by the system operator, any accounts that once had access no longer have access privileges.

Examples:

```
RVCL 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
RVCL FIXD,R,ALL
RVOL FIXD,U
   82/RW
RVCL 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
```

Messages:

ACCT-ERR — Account number specified is not between 0 and 255

FORM-ERR — Format of the command is invalid

PARM-ERR — Invalid parameter specified or parameter missing in the RVOLUME command

PRIV-ERR — MTM user without access privileges tried to access a restricted disc

VOLN-ERR — Volume specified not online or volume name invalid.

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 in the same manner 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 or semicolon (;) as a line termination. A message cannot be sent to a task currently rolled cut.

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

Messages:

ARGS-ERR	Message exceeded 64 characters
MNEM-ERR	Incorrect command name
NCPR-ERR	No message was provided. The first nonblank character following the SEND command was a carriage return.
SVC6-ERR	SVC 6 error returned indicating the task could not receive a message trap
TASK-ERR	No task loaded

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 cff 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.

Message:

MNEM-ERR	Incorrect command name

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:

```
                                         ⎡              ⎧   fd     ⎫⎤
SIGNON userid [,actno,password] ⎢,ENVIRONMENT=⎨           ⎬⎥
                                         ⎣              ⎩ 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 from 1 through 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 discs 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 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
```

Messages:

DUPLICATE USERNAME	Userid already in use
FORM-ERR	Command syntax error
INVALID ACCOUNT	Invalid or unrecognized account number
INVALID PASSWORD	Password invalid

MISSING PASSWORD	Password omitted
SEQ-ERR	SIGNON command entered when user is already signed on
SIGNON REQUIRED	Attempt to enter a command before signon or mistake in SIGNON command
USERNAME SYNTAX ERROR	Userid invalid

2.45 START COMMAND

The START command initiates execution of a dormant task.

Format:

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

Parameters:

address — specifies the address at which task execution is to begin. For user tasks, this is not a physical address but is 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 pass NOSEG,SCRAT to the task.
ST ,1000,ABC	Starts the currently selected task at transfer address and pass 1000,ABC to the task.

Messages:

FORM-ERR	Command syntax error exists
MNEM-ERR	Incorrect command name specified
PARM-ERR	Operand syntax error exists
SVC6-ERR TYPE=ARGS	Insufficient memory between UTOP and CTOP to pass all parameters
TASK-ERR	No task loaded

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:

$$\underline{\mathtt{T}}\mathtt{ASK}\ \left[\begin{Bmatrix}\mathtt{taskid}\\ \underline{\mathtt{.BG}}\mathtt{ROUND}\end{Bmatrix}\right]$$

Parameters:

taskid — is the name of the taskid that has been loaded into the foreground segment of memory.

.BGROUND — indicates that the task has been loaded as a background task.

Examples:

```
T .BG

T COPY
```

Messages:

MNEM-ERR	Incorrect command name specified
PARM-ERR	Operand syntax error exists

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:

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

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 allocaticn 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 bytes.

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 cf 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 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 logical record length of 126 bytes. The buffer size and index block size default to one sector. The file is assigned to lu 14 cf the loaded task.

Messages:

ALLO-ERR	Allocaticn failed for reason denoted by TYPE field
FCRM-ERR	Command syntax error
LU-ERR	Invalid logical unit number or logical unit assigned
MNEM-ERR	Incorrect command name
NOPR-ERR	Required operand missing

PARM-ERR	Operand syntax error
SPAC-ERR	Task exceeds established maximum system space usage
TASK-ERR	No task loaded

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 as a default. 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's terminal.

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

Example:

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

Messages:

MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error

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.

Messages:

ASGN-ERR — File or device could not be assigned for reason ncted in the TYPE field

FORM-ERR — Command syntax error encountered

I/O-ERR — I/O error encountered on the specified device or file.

LU-ERR	Invalid logical unit encountered
MNEM-ERR	Incorrect command name
PARM-ERR	Operand syntax error encountered
TASK-ERR	No task loaded

2.50 XALLOCATE COMMAND

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

Format:

$$\underline{\text{XA}}\text{LLOCATE fd,}\left\{\begin{array}{l}\underline{\text{CO}}\text{NTIGUCUS,fsize}\left[,\left\{\begin{array}{c}\text{keys}\\ \text{0000}\end{array}\right\}\right]\\ \underline{\text{IN}}\text{DEX}\left[,\left[\left\{\begin{array}{c}\text{lrecl}\\ 126\end{array}\right\}\right]\right]\left[/\left[\left\{\begin{array}{c}\text{bsize}\\ 1\end{array}\right\}\right]\right]\left[/\left[\left\{\begin{array}{c}\text{isize}\\ 1\end{array}\right\}\right]\right]\\ \qquad\left[,\left\{\begin{array}{c}\text{keys}\\ \text{0000}\end{array}\right\}\right]\\ \underline{\text{IT}}\text{AM}\left[,\left[\left\{\begin{array}{c}\text{lrecl}\\ 80\end{array}\right\}\right]\right]\left[/\left[\left\{\begin{array}{c}\text{bsize}\\ 1\end{array}\right\}\right]\right]\left[,\left\{\begin{array}{c}\text{keys}\\ \text{0000}\end{array}\right\}\right]\end{array}\right\}$$

Parameters:

fd — is the file descriptor of the file to be allocated.

CONTIGUOUS — specifies 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 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 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 differs from the ALLOCATE command in the following manner. If the fd to be allocated is a device name instead of a filename, the command is checked for syntax only. 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 will allow either a file or device name to be specified. If a filename is specified, the XALLOCATE command reallocates the file.

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

Messages:

ALLO-ERR	Allocation failed for reasons denoted by TYPE field
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error
MNEM-ERR	Incorrect command name
NOPR-ERR	Required operand missing
PARM-ERR	Invalid parameter

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:

$$\underline{\text{XDE}}\text{LETE fd}_1 \left[\text{,fd}_2\text{,...,fd}_n\right]$$

Parameter:

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

Functional Details:

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

Example:

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

Messages:

DELE-ERR	Delete failed for reason noted by TYPE field
FD-ERR	Invalid file descriptor
MNEM-ERR	Incorrect command name
NOPR-ERR	No operand specified

DELETE error TYPE field respcnses:

PRIV	File currently assigned to a task
PROT	Read/write protection keys not 0
VOL	Specified volume not mounted

CHAPTER 3
PROGRAM DEVELOPMENT COMMANDS

3.1 INTRODUCTION

The program development commands allow a user to:

- specify an existing source file or allocate a new source file to be operated on when a program development command is entered. This source file is called the current program. The commands performing this function are the EDIT command and any of the language commands.
- keep files current throughout a development effort. The program development commands date check the object and image files with the source file and perform a compile, link, and execute, or run when necessary. The commands performing these functions are: COMPILE, COMPLINK, EXECUTE, LINK, and RUN.
- manipulate files when more than one file is involved in a development effort. The commands performing this function are: ADD, ENVIRONMENT, LIST, and REMOVE.

3.2 MAINTAINING LANGUAGE-DEPENDENT INFORMATION

During a development effort, pertinent language-dependent information can be maintained in order to reduce the amount of information a user must enter. By using the appropriate language command (see Table 3-2), the following information can be invoked:

- Standard extensions for source files
- Tab settings when a user is in edit mode
- The compiler to be used
- The run time library to be linked

When a filename is specified in a language command, the system searches the appropriate volume for the source file. If the source file is found, it becomes the current program and is ready to be used. If the source file is not found:

1. A file named filename.ext is allocated on the specified volume or the default volume.

2. The following message is displayed:

```
*** FILE fd is ALLOCATED
```

3. The newly created source file becomes the current program.

4. The appropriate language-dependent information is invoked.

5. The editor is entered with the newly created source file ready to be used.

If a filename is not specified with the language command and no other command was previously specified, the single-module environment is entered. Specification of a filename is accomplished through the program development commands.

3.3 DEVELOPING A PROGRAM

A user can develop a program in one of two ways:

- The entire program can be contained in one file. This is called a single-module environment.

- The program can be contained in several files. This is called a multi-module environment.

Not all program development commands are valid in both types of environments as shown in Table 3-1.

TABLE 3-1 PROGRAM DEVELOPMENT COMMANDS

COMMAND	SINGLE-MODULE	MULTI-MODULE
ADD		x
COMPILE	x	x
COMPLINK	x	x
EDIT	x	x
ENVIRONMENT	x	x
EXECUTE	x	x
LINK	x	x
LIST		x
REMOVE		x
RUN	x	x

If a command is meaningful only in the multi-module environment but is entered in the single-module environment, the following message is displayed:

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

If a development command such as EDIT, COMPILE, COMPLINK, or EXEC is entered without first invoking the language-dependent information, the following message is displayed:

```
    *** LANGUAGE ENVIRONMENT NOT SET
```

For some commands, specifying a filename to identify the current program is only meaningful in a single-module environment. If entered in a multi-module environment, it is ignored.

3.3.1 The Single-Module Environment

Following is a general description of what occurs when a command is entered in a single-module environment.

A single-module environment is accessed by first entering a language command (see Table 3-2).

TABLE 3-2 LANGUAGE COMMANDS

LANGUAGE	COMMAND SYNTAX	STANDARD SOURCE FILE EXTENSIONS
CAL	CAL [[voln:] filename]	.CAL
CAL Macro	MACRO [[voln:] filename]	.MAC
FORTRAN VII	FORT [[voln:] filename]	.FTN (development)
FORTRAN VII	FORTO [[voln:] filename]	.FTN (optimizing)
COBOL	COBOL [[voln:] filename]	.CBL
REPORT PROGRAM GENERATOR	RPG [[voln:] filename]	.RPG

When a command is entered and a filename is specified, the system searches the appropriate volume for the source file. If the source file is not found, the following message is displayed:

```
    *** FILE fd NOT FOUND
```

If the source file is found, it becomes the current program. The system searches for the corresponding object file and date checks to ensure that it is current with the source file. If no object file exists or it is not current with the source file, the source file is compiled. Depending on the command entered, the system then searches for the task image file. If the task image file does not exist or is not current with the object file, the object file is linked. If all files are current, no action is taken. Depending on the command entered, once a current task image file exists it is loaded and run. The following message is displayed:

```
      voln:  filename EXECUTION FOLLOWS
```

When a filename is not specified and a current program exists, the current program's source, object, or image file are date checked; and a compile, link, or execute is performed if necessary. If a current program does not exist, the following message is displayed:

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

3.3.2 The Multi-Module Environment

In a multi-module environment a user deals with several files that, at some point, are linked together to form one load module. Filenames are maintained in an environment descriptor file (EDF), that is created via the ENVIRONMENT command. Other information, such as the programming language each file is written in, is also maintained. As files are created or deleted, their filenames must be added or removed from the EDF.

Not all commands in a multi-module environment need to have a filename specified. When these commands are entered, the requested operation is performed on all the files whose filenames are contained in the EDF. When a filename is specified, as in the case of COMPILE, RUN, and EDIT, the system makes sure that the specified filename exists in the EDF and performs the requested operation on that file.

Following is a general description of what takes place when a command is entered in a multi-module environment.

First, the system searches the appropriate volume for the EDF. Then it searches for all the files whose filenames are listed in the EDF. If all source files are not found, the following message is displayed:

```
*** SOURCE FILE NOT FOUND
```

If the source files are found, the system searches for all object files to date check to make sure they are current with the source files. If object files do not exist or are not current, the corresponding source files are compiled. Depending on the command entered, the system then searches for the task image file to date check. If the task image file does not exist or is not current with the object files, the object files are linked. If all files are current, no action is taken.

Once the task image file is current, it is loaded and run. The following message is displayed:

```
voln: filename EXECUTION FOLLOWS
```

3.4 LINKING

If a link is involved in the operation requested by the user, the following standard Link sequence is used by the system:

```
ESTABLISH TASK
INCLUDE current program
SHARED standard library for language
MAP PR:,ADDRESS,ALPHABETIC
BUILD filename .TSK
END
```

The user can supply a nonstandard Link sequence by allocating a file on the appropriate volume named:

```
voln: filename.LNK
```

In a single-module environment, filename is the filename of the current program. In a multi-module environment, filename is the filename of the EDF.

If this file exists, the system will use the nonstandard Link sequence rather than the standard Link sequence.

The following assumptions are made when linking in a multi-module environment:

- Files to be linked are in object code.
- The standard language subroutine library is used for all languages in the environment.
- No overlay structure is required.

3.5 ASSIGNING LOGICAL UNITS

There are three ways to assign logical units:

1. Use the ASSIGN command.
2. Use a standard command file, voln:filename.ASN. The filename portion of the command file is the filename of the EDF (see Section 3.7.5) in a multi-module environment and the filename of the current program in a single-module environment. This command file is a CSS file that loads the task and makes the requested assignments.
3. Preassign standard or nonstandard pseudo devices within the signon command substitution system (CSS) procedure, USERINIT.CSS. In this case, the pseudo devices are associated with each lu. Table 3-3 shows pairings of logical units and pseudo devices.

TABLE 3-3 LOGICAL UNIT (LU) AND PSEUDO DEVICE PAIRINGS

LU	PSEUDO DEVICE	USE
1	@SSYSIN	Input
2	@SSYSOUT	Output
3	@SSYSLIST	List
5	@SSYSCOM	Command input
7	@SSYSMSG	Messages

The user assigns files and devices to the pseudo devices.

Example:

```
    $SET @SSYSCOM=CON:

    $SET @SSYSPRT=PR:
```

See Chapter 5 for a description of the $SET variable.

These pseudo device assignments can be incorporated as part of the signon CSS, USERINIT.CSS. Once a user sets the pseudo devices, the lu assignments automatically are made at signon time. If a change in devices is desired, the user need only assign another device to the pseudo device. A user can make nonstandard assignments by using pseudo device names @SYSLU1, @SYSLU2, through pseudo device name @SYSLU20. These nonstandard assignments supersede assignments using the pseudo device names listed in Table 3-3.

3.6 LOADING AND RUNNING A TASK

When a task is specified to be run, the system searches for the task image file, filename.TSK. If found, the task is loaded and run. The following message is then displayed:

```
     voln:  filename EXECUTION FOLLOWS
```

If the task image file is not found, the command aborts, and the following message is displayed:

```
    TASK fd NOT FOUND
```

When a filename is not specified, the task corresponding to the existing current program is loaded and run.

If a current program does not exist, the command aborts, and the following message is displayed:

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

3.7 PROGRAM DEVELOPMENT COMMANDS

The program development commands are:

- ADD
- CAL
- COBOL
- COMPILE
- COMPLINK
- EDIT
- ENVIRONMENT or ENV
- EXECUTE
- FORT
- FORTO
- LINK
- LIST
- MACRO
- REMOVE
- RPG
- RUN

The language commands in the list are described in Table 3-2.

3.7.1 ADD Command

The ADD command adds a module to the EDF.

Multi-Module Format:

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

Parameters:

fd — is the file descriptor of the file to be added to the EDF.

cssprod — is the name of the CSS procedure to be used when a nonstandard compilation procedure is required.

Functional Details:

When the ADD command is entered, the current EDF is searched for the specified fd. If the fd is found, the following message is displayed:

```
    *** FILENAME CONFLICT:  ENTRY NOT ADDED
```

If the specified fd is not found, it is added to the EDF. To allocate the file on the user volume, see the EDIT command (see Section 3.7.4).

The cssprod parameter corresponds to the name of the CSS procedure to be used for compilation when a nonstandard compilation procedure is required. If the extension of the specified fd is the same as any one of the extensions listed in Table 3-2, cssprod does not have to be entered. If the extension of the specified fd is nonstandard (see Section 3.2), the cssprod must be specified or the following message is displayed:

```
    *** NON-STANDARD EXTENSION
```

If fd is omitted, the following message is displayed:

```
    *** SYNTAX ERROR
```

3.7.2 COMPILE Command

The COMPILE command produces an object file from the source file of the current program.

Single-Module Format:

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

Multi-Module Format:

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

Parameters:

voln: is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If voln: is not specified, the default is the default user volume.

filename is a 1- to 8-character alphanumeric name specifying the current program. If a filename is omitted in a single- or multi-module environment, the existing current program is compiled.

ALL specifies that all files in a multi-module environment are date checked and compiled if necessary.

3.7.3 COMPLINK Command

The COMPLINK command date checks source, object, and image files of the current program and performs a compile and/or link if necessary.

Single-Module Format:

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

Multi-Module Format:

```
    COMPLINK
```

Parameters:

voln: is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If voln: is not specified, the default is the default user volume.

filename is a 1- to 8-character alphanumeric name specifying the current program. Specification of a filename is only meaningful in a single-module environment.

current program is the existing current program operated on if the filename parameter is omitted in a single-module environment.

Functional Details:

When the COMPLINK command is entered in a multi-module the EDF are date checked, then compiled and linked together.

3.7.4 EDIT Command

The EDIT command enters the user into edit mode.

Single-and Multi-Module Format:

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

Parameters:

voln: is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If voln: is not specified, the default is the default user volume.

filename is a 1- to 8-character alphanumeric name specifying the current program.

current program is the existing current program operated on if the filename parameter is omitted in a single- or multi-module environment.

Single-Module Functional Details:

Use the appropriate language command (see Section 3.2) to enter the single-module environment with language-dependent information invoked.

When a filename is specified, the appropriate volume is searched and if the source file is found, it becomes the current program. If the source file is not found:

1. A source file is allocated.

2. This message is displayed:

```
    *** NEW PROGRAM
```

3. The newly allocated source file becomes the current program.

4. The editor is entered with the newly created source file ready to be used.

When a filename is not specified, the user enters the editor with the file corresponding to the current program in the edit buffer. If there is no existing current program, a source file is allocated as previously described.

Multi-Module Functional Details:

When a filename is specified, the system searches the appropriate volume for the EDF. The EDF is then searched for the specified filename. If the filename is not found in the EDF, the command aborts and the following message is displayed:

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

If the filename is found, the appropriate volume is searched for the source file. If the source file is found, it becomes the current program. If the source file is not found, an empty file with the specified filename is allocated, and the editor is entered with the empty file as the current program.

When a filename is not specified, the user enters the eiitor with the existing current program residing in the edit buffer. If a current program does not exist, the following message is displayed:

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

3.7.5 ENVIRONMENT Command

The ENVIRONMENT command enters a user into a multi-module environment with the specified filename as the filename of the EDF.

Multi-Module Format:

$$\left\{\begin{matrix}\text{ENVIRONMENT}\\ \text{ENV}\end{matrix}\right\} \left[\text{voln:}\right] \text{ filename}$$

Parameters:

voln: is a 1- to 4-character alphanumeric name specifying the volume on which the EDF does or will reside. If the voln: is omitted, the default is the default user volume.

filename ⁺is a 1- to 8-character alphanumeric name specifying the environment descriptor file, filename.EDF. The extension EDF is automatically appended.

Functional Details:

The specified filename is the EDF. If filename.EDF does not exist, a new, empty file is allocated and the following message is displayed:

```
    *** NEW EDF
```

Files can be added and removed from the EDF by using the ADD and REMOVE commands.

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

```
    *** SYNTAX ERROR
```

3.7.6 EXECUTE Command

The EXECUTE command date checks source, object, and image files of the current program and then compiles or links them if necessary. Once there is a current task image file, it is loaded and run.

Single-Module Format:

```
        ┌┌ [voln:] filename ┐┐
EXECUTE ││                  ││
        └└ current program  ┘┘ [,start parameters]
```

Multi-Module Format:

```
EXECUTE  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 default user volume.

filename is a 1- to 8-character alphanumeric name specifying the current task. Specification of a filename is only meaningful in a single-module environment.

start parameters are parameters particular to the compiler, assembler, or link editor being used. These parameters are usually specified in the START command for particular tasks.

current program is the existing current program operated on if a filename is not specified in a single-module environment.

Functional Details:

When the EXECUTE command is entered in a multi-module environment, all files whose filenames are listed in the EDF are compiled and linked if necessary. The task is then loaded and run.

If start parameters are specified in a single-module environment, they are invoked every time the task is run until a different task becomes the current task. If specified in a multi-module environment, the start parameters are invoked every time the task is run until the EDF is changed or the user enters a single-module environment.

3.7.7 LINK Command

The LINK command date checks source and object files and then compiles them if necessary. Object files are then linked to yield the task image file.

Single-Module Format:

```
     ┌┌ [voln:]  filename ┐┐
LINK ││                   ││
     └└ 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 default user volume.

filename is a 1- to 8-character alphanumeric name specifying the current program. Specifying a filename is meaningful only in a single-module environment.

current program is the existing current program operated on if a filename is not specified in a single-module environment.

Functional Details:

When the LINK command is entered in a multi-module environment, all files whose filenames are listed in the EDF are compiled and then linked to yield the task image file.

3.7.8 LIST Command

The LIST command lists all filenames in the current EDF.

Single-Module Format:

```
    LIST
```

Functional Details:

When the LIST command is entered, the listing is sent to the list device specified by @SYSLIST when lu assignments were made.

If the LIST command is entered and there are no filenames contained in the EDF, the following message is displayed:

```
    *** ENVIRONMENT EMPTY
```

3.7.9 REMOVE Command

The REMOVE command deletes a specified filename from the current EDF.

Single-Module Format:

```
    REMOVE fd
```

Parameters:

fd is the file descriptor of the filename to be removed from the multi-module environment.

Functional Details:

When the REMOVE command is entered, the current EDF is searched for the specified fd. If the fd is not found, the following message is displayed:

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

If the specified fd is found, the filename is deleted from the current EDF, and the file is deleted from the volume on which it resides.

If fd is omitted, the following message is displayed:

```
    *** SYNTAX ERROR
```

3.7.10 RUN Command

The RUN command loads and runs the specified task image file.

Single-Module Format:

```
     ┌⎧ [voln:] filename ⎫┐
RUN  │⎨                  ⎬│  [,start parameters]
     └⎩ task image file  ⎭┘
```

Multi-Module Format:

```
     ┌⎧        fd        ⎫┐
RUN  │⎨                  ⎬│  [,start parameters]
     └⎩ task image file  ⎭┘
```

Parameters:

voln: is a 1- to 4-character alphanumeric name specifying the volume on which the task image file resides. If this parameter is omitted, the default is the default user volume.

filename is a 1- to 8-character name specifying the task image file.

fd is the file descriptor of the current task.

start parameters are parameters particular to the assembler, compiler, or link editor being used. These parameters are usually specified in the START command for particular tasks.

task image file is the task image file corresponding to the existing current program. This file is loaded and run if no filename parameter is specified in a single- or multi-module environment.

Example:

Development in a single-module environment:

```
    * FORT                                Specify  FORTRAN  environment.
    * $SET@SYSIN=CON:                     Set pseudo devices.
    * $SET@SYSOUT=CON:
    * $SET@SYSPRT=CON:
    * EXEC TEST                           Find TEST.FTN and compile.
    - FORTRAN:TEST
   *** COMPILE ERRORS, LISTING IN SYSPRT  Compilation errors


    * EDIT
    - EDIT:TEST                           Find and correct errors.
    .
    . (Edit sequence)
    .
    * EXEC                                Attempt to execute.


    - FORTRAN:TEST                        Compile.
    - LINK:TEST                           Linkedit.
    - TEST EXECUTION FOLLOWS              Run
    . (Execution sequence)
    .
    .
    - END OF TASK CODE=0
    * EXEC                                Run again and ensure program is
                                          compiled and linked.


    - TEST EXECUTION FOLLOWS              Compile, link unnecessary.  Object
    . (Execution sequence)                and task up-to-date.
    .
    .


    * RUN
    - TEST EXECUTION FOLLOWS              No compile or link - run only.
    . (Execution sequence)
    .
    .
    - END OF TASK CODE=0
    * EXEC NEWPROG                        Execute NEWPROG.

    *** FILE NEWPROG.FTN NOT FOUND        System finds NEWPROG.MAC (macro
                                          source file) not NEWPROG.FTN.
                                          Must enter macro environment to
                                          execute.
   *
   *
   * MACRO                          Specify macro environment.
   * EXEC NEWPROG                   Execute NEWPROG.MAC
   - MACRO:NEWPROG                  Expand.
   - CAL:NEWPROG                    Assemble.
   - LINK:NEWPROG                   Linkedit.
   - NEWPROG EXECUTION FOLLOWS
   . (Execution sequence)
   .
   .
   - END OF TASK CODE=0
   * EDIT

   - EDIT:NEWPROG
                                    Make changes to NEWPROG (source file).
   . (Edit sequence)
   .
   .
   * EXEC                           Execute NEWPROG.MAC after changes.
   - MACRO:NEWPROG                  Expand
   - CAL:NEWPROG                    Assemble
   - LINK:NEWPROG                   Linkedit
   -NEWPROG EXECUTION FOLLOWS
   . (Execution sequence)
   .
   .
   - END OF TASK CODE=0
```

Examples:

Development in a multi-module environment:

```
   * ENV BIGTASK                    BIGTASK.EDF is EDF.
   *** NEW EDF                      File is allocated, since it did not exist.
   * ADD FOOSUB.CAL                 Add 3 source module names to EDF.
   * ADD MACRTY.CAL
   * ADD FTOR.FTN
   * LIST                           List all modules in multi-module
                                    environment.

   - FOOSUB.CAL
   - MACRTY.CAL
   - FTOR.FTN
   * ADD SUBFUNC.FTN
   * ADD YSUB.MAC
```

```
* REMOVE FOOSUB.CAL
* EDIT
- EDIT:SUBFUNC.FTN                      Make changes to SUBFUNC.FTN
. (Edit sequence)
.
.
* EDIT
- EDIT:YSUB.MAC                         Make changes to YSUB.MAC.
. (Edit sequence)
.
.
* EXEC
- FORTRAN:FTOR.FTN
- FORTRAN:SUBFUNC.FTN                   SUBFUNC.FTN and YSUB.MAC
- MACRO:YSUB.MAC                        object files are outdated.
- CAL:YSUB
- CAL:MACRTY.CAL
- LINK:BIGTASK
- BIGTASK EXECUTION FOLLOWS
. (Execution sequence)                  Execution errors traced to YSUB.MAC
.
.
- END OF TASK CODE=0

* EDIT
- EDIT:YSUB.MAC                         Correct errors in YSUB.MAC.
. (Edit sequence)
.
.
* EXEC                                  SUBFUNC.FTN not compiled - object
                                        file is up to date.

- MACRO:YSUB.MAC                        YSUB.MAC object file is outdated. Expand,
                                        assemble, and linkedit.
- CAL:YSUB.MAC
- LINK:BIGTASK
- BIGTASK EXECUTION FOLLOWS
. (Execution successful)
.
.
- END OF TASK CODE=0
* EDIT                                  Current program set to SUBFUNC.FTN.
- EDIT:SUBFUNC.FTN                      Make changes to SUBFUNC.FTN.
. (Edit sequence)
.
.
* COMPILE
- FORTRAN:SUBFUNC.FTN                   Compile SUBFUNC.FTN
*
```

CHAPTER 4
MULTI-TERMINAL MONITOR (MTM) BATCH PROCESSING

4.1 INTRODUCTION

In addition to interactive processing capabilities, MTM also provides facilities to support concurrent batch processing. MTM allows 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. There is no priority associated with a job in the batch queue. Jobs are processed on a first in, first out basis.

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 consists of a series of operator commands and/or command substitution system (CSS) routines. The commands presented in this section are unique to the batch environment.

4.2.1 INQUIRE Command

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

Format:

INQUIRE [fd] [,fd_1]

Parameters:

fd identifies the job for which the status is desired. If fd is not specified, all jobs with account numbers the same as the user's are displayed.

fd_1 specifies the file or device to which the display is output. If this parameter is omitted, the display is output to the terminal.

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.

Messages:

FD-ERR — Invalid file descriptor entered

FORM-ERR — Command syntax error exists

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:

$$\underline{\text{LO}}\text{G} \left[\text{fd} \left[, \begin{Bmatrix} \underline{\text{N}}\text{OCOPY} \\ \text{COPY} \end{Bmatrix} \right] \right] \left[, \begin{Bmatrix} \text{n} \\ 15 \end{Bmatrix} \right]$$

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:

Checkpointing is only meaningful for indexed files on disc.

Example:

```
LOG PR:
```

4.2.3 PURGE Command

The PURGE command purges a submitted job.

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.
```

Messages:

FD-ERR	Invalid file descriptor entered
FORM-ERR	Command syntax error exists
JOB NOT FOUND	

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

Message:

MNEM-ERR	Incorrect command name

4.2.5 SIGNON Command

SIGNON must be the first command in a batch job.

Format:

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

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

Parameters:

userid — is a 1- to 8-character alphanumeric string specifying terminal user identification.

actno — is a 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 discs 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
```

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 created by the user 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 job will run is relative to the priority established at sysgen time via the SGN.PRIO option. See the OS/32 Multi-Terminal Monitor (MTM) System Planning and Operator Reference Manual.

Usually when default priorities are established at sysgen time, interactive tasks run at one priority lower than MTM; batch jobs run at two priorites lower than MTM. For example:

Assume SGN.PRIO = 1

MTM	(128)	priority of MTM
interactive jobs	MTM+1(129)	one lower than MTM
batch jobs	MTM+2(130)	two lower than MTM

Whatever priorities are established for the above tasks at sysgen time are considered MTM default priorities. A user task (u-task), which initially has a priority established when it is built (Link priority), can have a new priority set when it is submitted. If no priority is specified at submit time, the job is run at the default priority or the priority established when the task was built (Link priority), whichever is lower. The rules for establishing priorities are:

- Batch jobs can be specified to run at the same priority as interactive tasks but not higher than interactive tasks. This would be considered invalid.
- If a priority is specified, the batch job will run at that priority if valid, or the Link priority, whichever is lower. If the priority is not valid, the following message is displayed, and the default priority is assigned by MTM:

```
WARNING - REQUESTED PRIORITY n1 ILLEGAL, n2 USED
```

- If no priority is specified, the batch job will run at the default priority or the Link priority, whichever is lower.

This 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 as follows:

```
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 as follows:

```
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
```

Messages:

BTCH-ERR	Batch capability not available
FD-ERR	Invalid file descriptor
FORM-ERR	Command syntax error

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 can later be resubmitted as a batch jobs 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 set to 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 DISCS ON BATCH JOBS

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

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

CHAPTER 5
COMMAND SUBSTITUTION SYSTEM (CSS)

5.1 GENERAL DESCRIPTION

The Command Substitution System (CSS) is an extension to the OS/32 command language. It enables the user to establish files of dynamically modifiable commands which 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 small number of 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 that complex command sequences can be developed.

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

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

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. When the leading characters of an fd are the same as a command, then MTM assumes a command:

CLO.CSS	CLOSE
ASP12.CSS	OK
PC12.CSS	OK
AS3.CSS	ASSIGN 3
ASG3.CSS	OK

However, by specifying a volume name and/or extension, a CSS file can be called that might normally conflict with an MTM supported command. The following are all valid calls to a CSS file with the fd of CLOSE:

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

If the file extension is omitted, the CSS extension is assumed. If the file does not exist as specified, the error message MNEM-ERR is returned.

Parameters are passed to a CSS file by appending them to the fd of the CSS file. If a parameter contains the double quote character (") then all characters up to the next double quote are passed. The double quotes themselves are not passed. The first parameter must be separated from the filename or device by a space; all other parameters must be separated by commas. Null parameters are permitted. Valid CSS calls are:

RUN — calls CSS file RUN.CSS on the default user volume.

JUMP A,B,C, — calls CSS file JUMP.CSS on the default user volume with three parameters A,B,C.

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

VOLN:JUMP — calls CSS file JUMP.CSS on the volume VOLN.

ABC P1,"P2A, P2B,P2C",P3 — calls CSS file ABC.CSS on the default user volume with three parameters. Parameter 1 is P1. Parameter 2 is P2A, P2B,P2C. Parameter 3 is P3.

5.3 USE OF PARAMETERS

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
```

A reference to a parameter is terminated by a non-numeric character.

Example:

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

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

Notice that 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 considered to be 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 considered to be null.

Parameter @0 is a special parameter used to reference the name of the CSS file it is contained in. Parameter @0 is replaced during the preprocessing of the command line with precisely the same style descriptor 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 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 variables:

- Global variables
- Local variables

Global variables exist from signon to signoff or until they are freed via the $FREE command. A local variable can be used only within the CSS level in which it was 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 and the remaining 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, the $SET command is used.

5.4.4 Reserved Variables

Variable names starting with the character string @SYS are reserved for system use in the program development environment. If a user tries to define variables starting with @SYS, a message is displayed; however, a user does have read and write access to @SYS variables from within a particular development environment.

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 $ helps to indicate where a CSS has been 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:

```
      (fd)
$BUILD{    } [,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 is used to terminate a CSS stream. This command causes closing of all CSS files and deactivation of CSS.

Format:

```
    $CLEAR
```

Functional Details:

The $CLEAR command may 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 turns on the listing and all subsequent commands are copied to the terminal before being executed. The $NOCOPY command turns off the listing, but is itself listed. The $COPY command is effective in debugging CSS job streams.

Format:

```
    $COPY

    $NOCOPY
```

5.5.5 $EXIT Command

The $EXIT command is used to terminate a CSS procedure. Control is then returned to the calling CSS procedure of the terminal if the CSS procedure was called from the terminal. All commands on the same line after the CSS call are ignored.

Format:

```
    $EXIT
```

5.5.6 $FREE Command

The $FREE command frees one or more variables.

Format:

$\$\text{FREE varname}_1[,\ldots,\text{varname}_n]$

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 length of the value to which it will be set by the $SET command.

Format:

$$\text{\$GLOBAL varname} \left[\begin{pmatrix}\text{length}\\ 8\end{pmatrix}\right]_1 \left[,\ldots,\text{varname}\left[\begin{pmatrix}\text{length}\\ 8\end{pmatrix}\right]_n\right]$$

Parameters:

varname — is a 1- to 8-character name (the first character is alphabetic) 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 boundary for a CSS job. A CSS job consists of all the terminal user commands and tasks loaded and started between a $JOB and $TERMJOB. The $JOB command indicates the start of a CSS job and the $TERMJOB indicates the end of a CSS job. These commands need not be included in a CSS procedure but are useful in preventing errors in one CSS job from affecting subsequent CSS job processing. Most errors encountered in executing terminal user commands in a CSS job cause the remaining statements to be skipped until a $TERMJOB is encountered. The skips to $TERMJOB occur if the error is detected within a CSS job. The job is aborted and the next command executed is the first command after $TERMJOB. At this point the end of task code is 255. If the error occurs outside a job, CSS is aborted. If $TERMJOB is omitted, errors can cause a subsequent $JOB statement to be skipped.

Separating CSS jobs delimited by $JOB and $TERMJOB statements eliminates the chance of errors in one job affecting another.

$JOB resets the end of task code to 0.

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 specifyi g 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.

Interactive jobs have no default limits established at sysgen time. However, the terminal 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 length of the value to which it will be set by the $SET command.

Format:

$$\text{\$LOCAL varname}\left[\begin{pmatrix}\text{length}\\ 8\end{pmatrix}\right]_1 \left[\text{,...,varname}\left[\begin{pmatrix}\text{length}\\ 8\end{pmatrix}\right]_n\right]$$

Parameters:

varname   is a 1- to 8-character name (the first character is alphabetic) 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.

5.5.11 $SET Command

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

Format:

```
    $SET varname=e
```

Parameter:

variable= 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 be used 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 value between 0 and 255.

5.5.13 $SKIP Command

The $SKIP command is used between the $JOB command and $TERMJOB. 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:

$$\text{\$WAIT}\ \left[\begin{Bmatrix} n \\ 1 \end{Bmatrix}\right]$$

Parameter:

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

Functional Details:

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

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

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.

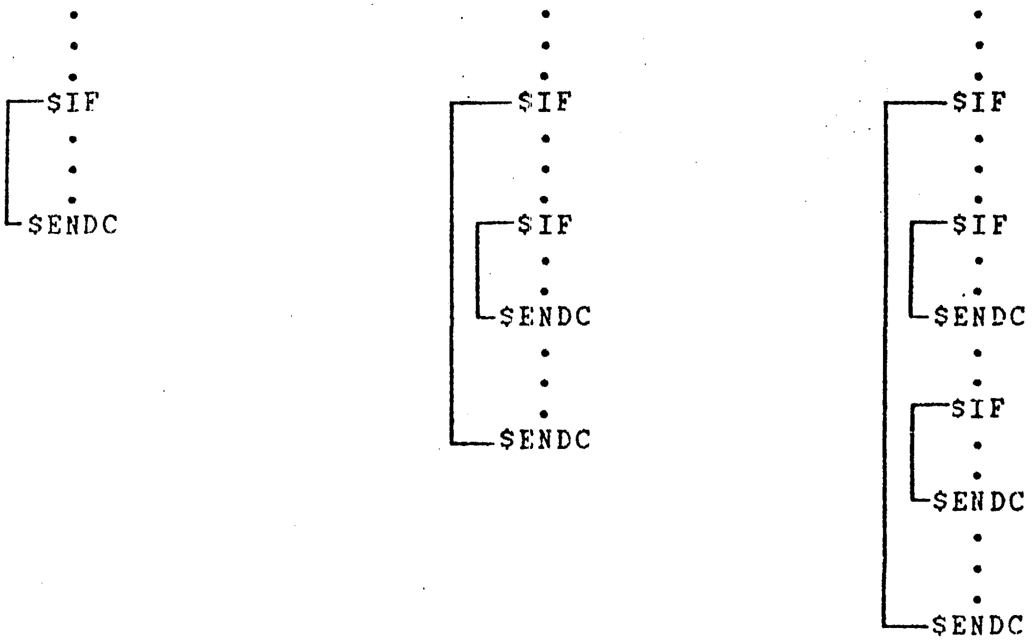
5.6 LOGICAL IF COMMANDS

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 command establishes a condition that is tested by the CSS processor. If the result of this test is true, then 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 EOT 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 such skipping attempts to skip beyond EOF, a command error is given.

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 such skipping attempts to skip beyond EOF, a command error is given.

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:

```
    $IFNULL @1@2@3
```

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

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
```

Label is from 1 to 8 alphanumeric characters, the first of which must be alphabetic.

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 is resumed.

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 at the time of execution of the $GOTO.

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 corresponding $ELSE or $ENDC. If an extension does not exist, subsequent commands are skipped up to the corresponding $ELSE or $ENDC.

Format:

```
    $IFEXTENSION fd
```

Parameter:

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

5.9 $IFVOLUME Command

The $IFVOLUME command is used to test for the existence of a volume in an fd. If a volume exists, subsequent commands are executed up to a corresponding $ELSE or $ENDC. If the volume field was omitted in the file descriptor, subsequent commands are skipped up to the corresponding $ELSE or $ENDC.

Format:

```
    $IFVOLUME fd
```

Parameter:

fd is the file descriptor tested to determine if a volume 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 number.

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.

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:

$$\text{\$IF} \left\{ \begin{array}{c} \underline{\text{CH}}\text{ARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{array} \right\} \text{arg}_1 \ \underline{\text{E}}\text{QUAL arg}_2$$

$$\text{\$IF} \left\{ \begin{array}{c} \underline{\text{CH}}\text{ARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{array} \right\} \text{arg}_1 \ \underline{\text{NE}}\text{QUAL arg}_2$$

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

The $IF...GREATER command is used to determine if arg_1 is greater than arg_2. The $IF...NGREATER command is used to determine if arg_1 is not greater than arg_2.

Format:

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{GR}}\text{EATER}\ \text{arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{NG}}\text{REATER}\ \text{arg}_2$$

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

The $IF...LESS command is used to determine if arg_1 is less than arg_2. The $IF...NLESS command is used to determine if arg_1 is not less than arg_2.

Format:

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{LE}}\text{SS}\ \text{arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{NLE}}\text{SS}\ \text{arg}_2$$

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 provides for copying a batch stream of cards such as source programs, operator command, command substitution system (CSS) files, or other user data onto a disc 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 disc 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 disc 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.

NOTE

> 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 disc file named TEST.DTA (associated with account number 12).

It deletes and reallocates 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 disc and submits the file as a batch job. The commands are executed in sequence. The file remains on the disc after execution is complete.

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.

Example:

There are two methods for submitting a batch job using the Spooler.

Method 1:

First a CSS file is copied from a card file to a disc 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 disc 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 disc, it is deleted and reallocated. After this batch job completes, the file TEST.JOB remains on the disc:

```
/@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 device simultaneously. Data to be printed or punched is written to disc 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 disc space is available to accommodate output spooling. The user task (u-task) is responsible for handling end of medium (EOM) status while writing to spool files within their own standard I/O error recovery routines.

Printing multiple copies of a disc file or punching multiple copies of a card deck is accomplished through use of the Spooler. To print or punch a disc 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 as:

```
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 request:

```
    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 file can be printed from the system console by entering a .SPL PRINT command whenever the Spooler is started.

APPENDIX A
MULTI-TERMINAL MONITOR (MTM) COMMAND SUMMARY

```
              ⎧ CONTIGUCUS,fsize ⎡,⎧ keys ⎫⎤
              ⎪                  ⎣ ⎩ 0000 ⎭⎦
              ⎪
              ⎪ INDEX ⎡,⎡⎧lrecl⎫⎤⎤ ⎡/⎡⎧bsize⎫⎤⎤ ⎡/⎡⎧isize⎫⎤⎤
              ⎪       ⎣ ⎣⎩ 126 ⎭⎦⎦ ⎣ ⎣⎩  1  ⎭⎦⎦ ⎣ ⎣⎩  1  ⎭⎦⎦
ALLOCATE fd, <                                                   >
              ⎪       ⎡,⎧ keys ⎫⎤
              ⎪       ⎣ ⎩ 0000 ⎭⎦
              ⎪
              ⎪ ITAM ⎡,⎡⎧lrecl⎫⎤⎤ ⎡/⎡⎧bsize⎫⎤⎤ ⎡,⎧ keys ⎫⎤
              ⎩      ⎣ ⎣⎩ 80  ⎭⎦⎦ ⎣ ⎣⎩  1  ⎭⎦⎦ ⎣ ⎩ 0000 ⎭⎦


                ⎡ ⎡⎧access privileges⎫⎤                  ⎡ ⎧SVC15⎫⎤⎤
                ⎢ ⎢⎪       SRW       ⎪⎥ ⎡ ⎧ keys ⎫⎤      ⎢ ⎪     ⎪⎥⎥
ASSIGN lu,fd    ⎢,⎢<       SREW      >⎥ ⎢,⎨      ⎬⎥      ⎢,⎨SVCF ⎬⎥⎥
                ⎢ ⎢⎪                 ⎪⎥ ⎣ ⎩ 0000 ⎭⎦      ⎢ ⎪     ⎪⎥⎥
                ⎣ ⎣⎩       SRO       ⎭⎦                  ⎣ ⎩ VFC ⎭⎦⎦


BFILE [fd,] lu


       ⎧address⎫
BIAS   ⎨       ⎬
       ⎩   *   ⎭


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

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

DISPLAY FLOAT [,{      fd     }]
              [ {user console }]
```

```
                        fd
DISPLAY LU        [,{              }]
                     user console


                               fd
DISPLAY PARAMETERS    [,{              }]
                         user console


                              fd
DISPLAY REGISTERS    [,{              }]
                        user console


                          fd
DISPLAY TIME     [,{              }]
                    user console


                           fd
DISPLAY USERS     [,{              }]
                     user console


         ( MESSAGE  )
         ( PROMPT   )
ENABLE  {  ETM       }
         ( $VARIABLE)


                    ( ,n         )     (     fd        )
EXAMINE address1  [{  /address2   }] [,{               }]
                    ( ,1         )     ( user console  )


EFILE [fd,] lu


ERECORD [fd,] lu


       (mnemonic)
HELP  {          }
       (   *    )


INIT fd [,{segsize increment}]
          {        1        }
```

```
  INQUIRE [fd] [,fd1]

| LOAD [taskid,]fd[,segsize increment]

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

             (userid   )
  MESSAGE   {          }  message
             (.OPERATOR)

                      ┌(data1)┐
  MODIFY address,     │{     }│ [,data2,...,datan]
                      └(  0  )┘

            ┌(AFPAUSE   )┐  ┌ (SVCPAUSE   )┐
  OPTIONS   │{           }│  │,{           }│  [,NONRESIDENT]
            └(AFCONTINUE)┘  └ (SVCCONTINUE)┘

  PAUSE

|            (MESSAGE  )
|            (PROMPT   )
|  PREVENT  {ETM       }
|            ($VARIABLE)

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

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

  PURGE fd

  RENAME oldfd,newfd

  REPROTECT fd,new keys
```

```
REWIND [fd,] lu

    or

RW [fd,] lu

                ⎧          ⎧actno1 [/⎧RW⎫] [,...,actno256 [/⎧RW⎫]] ⎫ ⎫
                ⎪          ⎪         ⎩RO⎭                   ⎩RO⎭   ⎪ ⎪
                ⎪  ADD,    ⎨                                       ⎬ ⎪
                ⎪          ⎪              ALL [/⎧RW⎫]               ⎪ ⎪
                ⎪          ⎩                    ⎩RO⎭               ⎭ ⎪
                ⎪          ⎧actno1 [/⎧RW⎫] [,...,actno256 [/⎧RW⎫]] ⎫ ⎪
                ⎪          ⎪         ⎩RO⎭                   ⎩RO⎭   ⎪ ⎪
RVOLUME voln,   ⎨  REMOVE, ⎨                                       ⎬ ⎬
                ⎪          ⎪              ALL [/⎧RW⎫]               ⎪ ⎪
                ⎪          ⎩                    ⎩RO⎭               ⎭ ⎪
                ⎪                                                    ⎪
                ⎪                      ⎡  ⎧     actno       ⎫⎤       ⎪
                ⎪  USERS               ⎢, ⎨actno1 - actno2  ⎬⎥       ⎪
                ⎩                      ⎣  ⎩     0-255       ⎭⎦       ⎭

SEND message [;]

SIGNOFF

                                          ⎡              ⎧  fd     ⎫⎤
SIGNON userid [,actno,password]           ⎢,ENVIRONMENT= ⎨         ⎬⎥
                                          ⎣              ⎩NULL [:] ⎭⎦
        [,CPUTIME=maxtime]
        [,classid=iocount1 [,...,classid=iocount32]]

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

       ⎡⎧     address      ⎫⎤
START  ⎢⎨                  ⎬⎥ [,parameter1,...,parametern]
       ⎣⎩ transfer address ⎭⎦

       ⎡⎧ taskid   ⎫⎤
TASK   ⎢⎨          ⎬⎥
       ⎣⎩ .BGROUND ⎭⎦
```

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

$$\underline{\text{V}}\text{OLUME }[\text{voln}]$$

$$\underline{\text{WF}}\text{ILE }[\text{fd,}]\text{ lu}$$

$$\underline{\text{XAL}}\text{LOCATE fd,}\left\{\begin{array}{l}\underline{\text{CO}}\text{NTIGUCUS,fsize}\left[,\begin{Bmatrix}\text{keys}\\ 0000\end{Bmatrix}\right]\\ \underline{\text{IN}}\text{DEX}\left[,\left[\begin{Bmatrix}\text{lrecl}\\ 126\end{Bmatrix}\right]\right]\left[/\left[\begin{Bmatrix}\text{bsize}\\ 1\end{Bmatrix}\right]\right]\left[/\left[\begin{Bmatrix}\text{isize}\\ 1\end{Bmatrix}\right]\right]\\ \qquad\left[,\begin{Bmatrix}\text{keys}\\ 0000\end{Bmatrix}\right]\\ \underline{\text{IT}}\text{AM}\left[,\left[\begin{Bmatrix}\text{lrecl}\\ 80\end{Bmatrix}\right]\right]\left[/\left[\begin{Bmatrix}\text{bsize}\\ 1\end{Bmatrix}\right]\right]\left[,\begin{Bmatrix}\text{keys}\\ 0000\end{Bmatrix}\right]\end{array}\right\}$$

$$\underline{\text{XDE}}\text{LETE } \text{fd}_1\left[,\text{fd}_2,\ldots,\text{fd}_n\right]$$

APPENDIX B
PROGRAM DEVELOPMENT COMMAND SUMMARY

```
ADD fd  [,cssprod]

CAL  [[voln:] filename]

COBOL  [[voln:] filename]

         [( [voln:]  filename  )]
COMPILE  [{                    }]
         [( current program    )]

         [( [voln:]  filename  )]
COMPILE  [{        ALL         }]
         [( current program    )]

          [( [voln:]  filename )]
COMPLINK  [{                   }]
          [( current program   )]

      [( [voln:]  filename )]
EDIT  [{                   }]
      [( current program   )]

(ENVIRONMENT)
{           }  [voln:]  filename
(    ENV    )

         [( [voln:   filename] )]
EXECUTE  [{                    }]
         [( current program    )]  [,start parameters]

EXECUTE  start parameters
```

```
FORT [[voln:] filename]

FORTO [[voln:] filename]

      ┌ ⎧ [voln:]  filename ⎫ ┐
LINK  │ ⎨                   ⎬ │
      └ ⎩ current program   ⎭ ┘

LINK

LIST

MACRO [[voln:] filename]

REMOVE fd

RPG [[voln:] filename]

     ┌ ⎧ [voln:]  filename ⎫ ┐
RUN  │ ⎨                   ⎬ │
     └ ⎩ task image file   ⎭ ┘[,start parameters]

     ┌ ⎧        fd         ⎫ ┐
RUN  │ ⎨                   ⎬ │
     └ ⎩ task image file   ⎭ ┘ [,start parameters]
```

APPENDIX C
MULTI-TERMINAL MONITOR (MTM) COMMAND SUBSITITUTION SYSTEM (CSS) COMMAND SUMMARY

$\$\text{BUILD}\left\{\begin{matrix}\text{fd}\\ \text{lu}\end{matrix}\right\}\ [\text{,APPEND}]$

•
•
•

$ENDB

$CLEAR

$CONTINUE

$COPY

$NOCOPY

$ELSE

$ENDC

$EXIT

$\$\text{FREE varname}_1\ [\text{,...,varname}_n]$

$\$\text{GLOBAL varname}\left[\begin{pmatrix}\text{length}\\ 8\end{pmatrix}\right]_1\left[\text{,...,varname}\left[\begin{pmatrix}\text{length}\\ 8\end{pmatrix}\right]_n\right]$

$GOTO label

$LABEL label

$$\text{\$IF}\begin{Bmatrix}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{Bmatrix}\text{arg}_1\ \underline{\text{EQ}}\text{UAL arg}_2$$

$$\text{\$IF}\begin{Bmatrix}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{Bmatrix}\text{arg}_1\ \underline{\text{NE}}\text{QUAL arg}_2$$

$$\text{\$IF}\begin{Bmatrix}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{Bmatrix}\text{arg}_1\ \underline{\text{GR}}\text{EATER arg}_2$$

$$\text{\$IF}\begin{Bmatrix}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{Bmatrix}\text{arg}_1\ \underline{\text{NG}}\text{REATER arg}_2$$

$$\text{\$IF}\begin{Bmatrix}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{Bmatrix}\text{arg}_1\ \underline{\text{LE}}\text{SS arg}_2$$

$$\text{\$IF}\begin{Bmatrix}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{Bmatrix}\text{arg}_1\ \underline{\text{NLE}}\text{SS arg}_2$$

$IFE n

$\underline{\text{\$IFEX}}\text{TENSION fd}$

$IFG n

```
$IFL n

$IFNE n

$IFNG n

$IFNL n

$IFNULL @n

$IFNNULL @n

$IFVOLUME fd

$IFX fd

$IFNX fd

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

               ⎡⎛length⎞⎤   ⎡         ⎡⎛length⎞⎤ ⎤
$LOCAL varname ⎢⎜      ⎟⎥ , ⎢,...,varname ⎢⎜      ⎟⎥ ⎥
               ⎣⎝  8   ⎠⎦1  ⎣         ⎣⎝  8   ⎠⎦n⎦

$NOCOPY

$PAUSE

$SET varname=e

$SET CODE n
```

$\underline{\text{\$S}}\text{KIP}$

$\text{\$WAIT} \left[\begin{Bmatrix} n \\ 1 \end{Bmatrix}\right]$

$\underline{\text{\$W}}\text{RITE text } [;]$

APPENDIX D
TERMINAL USER COMMAND MESSAGE SUMMARY

ACCT-ERR

The account number specified is not between 0 and 255.

ALLO-ERR TYPE=NAME

A desired filename currently exists on the specified volume.

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

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

ALLO-ERR TYPE=TYPE

The volume specified is not a direct access device.

ALLO-ERR TYPE=VOL

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

ARGS-ERR

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

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 is 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 at a nonexistent file.

ASGN-ERR TYPE=PRIV

A file, which is 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 statement 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 disc to allocate a physical block.

ASGN-ERR TYPE=SPAC

An assign is refused because the system space available for task use 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 discs 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=BUFF

There is insufficient memory available in system space to delete an indexed file.

DELE-ERR TYPE=NAME

File with a specified name was not found.

DELE-ERR TYPE=PRIV

An attempt is being made to delete a file that is currently assigned.

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 disc is being loaded and there is not enough system space for the load operation.

fd IS NOT A CONTIGUOUS FILE

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

FORM-ERR

The command format is invalid.

GOTO-ERR

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

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 EOF or EOM before completing the operation.

I/O-ERR TYPE=FUNC

An invalid operation is being directed toward a device, e.g., attempting to write 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 has 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

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 a device that is unavailable.

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 has not been built with Link.

LOAD-ERR TYPE=MEM

A load is attempted when a large enough segment is unavailable.

LOAD-ERR TYPE=MTCB

The maximum number of tasks specified at sysgen time has been 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, and 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.

MISSING PASSWORD

Password omitted.

MNEM-ERR

The command entered is unrecognizable.

NOFP-ERR

No floating point support exists in the system.

NOPR-ERR

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

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

The new 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.

SEQ-ERR

A command was entered out of sequence. Terminate or pause the currently active task and re-enter the command.

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 usage.

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 not dormant.

SVC6-ERR TYPE=NMSG

The task currently executing at the terminal could not receive a message trap.

SVC6-ERR TYPE=PRES

The Spooler is not loaded, and a request is made that requires this program.

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's account CPU limit expired.

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
COMMAND SUBSTITUTION SYSTEM (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 EOT 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.

@SYSXXX-VARIABLE ERROR, ILLEGAL NAME

indicates that a variable was defined beginning with the reserved characters @SYS or a system variable was attempted to be freed.

@XXXX-VARIABLE ERROR, ALREADY EXISTS

indicates that a local variable that already exists attempted to be defined.

@XXXX-VARIABLE ERROR, EXCEEDS USER LIMIT

indicates that the variable limit set at sysgen was exceeded.

@XXXX-VARIABLE ERROR, DEFINITION TOO LONG

indicates that the length of the defined variable is greater than 32.

@XXXX-VARIABLE ERROR, DOES NOT EXIST

indicates that the value of a variable that does not exist was attempted to be set or freed. Also, during CSS execution, a variable definition is required and that variable does not exist.

@XXXX-VARIABLE ERROR, DEFINITION DOES NOT EXIST

indicates that the value of a variable was attempted to be set to the value of a second variable that does not exist.

@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 COMMAND MESSAGE SUMMARY

*** COMPILE ERRORS, LISTING IN fd

Errors were encountered while compiling. A listing of these errors is found in the specified fd.

*** CURRENT PROGRAM NOT SPECIFIED

A filename was not specified, and no current program exists.

*** ENVIRONMENT EMPTY

The LIST command was specified, but there were no filenames in the EDF.

*** FILE fd IS ALLOCATED

The file specified in a language command could not be found; a new file was allocated.

*** FILE fd NOT FOUND

The source file corresponding to the specified filename could not be found. This message is output in a single-module environment.

*** FILENAME CONFLICT: ENTRY NOT ADDED

An attempt was made to add a filename to the EDF, but that filename was already listed in the EDF.

*** FILENAME NOT IN ENVIRONMENT

The specified filename was not found in the EDF.

*** LANGUAGE ENVIRONMENT NOT SET

A development command such as EDIT, COMPILE, COMPLINK, or EXEC was entered without first invoking language-dependent information.

*** NEW EDF

The filename specified in the ENVIRONMENT command does not exist. An empty file has been allocated.

```
*** NEW PROGRAM
```

The editor was entered with a nonexistent file; a new file is allocated.

```
*** NON-STANDARD EXTENSION
```

An attempt was made to add a filename with a nonstandard language extension to the EDF without the cssprod parameter specified.

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

A command that is only meaningful in a multi-module environment was specified in a single-module environment.

```
*** SOURCE FILE NOT FOUND
```

The source file corresponding to the specified filename could not be found. This message is output in a multi-module environment.

```
*** SYNTAX ERROR
```

A filename was not specified.

```
*** TASK fd NOT FOUND
```

The specified task image file could not be found.

```
voln: filename EXECUTION FOLLOWS
```

A task image file has been loaded and run.

The following messages indicate the steps of program development and are displayed for user convenience:

```
CAL: voln: filename

COBOL: voln: filename

EDIT: voln: filename

FORTRAN: voln: filename

LINKEDIT: voln: filename

MACRO: voln: filename

RPG: voln: filename
```

INDEX

CHAPTER 5
COMMAND SUBSTITUTION SYSTEM (CSS)

5.1 GENERAL DESCRIPTION

The Command Substitution System (CSS) is an extension to the OS/32 command language. It enables the user to establish files of dynamically modifiable commands which 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 small number of 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 that complex command sequences can be developed.

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

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

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. When the leading characters of an fd