M T S

The Michigan Terminal System

# The Michigan Terminal System

**Volume 1**

**Reference R1001**

**November 1991**

University of Michigan
Information Technology Division
Consulting and Support Services

DISCLAIMER

The MTS manuals are intended to represent the current state of the Michigan Terminal System (MTS), but because the system is constantly being developed, extended, and refined, sections of this volume will become obsolete. The user should refer to the *Information Technology Digest* and other ITD documentation for the latest information about changes to MTS.

# CONTENTS

# PREFACE

The software developed by the Information Technology Division (ITD) technical staff for the operation of the high-speed IBM 370-compatible computers can be described as a multiprogramming supervisor that handles a number of resident, reentrant programs. Among them is a large subsystem, called MTS (Michigan Terminal System), for command interpretation, execution control, file management, and accounting maintenance. Most users interact with the computer's resources through MTS.

The *MTS Volumes* are a series of manuals that describe in detail the facilities provided by the Michigan Terminal System. Administrative policies of ITD and the physical facilities provided are described in other publications.

The *MTS Volumes* now in print are listed below. The date indicates the most recent edition of each volume; however, since volumes are periodically updated, users should check the file *ITDPUBLICATIONS, or watch for announcements in the *Information Technology Digest*, to ensure that their volumes are fully up to date.

| | |
|---|---|
| *Volume 1:* | *The Michigan Terminal System*, Reference R1001, November 1991 |
| *Volume 2:* | *Public File Descriptions*, Reference R1002, January 1987 |
| *Volume 3* | *System Subroutine Descriptions*, Reference R1003, April 1981 |
| *Volume 4:* | *Terminals and Networks in MTS*, Reference R1004, July 1988 |
| *Volume 5:* | *System Services*, Reference R1005, May 1983 |
| *Volume 6:* | *FORTRAN in MTS*, Reference R1006, October 1983 |
| *Volume 7:* | *PL/I in MTS*, Reference R1007, September 1982 |
| *Volume 8:* | *LISP and SLIP in MTS*, Reference R1008, June 1976 |
| *Volume 9:* | *SNOBOL4 in MTS*, Reference R1009, September 1975 |
| *Volume 10:* | *BASIC in MTS*, Reference R1010, December 1980 |
| *Volume 11:* | *Plot Description System*, Reference R1011, August 1978 |
| *Volume 12:* | *PIL/2 in MTS*, Reference R1012, December 1974 |
| *Volume 13:* | *The Symbolic Debugging System*, Reference R1013, September 1985 |
| *Volume 14:* | *360/370 Assemblers in MTS*, Reference R1014, May 1983 |
| *Volume 15:* | *FORMAT and TEXT360*, Reference R1015, April 1977 |
| *Volume 16:* | *ALGOL W in MTS*, Reference R1016, September 1980 |
| *Volume 17:* | *Integrated Graphics System*, Reference R1017, December 1980 |
| *Volume 18:* | *The MTS File Editor*, Reference R1018, February 1988 |
| *Volume 19:* | *Magnetic Tapes in MTS*, Reference R1019, November 1991 |
| *Volume 20:* | *Pascal in MTS*, Reference R1020, December 1985 |
| *Volume 21:* | *MTS Command Extensions and Macros*, Reference R1021, April 1986 |
| *Volume 22:* | *Utilisp in MTS*, Reference R1022, May 1988 |
| *Volume 23:* | *Messaging and Conferencing in MTS*, Reference R1023, February 1991 |

The numerical order of the volumes does not necessarily reflect the chronological order of their appearance; however, in general, the higher the number, the more specialized the volume. *MTS Volume 1*, for example, introduces the user to MTS and describes in general the MTS operating system, while *MTS Volume 10* deals exclusively with BASIC.

The attempt to make each volume complete in itself and reasonably independent of others in the series naturally results in a certain amount of repetition. Public file descriptions, for example, may appear in more than one volume. However, this arrangement permits the user to buy only those volumes that serve his or her immediate needs.

Richard A. Salisbury
General Editor

# PREFACE TO VOLUME 1

The November 1991 edition of *MTS Volume 1* reflects the changes that have been made to MTS since the February 1988 edition.

Sections have been added describing the use of Resource Manager.

References and descriptions have been added for the FTP, LOCATE, and VIEW commands. In addition, several new options have been added for the DISPLAY and SET commands.

# A BRIEF OVERVIEW OF MTS

MTS, the Michigan Terminal System, is a terminal-oriented, time-sharing operating system that offers terminal, server, and batch facilities. It was originally developed by the University of Michigan Computing Center for an IBM 360/67, and has since been modified to run on the IBM System/370 and Extended Architecture (XA) systems. It is currently running as the production system at several academic installations[1] on the IBM 370, 3033, 3090, 4300, and ES/9000 systems and the Amdahl 470, 580, and 590 systems. It has also been distributed to other installations and run on other computers.

The purpose of this section is to provide a brief description of the capabilities of MTS and an overview of its appearance to the user. Detailed information on various aspects of the Michigan Terminal System are given in the following sections of this volume and in other volumes describing the system.

## HISTORY

Development of MTS began in 1966 on an IBM 360/50 (a computer without dynamic-relocation hardware). In January 1967, the system was converted to the newly arrived IBM 360/67, and in the spring of 1967, MTS was offered to the University of Michigan community. At that time the system could support about five terminal users and one batch stream, due to the storage limitations inherent in a non-relocate system.

In November 1967, a new version of MTS was put into service, which exploited the relocation hardware of the 360/67 for address translation and made use of a high-speed drum for paging. Only changes in the supervisor were necessary in order to effect this transition, and neither the MTS operating system itself nor user programs were affected. As a result of this change, a tenfold increase in the number of tasks that could be serviced by the system at any one time was achieved. In August 1968, a dual-processor 360/67 replaced the single-processor 360/67, and shortly thereafter a version of the system that supported multiprocessing was installed.

The development of the MTS operating system has been shared by various MTS installations. The University of British Columbia Computing Centre was the first MTS installation to install an IBM System/370 computer (a 370/168); thus the modifications to the MTS operating system for this equipment were made there. Over the years, MTS has been installed on the following processors at the University of Michigan:

| | |
|---|---|
| August 1968 | IBM 360/67 |
| January 1975 | IBM 370/168 |
| July 1975 | Amdahl 470V/6 |
| March 1979 | Amdahl 470V/7 |
| September 1980 | Amdahl 470V/8 |
| November 1982 | Amdahl 5860 |
| October 1986 | IBM 3090-400 |
| July 1988 | IBM 3090-600E |

------------------

[1]The original MTS installations are: University of Alberta, University of British Columbia, Durham University, University of Michigan, University of Newcastle upon Tyne, Rensselaer Polytechnic Institute, Simon Fraser University, and Wayne State University.

| | |
|---|---|
| June 1991 | IBM ES/9000-720 |

In order to give some idea of the capacity of MTS, the following numbers relating to use at the University of Michigan are offered. The maximum number of terminal users that can be signed on simultaneously is over 600 (on both systems combined). The number of batch jobs that can be run simultaneously is controlled by the system's load-leveling program, and varies from 3 to 10. There are currently over 450,000 files in the file system. At the peak of a semester, about 50,000 distinct user identification codes (userIDs) are authorized. There are about two dozen remote batch terminals in various parts of the University, other universities, and governmental agencies.

## ACCESS TO THE SYSTEM

As its name implies, the Michigan Terminal System was designed for access from interactive terminals, although the system also has extensive facilities for providing servers and processing batch jobs.

Under MTS, terminal, server, and batch access differ from each other as little as possible. The command language is identical, and the same translators and utility programs are available in each mode. However, servers and batch are usually inappropriate for jobs requiring interaction between the user and the program.

Although some users need only one type of access, the majority of users use all types: terminals for program development and debugging, batch for bulk-data input and output and production running (for noninteractive programs), and servers for accessing specialized services from MTS.

Terminal access depends on the type of terminal being used. The following terminal types are currently supported by MTS (this is not an exhaustive list; see *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004, for further details):

The Ontel Terminal
The IBM PC, the Apple Macintosh, and other microcomputers
The DEC VT100 Terminal
The IBM 3278 Terminal
The Tektronix 4010 Storage Tube Display
The Visual 550 Display

Most terminals that are compatible with any of the above terminals are also supported. Most of these terminals (except for the IBM 3278s) are linked to MTS through the UMnet/Michnet Computer Network by means of hardwire connections or the telephone dial-up facilities. Details on using terminals through UMnet/Michnet are given in *Introduction to Terminals and Microcomputers*, Tutorial T7005, and in the section "Merit/UMnet" in *MTS Volume 4*. Details on using the Ontel terminals are given in the section the "The Ontel Terminal" in *MTS Volume 4*.

MTS is connected to the UMnet/Michnet Computer Network which provides direct access to the computing facilities at Wayne State University, Michigan State University, Western Michigan University, and several other Michigan colleges and universities. Because of the UMnet/Michnet connection, access to MTS at the University of Michigan is also possible from any location served by the Telenet commercial network.

## THE FILE SYSTEM

Unlike many other operating systems in which the user has to be aware of disk volumes and remember which files reside on which disk, in MTS each reference by the user to a file stored on a public volume is made only via the file name. Ordinarily, the user does not know which volume contains a given file. The name is private to a user, thus allowing any user in the system to have a file by the same name. Internal to the system, the name is qualified by the userID. Most files are line files, which allow either direct access to any line by means of the line number, or else sequential access from one line to the next. All files are physically stored in fixed-length records of 4096 bytes each (which matches the virtual memory page size).

Normally, each userID has unlimited access to its own files, and no other userID has any access to them. However, the user can permit any ID, group of IDs, or specific programs and/or commands to have access to any specified file. The basic access classes are read, write-append, write-change, destroy, and permit; any one or any combination (including NONE) of these may be chosen for any combination of userIDs and files, independent of the other files.

## DEVICE SUPPORT

At the time of the initial development of MTS, it was recognized that an ever increasing number of different devices could be expected. As a result, there is a firmly specified, centralized interface for I/O. The program that controls a device is called a device support routine (DSR); there is a DSR for each type of device supported. There is also a single common interface to these DSRs, and *all* I/O (including system I/O) goes through this common interface. The interface manages the details of I/O processing that are device-independent; all device-dependencies are managed entirely by the DSR. The package of routines that comprises a DSR has several entry points to manage various aspects of I/O, such as opening, closing, reading a record, writing a record, control operations, etc. Since all devices and most people are record-oriented, I/O in MTS is also record-oriented.

## ACCOUNTING AND BILLING

In order to manage an authorization and accounting system for the user accounts and to do line-item billing for these accounts, an elaborate structure has evolved. The following describes the part most apparent to users.

There are several files maintained by the system for accounting purposes. One file has a line in it for every valid userID, which contains an encoded form of the user's password, the maximum allowed amount and the current cumulative amount for several quantities, and various other items of billing information. Among the information monitored are terminal time, CPU time, a virtual storage integral, file space in use, a file space integral, dollars spent, the number of signons, batch runs and terminal runs, and the number of cards read, cards punched, pages printed and lines printed. Before a user is allowed to sign on, the cumulative dollars spent are compared with the maximum dollars allowed; various other maxima fields are also checked at appropriate times.

Another file contains statistics on system use, one line for each session (signon to signoff period) which the user has with the system. This line is written after signoff and contains a detailed summary of the billable resources used during that session. The billing is subsequently done using information from this file.

Billing is based on several items. Hardware and labor costs are divided into categories, and the charge per unit in each category is the cost of that category divided by the total use in that category. Four of the items are unit-record charges: a cost per card read, per line printed, per page-printer images and sheets printed, per page printed (materials cost), and per card punched. Both batch and terminal runs are charged per second of CPU time, and for "tenancy," which is a virtual memory used vs. CPU-time integral. Terminal sessions also incur a charge per connect-hour to pay for transmission controllers, etc. File storage is charged for on a file space vs. elapsed-time integral. Charges are also made for tape mounts, tape-drive tenancy, plotting time, etc.

Public servers are free and an account is not needed to access them.

## BULK-DATA INPUT AND OUTPUT

Batch users may enter input data via punched cards and either batch or terminal users may receive output via printed paper or punched cards.

Magnetic tapes may be read and written both by batch and terminal users. Introductory information about submitting, retrieving, and using magnetic tapes is given in *Introduction to Magnetic Tapes*, Tutorial T7002. Reference documentation for magnetic tapes is given in *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019.

## VIRTUAL MEMORY

One of the primary constraints facing programmers has been the amount of addressable memory available. Addressable memory refers to storage that can be addressed directly. In the past, when a program required more than the maximum main storage available, the "ping-pong" or "overlay" method was often used. Using this technique, the programmer divided the program into self-contained sections, each small enough to fit into main storage. Each section was loaded from an auxiliary storage device such as a magnetic tape or disk and executed separately, and it was up to the programmer to control the order, loading, and linking of the sections. This was no simple task.

The concept of virtual memory, implemented in the IBM 360/67 and the IBM System/370 computers, allows the use of more addressable memory than is physically available. In effect, the system does the "ping-ponging" and the programmer does not have to worry about any of the details. The system breaks the program into sections called pages and manages the transfer of pages between main storage and suitable auxiliary devices. This transferring of pages is called *paging*.

Virtual memory, which can be many times larger than actual physical memory, is divided into *segments*. A segment is 256 pages and a page is 4096 bytes (characters); thus, there are 1,048,576 bytes in one segment. There are 64 segments which are allocated as follows:

| | |
|---|---|
| Seg. 0–5: | Resident system programs |
| Seg. 6–7: | System storage for user's job |
| Seg. 8: | Resident system programs (NAS) |
| Seg. 9–55: | Storage available for user programs |
| Seg. 56–59: | System storage for user's job |
| Seg. 60–63: | Resident system programs (NAS) |

The primary advantage of the virtual memory concept is apparent—each user has access to a very large address space. The system ensures that when a page is needed, it is brought into main storage if it is not already present. In order to utilize the system efficiently, it is desirable to minimize the number of different pages needed in rapid succession.

## RESOURCE MANAGER

The Resource Manager (RM) is a system that was developed to handle all the facilities associated with printing and batch processing on a large mainframe computer. It is an integral part of the Michigan Terminal System (MTS) and is invoked automatically whenever a job is initiated that requires one of these facilities:

(1) page, local, or line printers

(2) batch processing

(3) BITNET file transfers

(4) BITNET messages

The Resource Manager

(1) assigns a job number and job name in order to keep track of the job;

(2) assesses the requirements of a job, based on certain defaults, the size of a job, and information supplied on $CONTROL and $SIGNON commands;

(3) assigns a priority based on this assessment, and passes a job to the appropriate queue to await processing;

(4) releases a job for processing when the required device or facility is available;

(5) directs output to the device indicated by a job (if this is a printer, it recognizes keywords that might have been used to modify the manner of printing);

(6) removes a job from the system upon completion or cancellation;

(7) retains a record of a job for a period of time;

(8) maintains a log of a job from start to finish.

## THE USER'S VIEW

The user can access the system from a terminal, by submitting a batch job, or by invoking a server. Access from a terminal is called conversational mode (or terminal mode); access from a batch job is called batch mode; access from a server is called server mode. Switching between terminal and batch mode is not difficult since all files, commands, and facilities are available in each mode.

## Terminal Jobs

When a terminal user calls in, a number of signon messages are printed after the connection is established. A new line beginning with a pound sign (#) is then printed and the terminal waits for input. This initial character, which is printed on both input and output lines, is called a *prefix character* and indicates "who is talking or listening;" i.e., a (#) means the user is in MTS command mode. The first command from the user must be a SIGNON command or HELP command. A user whose userID is WXYZ would enter

```
        SIGNON WXYZ
```

After the SIGNON command has been entered, the user is prompted to enter the password associated with the userID given on the SIGNON command.

## Server Connections

Server connections allow users to sign on and request a special service while bypassing the dialogue of an interactive MTS terminal session. The user gives the system the name of the desired server and is automatically connected to the server. The server then instructs the user how to enter the necessary commands to obtain the desired service. Some servers will charge the userID for the service rendered; other servers are free.

## Batch Jobs

For a batch job, the user submits a deck of cards via a card reader or else uses a terminal to enter, edit, and submit the "cards." Each batch job is assigned a job receipt number that is used to identify all printed and punched output produced by the job. Printed output includes a *header sheet* containing the job receipt number and userID and a *tail sheet* containing the job receipt number and a statistical summary of the job that includes the approximate cost of the job. Batch and terminal use are essentially identical in commands and in action, with the exception that no interaction is possible in batch; if some parameter is incorrect or omitted, instead of prompting for a replacement (as would be the case in conversational mode), MTS terminates the command (an exception is the SIGNON command where an error terminates the entire job). In addition, the batch user is required to give maximum limits on the SIGNON command for the number of pages the job will print, the number of cards it will punch, and the amount of CPU time it will use (if any of these is not given, a default is supplied). An attempt to exceed any of these limits causes termination of the job.

## Files and Editing

Once signed onto the system, the user may wish to create a file (with a CREATE command) or modify an existing file. Almost all files the user works with are line files. A line file consists of a collection of zero or more lines. Associated with each line is a line number (which is *not* part of the line), and lines are ordered by line number. The line number appears to the user as a number in the range –2147483.648 to 2147483.647.

A user may create and use private files (i.e., files belonging to the userID), and temporary files (files automatically created when first referenced, and automatically destroyed when the user signs off). Temporary files are distinguished by a minus sign "–" which prefixes the file name. Users may also access public files, which usually contain translators or utility programs; a public file name is distinguished by having an asterisk (*) as the first character.

Users usually enter data into a file or modify data already in a file by using the MTS File Editor. The File Editor acts as a separate subsystem with its own command language and is known as the EDIT CLS (command language subsystem). The user enters edit mode by entering a EDIT command with the name of the file to be edited as a parameter. While in edit mode, the prefix character is the colon (:), which helps remind the user at all times that the system is in edit mode. The INSERT, DELETE, and REPLACE commands may be used in this mode to change entire lines, the CHANGE command to change part of a line, the OVERLAY command to overlay information on top of a line, and so on. Each File Editor command may include a line number parameter to indicate which line or lines to work on, as for example

```
        ALTER 10 'ABC'DEF'
```

alters the string "ABC" to "DEF" in line 10. One can also use the File Editor in a purely context-driven fashion, using SCAN and similar commands to find the appropriate lines, and then the pertinent command without a line number in order to work on the current line.

Further information about the MTS File Editor is given in *MTS Volume 18: The MTS File Editor*, Reference R1018.

## File and Devices Names

One of the most important concepts that unifies data referencing in MTS is that of the general file-or-device name (FDname). This is illustrated using the LIST and COPY commands. Both of these commands have two parameters which are FDnames, and proceed by reading lines from the file or device specified by the first parameter and writing them on the file or device specified by the second parameter. LIST differs from COPY in that the line number of the line is appended to the front of the output line. If no second parameter is given, a default is assigned. For batch jobs, the default device is the line printer; for terminal jobs, it is the display screen. Thus,

```
        LIST ALPHA
```

prints each line in the file ALPHA, starting at line 1 (or the first line thereafter, if line 1 does not exist), and prints the line number on the front of each line. The command

```
        LIST ALPHA(-10)
```

does the same, but it starts at line −10, whereas

```
        LIST ALPHA(5,11,2)
```

lists the lines 5, 7, 9, and 11 (from 5 to 11 by increments of 2).

Modifiers may also be attached to an FDname. Normally,

```
        COPY ALPHA BETA
```

makes a copy of file ALPHA in file BETA, with lines renumbered in BETA to start at 1 in increments of 1. A copy of a file that preserves existing line numbers may be made by specifying an indexed attribute on the receiving file:

```
        COPY ALPHA BETA@I
```

This specifies that in writing each line into BETA, an indexed-write operation should be performed using the line number received when the line was read from ALPHA.

In place of the file name, the user may specify a pseudodevice name, which can identify a real device, such as *T* for a tape that has been mounted, or which can stand for some logical I/O entity, such as *SOURCE* for the command stream source, *SINK* for command output, *PRINT* for the output stream to a printer, etc.

Lastly, the FDname may be "concatenated." An *explicit* concatenation consists of two or more FDnames connected with plus signs (+):

```
LIST ALPHA(1,10)+BETA
```

An *Implicit* concatenation occurs if a line read from a file or device is of the form

```
$CONTINUE WITH FDname [RETURN]
```

where the brackets indicate that RETURN is optional. The dollar sign ($) is always required. Reading is transferred to the FDname specified, transparent to whatever program is doing the reading. If RETURN is specified, then an end-of-file condition on FDname causes reading to proceed at the line following the "$CONTINUE WITH" line in the original file. The FDname in an implicit concatenation can, of course, have the full form of an FDname as described above, including explicit concatenations, etc.

Commands for manipulating files include CREATE, DESTROY, DUPLICATE, RENUMBER, RENAME, PERMIT to set access rights for a file, FILESTATUS and FILEMENU to inquire about various properties of a file, and EMPTY to retain a file but empty its contents so that something new can be put in it. In conversational mode, both EMPTY and DESTROY require user confirmation to provide some protection against accidental loss of data.

## Program Execution

The RUN command loads and executes a program. For example, the command

```
RUN *TAPESUBMIT
```

loads and executes the object module contained in the file *TAPESUBMIT. Programs that are run may refer to the named logical I/O units INPUT, PRINT, OBJECT, etc. (by calling the subroutines INPUT, PRINT, OBJECT, etc.) or to units 0 through 99 (by calling the subroutines READ and WRITE). Assignments of FDnames to logical I/O units are made by means of keyword parameters on the RUN command. Thus, if the program in the file PROGRAM gets its input from logical unit INPUT (the standard system input stream), then to specify the file FYLE as input the user would issue

```
RUN PROGRAM INPUT=FYLE
```

To read from line 25 to the end of the file FYLE, the user would issue

```
RUN OBJ INPUT=FYLE(25)
```

The DEBUG command is the same as the RUN command, but instead of starting execution of the program directly, debug mode is entered after loading so that execution of the program may be monitored. See *MTS Volume 13: The Symbolic Debugging System*, Reference R1013, for information on the DEBUG command.

From the viewpoint of the system, translators are *user programs* which are executed via the RUN command. For example, to compile a FORTRAN program, the user enters RUN *FORTRANVS giving appropriate assignments for logical I/O units: INPUT, the source program; PRINT, the listing output; OBJECT, the object deck output, etc. Parameters are passed to the program via the optional PAR field.

## MTS COMMAND LANGUAGE

In order to use the computer, one must first get the attention of MTS and be identified through the SIGNON command. Once this has been done and MTS has declared the signon to be legitimate, the system is at the user's disposal.

In this section, the commands are listed in their most common form followed by a brief explanation of their function. Not every possible option and parameter is mentioned. A detailed explanation of the commands is given in the section "The System Command Language" in this volume. The synopsis of the complete syntax for the MTS command language is given in the *MTS Reference Summary*, a pocket-sized guide to MTS.

### Global Control

| | |
|---|---|
| SIGNON userid | tells MTS that the user with userID "userid" wants to use the system. The password associated with the userID must be entered on the following line. |
| SIGNOFF | signs the user off the system and gives a summary of the cost of the run. |
| SET | is used to set various options and variables in the system. These control such functions as setting the user's password and/or name, mail/message notification, printed output routing, etc. |
| SINK FDname | switches the current output sink to the file or device "FDname" instead of the printer (in batch mode) or the terminal (in conversational mode). |
| SOURCE FDname | switches the current input source to the file or device "FDname" instead of the card reader (in batch mode) or the terminal (in conversational mode). |

### Program Control

| | |
|---|---|
| RUN FDname | loads and executes the program in the file "FDname". These files can be public, such as *FORTRANVS or *PASCALJB, or private, such as MYPROGR or –LOAD. |
| RERUN | reissues the previous RUN command. |
| LOAD FDname | loads but does not execute the program in the file "FDname". The program can then be DISPLAYed and/or ALTERed. Execution can be started by the START command. |

| | |
|---|---|
| START | starts the execution of a program which has already been loaded by a LOAD command. |
| RESTART | restarts the program at the point of the last interrupt or at a specified location. START and RESTART are synonymous and may be used interchangeably. |
| UNLOAD | releases storage and devices acquired by the previous LOAD or RUN. It is useful when the execution of a program did not terminate normally. |
| IF | tests program execution return code. |

**Debugging**

| | |
|---|---|
| DEBUG FDname | loads the program in the file "FDname" under the control of the Symbolic Debugging System (SDS). SDS has its own set of commands which allows the user to interactively debug a program. A listing of the important debug commands is given later. |
| SDS | returns control to SDS from MTS. |
| DUMP | prints out the contents of the general registers, floating-point registers, and all the virtual memory locations associated with the current job. |

**File Handling**

| | |
|---|---|
| CREATE filename | creates a file named "filename". If it is not a temporary file (that is, if its name does not begin with a minus sign), it is retained until the user destroys it. |
| DESTROY filename | destroys the file named "filename" and returns the space on the disk to the pool of available storage. |
| EMPTY filename | empties the contents of the file "filename", but retains its space and identity. Thus, the file can be reused. |
| RENAME filename1 filename2 | renames the file "filename1" to "filename2". |
| RENUMBER filename | renumbers the line numbers in the file "filename". |
| TRUNCATE filename | deallocates unused space at the end of file "filename". |
| COPY FDname1 TO FDname2 | copies the contents of the file or device "FDname1" into the file or device "FDname2". Unless specified otherwise, the lines in "FDname2" are numbered from 1 with increments of 1 regardless of their numbers in the old file. "FDname1" is retained unaltered. |

| | |
|---|---|
| DUPLICATE oldname AS newname | copies the contents of the file “oldname” exactly into the file “newname”. The original line numbers are retained. |
| LIST FDname | lists the contents of the file or device “FDname”. |
| RELEASE | releases *PRINT*, *BATCH*, or *PUNCH*, or pseudodevices mounted by the MOUNT command. |
| EDIT filename | enters edit mode. The MTS File Editor has its own set of commands which allow the direct editing of the file “filename”. |
| PERMIT filename access | specifies the access that other users have to the file “filename” to be “access”. |
| FILESTATUS | gives file status information for the user’s files. |
| FILEMENU | gives file status information in full-screen format. |

**Miscellaneous**

| | |
|---|---|
| CALC | enters into the desk calculator mode of operation. |
| CONTROL FDname xx | performs the file or device control operation “xx” on the file or device “FDname”. |
| DISPLAY option | displays information about the status of the system or the user’s task. |
| FSMESSAGE | enters into the Full-Screen Message System. |
| FTP | enters the File Transfer Protocol system. |
| LOCATE | gives job status information. |
| LOG ON FDname | begins printing a log (copy) of a terminal session on the file or device “FDname”. |
| MAKE program | enters the program management facility which builds the contents for the file “program” depending on rules given in a make-file. |
| MESSAGE | enters into the MTS Message System. |
| MOUNT xx | requests the mounting of the magnetic tape, server, or UMnet/Michnet Computer Network connection “xx”. |
| NET xx | enters into the UMnet/Michnet Computer Network system for the connection “xx”. |
| SYSTEMSTATUS | gives status information for a job or the system. |

| | |
|---|---|
| VIEW | looks at queued batch and print jobs. |

## THE EDITOR COMMAND LANGUAGE

After the user has entered the command EDIT xx, MTS turns control over to the MTS File Editor, which is then ready to accept commands concerning the editing of the file named “xx”.

Files may be edited either by line number or by context.

The File Editor maintains a current line pointer which is initially set to the first line of the file. Commands that require line number specification may refer to the line pointer or may explicitly mention the appropriate lines.

Visual-mode (or full-screen mode) editing is available for dedicated MTS terminals such as the Ontel and the IBM 3278, and for microcomputers such as the IBM PC (and its compatibles), the Apple Macintosh, and the DEC VT100.

An introductory description of the MTS File Editor is given in *Introduction to the MTS File Editor*, Tutorial T7006. The complete description of the File Editor is given in *MTS Volume 18: The MTS File Editor*, Reference R1018.

## THE DEBUG COMMAND LANGUAGE

The Symbolic Debugging System (SDS) is a conversational program checkout facility which aids in debugging programs. The MTS command DEBUG xx has the same format as the RUN command. It tells MTS to load the program in file “xx” and then to give control to SDS. Using the SDS command language, the programmer may initiate execution of a program and monitor its flow by specifying breakpoints where instructions and data may be displayed and modified.

The programmer may refer to locations in the program symbolically, by relative address, or by absolute (virtual or loaded) address. Symbolic referencing can be used only with those language processors which generate a symbol table with the object programs they produce. Currently, this includes the 360/370 assembler, the FORTRAN G and H compilers, the VS FORTRAN compiler, the PL/I (F) compiler, the PL360 compiler, the PLUS compiler, and the C87 compiler. If the user has requested that SDS display the contents of a particular location—using any of the above three ways to specify the location—SDS consults the symbol table, if there is one, to get the proper type and length of the location. If there is no symbol table, the contents are printed in hexadecimal in units of 4 bytes.

An introduction to SDS is given in *Introduction to Programming and Debugging in MTS*, Tutorial T7004. The complete description of SDS is given in *MTS Volume 13: The Symbolic Debugging System*, Reference R1013.

# USERIDS, LIMITS, AND SIGFILES

The ITD userID (also called the signon ID) identifies the user to the system and differentiates him or her from other users. It is a 4-character alphanumeric identifier assigned when an application for use of the mainframe computer is approved. Each userID is provided for a particular use; thus, a student enrolled in two courses using the computer would have a separate userID for each course. Each userID has independent limits on computer resources and each has separate accounting information maintained for it.

Every userID is associated with a project. A project may consist of several userIDs or just a single ID, but a userID can belong to only one project. Each project has a 4-character identifier called the "projectID." For instructional use, one projectID is usually assigned to a course, and each student in the course has a userID belonging to that project. A research group might have one projectID for the whole group, and each individual within the group would have a userID belonging to that project.

A project director or instructor may request that the project be organized so that the ACCOUNTING command may be used to allocate computer resources among the userIDs in the project. Billing and the transfer of funds to pay for use of the computer are based on projectIDs.

ProjectIDs are also useful if programs or data stored in a file belonging to one signon are to be shared by members of the project. (See the PERMIT command for details.)

## PASSWORDS

Whenever a user wishes to use MTS, the first command entered in a terminal session (or the first card of a batch job) must be a SIGNON command specifying the userID. To protect the user against unauthorized use of a userID, a password of from 1 to 12 characters (blanks and commas are not allowed) is assigned when the userID is created. The system stores an encoded form of the password in the accounting record for the userID. When the user enters the SIGNON command, the system inspects the accounting record to determine whether the userID is valid and whether funds remain. If both of these conditions exist, the system prompts the terminal user to enter the password (for a batch job the password should be entered on the card following the SIGNON command). The system encodes the user's password and then compares it with the one maintained in the accounting record. If they are identical, the SIGNON procedure is completed and the user may begin the computing activities. The algorithm for encoding passwords is nearly irreversible; hence, it is impossible for anyone, including ITD personnel, to recover a lost password. If a user forgets his or her password, an application must be made to the ITD Accounts Office for a new one.

The user can change the password for the userID whenever he or she is signed on by issuing a "SET PW" command. Users should change their passwords frequently. *Anyone* who knows a userID and its password has access to the funds and files of that userID. UserIDs are essentially public knowledge; they are printed conspicuously on all batch output to facilitate distribution. Therefore users should take care to see that printing of passwords is always suppressed. Methods for suppressing the printing of passwords are given in the section "Batch Use of MTS" and in *Introduction to Terminals and Microcomputers*, Tutorial T7005. If the password is printed, that portion of the output should be destroyed and then discarded; merely throwing it in a wastebasket does *not* guarantee that security will be maintained. Additional security may be obtained by using a password validation scheme with sigfile processing (see "Sigfiles and Security" below).

To change the password, the SET PW command is used. For batch jobs, the command sequence is given as

```
SET PW
newpw newpw
```

where "newpw" is the user's new password (starting in column 1). The password must be entered twice to reduce the possibility of setting the password to an unknown word due to a typing error.

For terminal jobs, the user is first prompted for the current (old) password and then prompted for the new password. As an aid to guard against typing errors when entering the new password, MTS requires the terminal user to verify the new password by entering it a second time. If both versions of the new password are the same, the new password becomes the current password; if they are different, an error message is printed and the password remains unchanged. Thus for terminal users, the command sequence is

```
SET PW
Enter old user password.
oldpw
Enter new user password.
newpw
Reenter new user password to confirm.
newpw
```

Automatic input blanking is provided for both batch and terminal jobs to prevent either the old password or the new password from printing on the user's printed output or terminal.

As an additional measure of protection, the system will print the message

```
Password has been changed.
```

If this message appears under suspicious circumstances, the user should immediately change the password before signing off (if in conversational mode) and notify a consultant or the ITD Accounts Office.

Attempts to sign on to a userID that fail because of improper password entry are logged by MTS and reported at the next successful signon with the following message:

```
There have been n unsuccessful attempts to sign on
to this account since the last successful signon.
```

where "n" is the number of signon attempts. This message, when used in conjunction with the last signon date and time, allows a user to detect all attempted uses of the account, whether successful or not. If this message is received often or with a large value for "n" and authorized users of the account are not responsible for the unsuccessful attempts, individual users may wish to contact ITD for further assistance. Any actual unauthorized use of an account should be reported promptly to the ITD Accounts Office.

## TIME AND FUNDS LIMITS

The computer resources used by a userID can be limited at three levels: in the accounting record, in an individual batch job or terminal session, and in the execution of individual programs within a batch job or terminal session.

In the accounting record, each userID normally has at least three limits: funds, file space, and an expiration time and date. Other limitations (e.g., prohibiting terminal use) may be requested. MTS is designed to prevent the user from overdrawing his or her account. The funds limit is checked when the user attempts to sign on. If the balance is zero or negative, the signon is not allowed. If the balance is positive, and the job is a batch job, the system checks to ensure that the remaining balance is sufficient to cover the CPU time specified on the SIGNON command; if the balance is insufficient, the job is not run. For each terminal session, the system simply stops when the user runs out of money.

At the individual session level, the user may specify a *global CPU time limit* in seconds as part of the SIGNON command, e.g.,

```
SIGNON WXYZ T=10
```

If the global time limit is exceeded on a batch job, the message

```
Global time limit exceeded [at xxxxxxxx]
```

is printed and the job is terminated (the location “xxxxxxxx” is printed only if the interrupt occurred while a program was executing). If the global time limit is exceeded on a terminal job, the above message is printed and the user is prompted for permission to continue what was interrupted or to return to MTS command mode. From this point, the job is continued without a global time limit.

The system also sets a funds-depletion time limit on the SIGNON command which is an *estimate* of the amount of CPU time needed to expend the remainder of the money allotted to the account. When this time limit is exceeded, the system recomputes the balance of funds. If the balance is still positive, a new time limit is set and the job is continued. If the balance is negative, the following message is printed:

```
You have run out of money [at xxxxxxxx]
```

(the location “xxxxxxxx” is printed only if the interrupt occurred while a program was executing). The current balance of funds is printed and the user is prompted to continue whatever was interrupted or to return to MTS command mode. The user is then given a new time limit of 3 seconds. Thereafter, at 3-second (CPU time) intervals, the warning is repeated.

*Warning:* The system will not prevent the user’s account from accumulating a deficit balance and will not force the user to terminate the session. If the initial estimate by the system of the CPU time needed to deplete the remaining balance of funds is inaccurate (due to the type of computing being performed by the user’s job) or if another person is concurrently signed onto the same account and is expending funds, the deficit incurred may be substantial. Rebates will not be granted because the system fails to prevent an account from being overdrawn.

At the individual program-execution level, the system imposes no time limits; however the user can specify a *local CPU time limit* on any RUN, RERUN, LOAD, DEBUG, RESTART, or START command to limit the execution time of the program, e.g.,

```
RUN *TEX T=5
```

sets a local time limit of 5 CPU seconds for the executing program. If no local limit is specified, program-execution times are limited only by the remaining portion of the global limit. For a *terminal* session, if no explicit global or local time limit is set, the only limit is the remaining money in the user’s account. Therefore, it is *strongly advised* that users specify explicit global or local time limits.

Since it is inconvenient to enter a local time limit on every RUN, RERUN, etc., command, the SET TIME=n command allows the user to establish a default local time limit. The value specified becomes the default local time limit until another SET TIME=n command is entered or until the user signs off. Users must remember to SET TIME=n for every batch or terminal job if they want a default local time limit (this may be easily done using a sigfile). An explicit local time limit presented on the RUN, etc., command always overrides the default value. Whenever a local limit is reached, program execution is interrupted and a message of the form

```
    Local time limit exceeded [at xxxxxxxx]
```

is printed. The user can restart execution by using the RESTART command. The user should specify a new local time limit on the RESTART command if the default value is inappropriate, .e.g.,

```
    RESTART T=3
```

A warning message of the form

```
    WARNING: Program being run with no local time limit.
```

is printed if the RESTART command is given without a local time limit when a previous local time limit was in effect.

A global time limit takes precedence over a local time limit only if the local limit is greater than the global limit.

Some programs cannot be restarted if they are interrupted because a time limit has been reached. Such programs (e.g., *WATFIV and *PLC) may establish an internal time limit that is smaller than the one the user specified; if the internal limit is reached, program execution is interrupted so that a meaningful dump can be produced.

The user can determine the cost of the session at any time by entering the command

DISPLAY COST

The user can also determine, after each MTS command, the incremental and total cost of the session by issuing the command

SET COST=ON

Similar information is available for the execution of programs by using the SET command options EBM and ETM.

The user may specify either a global or local plot-time limit for either a batch or terminal job. A global plot-time limit may be specified on the SIGNON command in the form

SIGNON WXYZ PT={t | tM}

and a local plot-time limit may be specified in the same form on the RUN, RERUN, START, RESTART, LOAD, or DEBUG commands. If given in the form “t”, the limit is in seconds; if given in the form “tM”, the limit is in minutes. If a global plot-time limit is exceeded, the plot job is automatically terminated; if a local plot-time limit is exceeded, the executing program is terminated. The default global and local plot-time limits are none which means that the plot-time limit is unrestricted.

## PAGE AND CARD LIMITS

For batch jobs, the user may specify limits for either the number of pages of output printed or the number of cards punched. This may be done either globally for the entire job or locally for a single RUN, RERUN, START, RESTART, LOAD, or DEBUG command.

A global page limit may be specified on the SIGNON command in the form

```
SIGNON WXYZ P=p
```

where "p" is the number of pages that may be printed. If the number of pages generated by the job exceeds this limit, the job is automatically terminated, and the message

```
Global page limit exceeded [at xxxxxxxx]
```

is printed where "xxxxxxxx" is the same as for the global time limit message described above. The default is 50 pages. The maximum number that may be specified is 99999.

A global card limit may be specified on the SIGNON command in the form

```
SIGNON WXYZ C=c
```

where "c" is the number of cards that may be punched. If the number of cards generated by the job exceeds this limit, the job is automatically terminated, and the message

```
Global card limit exceeded [at xxxxxxxx]
```

is printed. The default is 0 cards and the pseudodevice *PUNCH* is undefined if no global card limit is specified (in batch mode). The maximum number that may be specified is 99999.

Local page and card limits may also be specified on the RUN, RERUN, START, RESTART, LOAD, and DEBUG command. The form of the specification is the same as above for the SIGNON command. If the number of pages or cards generated by the executing program exceeds either limit, the program is automatically terminated, and the message

```
Local page limit exceeded [at xxxxxxxx]
```

or

```
Local card limit exceeded [at xxxxxxxx]
```

is printed.

## SIGFILES

Many option settings and default values, such as the default local time limit mentioned above, can be modified by the user. The standard settings usually satisfy the needs of the majority of users, but, obviously, they cannot satisfy all users. Therefore, MTS permits the user to enter commands (usually SET commands) to establish the conditions desired each time he or she signs on. However, since this procedure would be repetitive and wasteful, the concept of a *sigfile* or signon file was developed. A sigfile is an ordinary file which the user specifies (via SET SIGFILE=filename) as a source of commands to be executed immediately after signon, but before the system requests additional commands from the user's terminal (or batch job). The commands in the sigfile are not printed on the user's terminal (or in the batch output). For example, if a user wishes to establish a default local time limit of 10 seconds and wishes to have the cost used for processing each command printed upon completion of every command, he would construct a file containing the line

```
        SET TIME=10 COST=ON
```

and then specify the file as the sigfile. Assuming that the name of the sigfile is SIGF, the user would enter the command

```
        SET SIGFILE=SIGF
```

in MTS command mode. The name of the sigfile is saved in the accounting record when the user signs off. Thereafter, each time the user signs on (in batch or terminal mode) the sigfile commands are executed immediately, before additional commands are executed.

Sigfiles may also contain other commands: e.g., RUN commands to execute programs or CONTROL commands to establish other desired conditions such as setting terminal device commands. For example, the command

```
        CONTROL *MSINK* LEN=255
```

may be used to execute a terminal device command to set the output-line length for a terminal to be 255 characters.

## PROJECT SIGFILES

If the userID is under the control of a project director who is authorized to use the ACCOUNTING program with the project, an additional sigfile may be established. This sigfile may be assigned only by the project director and is called the *project sigfile*. The name does not imply that all userIDs belonging to the project are controlled by that project's sigfile. Instead, the project director may assign a project sigfile to any (or all) userIDs in the project, and may assign different project sigfiles to different userIDs in the project. If a userID has both a project sigfile and a user's sigfile, the project sigfile is processed first after each SIGNON and the user's sigfile is second. For example, if an instructor assigns a problem to be done at a terminal with a limit of 2 seconds of CPU time to execute, the student could construct a file containing the single line

```
        SET TIME=2
```

and then use the ACCOUNTING MANAGEMENT command to establish this file as the project sigfile for every userID in the project. Thereafter, whenever a student in the class signs on, the default local time limit will be 2 seconds. If a student forgets to put a local time limit on a RUN command and the program goes into a loop, it will be interrupted when 2 seconds of CPU time have been used. A local time limit so established by the project sigfile may help prevent the waste of computer resources that might otherwise occur. A user sigfile can override or reset this.

## SIGFILES AND SECURITY

Project sigfiles and/or user sigfiles may be used to provide additional security and control over the use of computer resources.

Better security may be achieved by requiring extended testing or validation, over and above the password, before a user is allowed to proceed. To further validate the signon, the user can prepare a program that requires a correct response to a question or command, such as

```
Enter the project secret code.
```

If the user responds correctly, thereby demonstrating that he or she is an authorized user of the userID, the validation program permits the user to proceed. If the response is unacceptable the program can terminate the job (e.g., either by calling the QUIT subroutine or by using the OFFBIT (item 23) of the CUINFO subroutine as described in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003). The user then constructs a file named, e.g., SECURITY, containing the line

```
RUN program
```

and enters SET SIGFILE=SECURITY to establish this file as the sigfile. Thereafter, anyone attempting to use the userID must respond to the questions of the program. If the user responds correctly, the session is allowed to continue; otherwise, the user is automatically signed off. An example of such a program is contained in the file *CKID (see *MTS Volume 2: Public File Descriptions*, Reference R1002, for details).

This scheme for providing extra security could be circumvented if someone were able to interrupt execution of the sigfile program, via an attention interrupt. To prevent this, the owner of the userID or project director can instruct the system to ignore attention interrupts until sigfile processing is complete. The owner may do this by entering the command

```
SET SIGFILEATTN=OFF
```

The project director may do so by entering the ACCOUNTING MANAGEMENT command and then entering

```
MOD userid,SFATTN=OFF
```

The project director has sole control over the attention switch for the project sigfile and the owner has sole control over the attention switch for the owner's sigfile. Programs that are to be executed during sigfile processing should be debugged thoroughly before sigfile attention interrupts are disabled.

The keyword SIGFILE=OFF may be specified on the SIGNON command to bypass any project or user sigfile processing unless this has been prohibited by the SIGFILEATTN or SFATTN options (see above). For example,

```
SIGNON WXYZ SIGFILE=OFF
```

If an authorized user cannot get past the sigfile-processing due to an error in a sigfile program, the owner of the userID may request the ITD Accounts Office to modify the accounting record so that attention interrupts will be recognized rather than ignored. (Such a request is submitted exactly like a request for a change of password.) The owner may then SIGNON, interrupt the faulty sigfile program, and then debug it.

Additional security for executing a sigfile program can be obtained by having the program immediately call the CUINFO subroutine to enable the OFFBIT, process the security algorithm, and then call the CUINFO subroutine to disable the OFFBIT immediately before program termination. This will cause the user to be signed off if the sigfile program fails to execute to completion due to any type of interrupt (e.g., attention interrupt, program interruption, timer interrupt).

Occasionally it is desirable to restrict the use of a userID to the running (execution) of a single program. Such a program should be invoked by a RUN command in the sigfile and should call the QUIT subroutine to sign the user off when the program terminates. Thus, the userID can never be used to run other programs; it never leaves "sigfile mode." For this type of use, the sigfile should belong to a different userID; otherwise there is no way to change it after it is established. Thus, if the userID to be restricted to "sigfile mode" is AAAA and the other userID is BBBB, the file, say SFYLE, would be created by BBBB and filled appropriately, and then permitted so that AAAA has read access. Then AAAA can sign on and enter the following commands:

```
SET SIGFILE=BBBB:SFYLE
SET SIGFILEATTN=OFF
```

This mechanism may be used to allow a large number of people to use a common userID for executing the sigfile program (e.g., a tutorial routine) while preventing them from using the userID for any other purpose.

# BATCH USE OF MTS

This section describes how to use the MTS batch job facility. It covers general concepts, and presents some basic examples for running a batch job using *BATCH*. This is followed by an explanation of the available sign-on parameters, printed output, and by other useful information.

## WHAT IS A BATCH JOB?

A batch job is simply a set of commands that are processed by the computer without interactive control. In other words, once you have submitted the job you cannot change the sequence of flow, nor correct any errors during execution. You sacrifice the flexibility available in *interactive mode*, and you may have to wait several hours, or even overnight, to receive your results.

MTS batch jobs may be

(1) punched on cards and read through a card reader,

(2) prepared at a terminal and copied to *BATCH*, or

(3) submitted over the UMnet/Michnet Computer Network.

Before deciding to use batch you should ask yourself the following questions.

(1) Do I know the exact steps required to complete the job?

(2) Can I run the program without being prompted for input from the terminal keyboard?

(3) Am I confident that the job will run without errors? Often batch jobs will exceed time or page limits (see "Common Errors with Batch" in this section).

(4) Am I prepared to wait (possibly overnight) to get the results back from my job?

If the answer is "yes" to the above questions, then you are ready to use batch.

### Reasons for Using Batch

There are two basic advantages for you in using batch mode: the first is cost, the second is time.

When you run your job interactively, you are in effect requesting the use of computer resources: for example, printers or disk space. Interactive mode means that the computer attempts to respond to your request as quickly as possible while at the same time responding to similar requests from all other users on the system. Batch mode allows the computer to take its time in responding and allocate resources to your job when they are not required by interactive jobs. What this means for you is perhaps a longer wait for completion of your job, but at a reduced cost. This reduced cost is due to the fact that you are not charged for terminal connect time and that your job may run at a time when the rates are lower. A more detailed explanation of the charges for batch processing is given in "Job Selection" in this section.

The second advantage in using batch is that under certain circumstances batch allows you more efficient use of your time. This is best illustrated by an example. Let us assume you have a job that typically consumes a lot of elapsed time (for example, running *RESTORE to reinstate several files). Rather than sit idly at a terminal waiting for execution to terminate, you might submit a batch job and get on with other work while it is being processed. If you require a *hardcopy* record of your run, this is supplied automatically in batch, and includes such pertinent information as time and accounting charges.

In addition to the above situations, any job that is processed at regular intervals in a standard manner is a good candidate for batch processing. These types of jobs are frequently referred to as "production jobs".

Remember that while you are waiting for your batch job to complete, you are free to continue working on other tasks, such as editing or submitting jobs, provided they do not use files required by your batch job.

## SUBMITTING A BATCH JOB

MTS batch jobs may be

(1) prepared at a terminal and copied to *BATCH*,

(2) submitted over the UMnet/Michnet Computer Network, or

(3) punched on cards and read through a card reader.

The easiest way to submit a job to batch is to copy the commands directly to *BATCH*.

Let us assume you were producing a course catalog, using a program in the file CATMAKER to generate your output, and you want to run your job in batch because it will cost less. The course catalog is in an MTS file called CATALOG. You can submit your batch job directly from *SOURCE* (normally the terminal keyboard) to the pseudodevice *BATCH* as follows (remember, you should not type the prompt characters):

```
# $COPY *SOURCE* *BATCH*
> $SIGNON * PAGES=100
  *BATCH* assigned job number 273907
> $RUN CATMAKER INPUT=CATALOG PRINT=*PRINT*
> $SIGNOFF
> $ENDFILE
# *BATCH* RM273907 released HOST=UM RATE=NORMAL
```

When you enter the line containing your $SIGNON command, you should always use an asterisk (*) instead of your userID if you are running the batch job from the same userID. This avoids the security risk of typing your password at your terminal. The PAGES=100 parameter requests a limit of 100 pages for your job (the default limit is 50). You may then type in any commands you wish to execute in batch; e.g., running CATMAKER.

After entering the first line, the system assigns your batch job a name and number. Your job name is RM273907 and your job number is 273907. Note that your job name is simply the job number with the letters RM prefixed to it. You can assign your own job names (more about this in "Assigning a Job Name" in this section). You can use your job name to keep track of the progress of your batch job (more about this in “Monitoring a Batch Job” in this section).

You terminate the batch job by typing the line $ENDFILE or by entering an end-of-file at the “>” $COPY prompt. Your job will then be released.

All batch jobs produce printed output which is sent directly to *PRINT*. Since you did not change the default for *PRINT*, the output from the job will be printed on one of the Xerox 9700 page printers. Details on printed output will be covered in “Printed Output for Batch Jobs” in this section.

## Entering Commands in a File

If you expect to run the job several times, you can put the commands in an MTS file and copy the file to *BATCH*.

Using your previous example, create a file called CAT.RUN and use the editor to add commands to the file.

```
# $EDIT CAT.RUN
: INSERT
? $SIGNON * PAGES=100
? $RUN CATMAKER INPUT=CATALOG PRINT=*PRINT*
? $SIGNOFF
: STOP
```

Now, any time you want to submit this as a batch job, just type:

```
$COPY CAT.RUN *BATCH*
```

Your job will be submitted to the batch queue. In general, we recommend that you use this second technique for submitting batch jobs for two reasons: first, it is easier to correct errors; second, if you are submitting the same job several times, it will be more efficient than typing in the entire command sequence each time.

## How Long Does Batch Take?

Often a batch job will begin executing immediately after it is submitted; however, this is not always the case. The time required for a job to complete is the *turn-around time*. Turn-around is affected by a number of factors including the number of users on the system, current jobs executing or in the queue, availability of resources such as printers, and the priority level of the job submitted. Because of this the turn-around for a particular job can vary significantly. The following are a few guidelines to make note of:

(1) A job that is submitted with the parameter RATE=MINIMUM on your $SIGNON command will usually run overnight (more details in “Job Selection” in this section).

(2) Jobs requiring large amounts of computer time or resources will, in general, have a slower turn-around time.

(3) Before submitting your job, it is possible to check current activity on the system using the $SYSTEMSTATUS command (see the section “Managing Batch and Print Jobs”).

(4) The system will *not* notify you when your job has finished; however, you can check using

the $SYSTEMSTATUS command. This will be explained in the next section.

## Monitoring a Batch Job

Once you have submitted your batch job, you may check its progress.

Assume you have submitted two jobs to *BATCH* and want to see if they are finished. You do not know what their names or numbers are so you issue the command

```
$SYSTEMSTATUS QUEUE *
```

or more simply

```
$SY Q *
```

The system will respond with the following:

```
1ABC:RM316636 is awaiting execution, P7, after 26 jobs.
1ABC:RM316638 was executed at 10:48:52 Sun Jan 14/90
1ABC:RM316639 is awaiting print, P13, after 21 jobs.
```

The first line refers to a batch job still awaiting execution; there are 26 jobs preceding it in the queue. It has an execution priority of 7 (indicated by P7), which is a medium priority. The second entry tells us that job RM316638 has been executed. The last entry refers to the printed output created by the second job which is waiting to be printed, there are 21 jobs ahead of it in the print queue. Note that $SYSTEMSTATUS QUEUE * will display all of your jobs currently waiting in a queue plus any jobs that have completed execution in the last eight hours.

Another form of the $SYSTEMSTATUS command which can be used as follows:

```
$SYSTEMSTATUS QUEUE USER          (or $SY Q U)
```

This form of the command will only display those jobs belonging to the user that are currently waiting to be executed. You can use this form if you are certain that your job has not been executed.

## Logging Batch Output

Once your job has been executed, the output will be sent to *PRINT*. Further details on printed output are given in the section “Printed Output for Batch Jobs” in this section.

Normally output from a batch job is printed. However, if you also want the output of your batch job to be put into a permanent file, then you should include the following commands at the start of your batch job:

```
$SIGNON *
$EMPTY MYLOG
$LOG *MSINK* ON MYLOG
     .
     .    (enter batch commands here)
     .
$SIGNOFF
```

Note that you will need to have created the file MYLOG before you submit the above job; the results of your batch run will then be placed into the file MYLOG.

## Viewing Batch Output

You may use the $VIEW command to look at the command lines of a batch job before it is executed or at the results of the job before they are printed (see the section "Managing Batch and Print Jobs" later in this volume).

## Cancelling a Batch Job

You can cancel a job submitted to *BATCH* using $CANCEL provided, of course, that the job has not been already executed. For example, suppose you have submitted a job with a low priority and subsequently decide that you want to submit it interactively rather than wait for the results. First, obtain the job name:

```
$SYSTEMSTATUS QUEUE *
```

The system responds,

```
1ABC:RM316636 is awaiting execution, P8, after 26 jobs.
```

Then you enter,

```
$CANCEL RM316636
```

You can then check again,

```
$SYSTEMSTATUS QUEUE *
1ABC:RM316636 was cancelled at 15:53:02 Sun Jan 14/90.
```

Further details on using $CANCEL are given in the section "Managing Batch and Print Jobs".

## Common Errors with Batch

A batch job will terminate abnormally if it requires more central processing unit (CPU) time or pages than you have allowed. In this case, you will find the message:

```
GLOBAL item LIMIT EXCEEDED
```

on the last page of your output, where *item* could be one of either TIME or PAGES. When you submit a job to batch there are a number of optional parameters that you can specify using $SIGNON. If you do not specify these, the system will automatically assign default values for you: 3 seconds for execution (CPU) time and 50 pages for printed output. If you expect to exceed these limits, be sure to specify appropriate values with the $SIGNON command. It is wise to estimate higher than you expect.

For example, if you are submitting a job that requires about 8 seconds of CPU time and normally prints 376 pages, you could define reasonable limits such as:

```
$SIGNON * PAGES=400 TIME=10
```

These are slightly higher than your requirements to allow for a small margin of error, but not so high that the priority of the job would be adversely affected.

There are two reasons why it is not a good idea to request an excess of either time or pages to ensure that your job will execute. First, time and page restrictions are an insurance to you that your job is not out of control producing unwanted pages of output while using up precious computer dollars; second, excessive time limits will lower your job's position in the execution queue and thus increase the turn-around time and may even cause the job to be held by the operator and not run. For more details on estimating TIME and PAGE requirements, see "Estimating Time and Pages" in this section.

Regardless of the limits you define on your $SIGNON command, you will only be charged for the amount of resources actually used. However, if you request a limit that would exceed the funds remaining in your account—even if your job does not actually require that much—the system will *not let your job sign on*. The job will abort.

Typing and syntax errors, as mentioned earlier, are also quite common. These are more easily corrected when batch jobs are submitted by entering your commands into a file. Note also that although the use of the dollar sign is optional (i.e., $SIGNON and SIGNON are equivalent), we recommend that you always use them. If an error is detected during execution, MTS will look for the next line in your file beginning with a "$" and continue from there.

## CHANGING BATCH DEFAULTS

Unless you explicitly modify the attributes of a batch job, it will execute subject to the following criteria:

(1) The job is executed at normal priority.

(2) It is printed in landscape format on the Xerox 9700 page printer, on standard 8 1/2 by 11 paper.

(3) There is a 3-second global limit for execution (CPU) time and a 50-page global limit for printed output.

(4) All commands and system responses are echoed in the the printout.

You can change these default options by using the $SIGNON command. The first line of your batch job contains a $SIGNON command with either a userID or an asterisk (*). The general form of this command is:

```
$SIGNON * parameters
```

or

```
$SIGNON userid parameters
```

The first form, which uses the asterisk, can only be used when you are submitting a job from your own userID. If you are submitting a job for a userID other than the one you are currently signed on to, then you must use the second form. Note that if you are using the second form, you will also need to provide the password for the userID that you are using. Generally, most users will find the first form easier to use. In both cases parameters are optional and can be one or more selected from the list in Table 2. The following is a simple example:

```
$SIGNON * RATE=MINIMUM PAGES=200
```

In the above example you will be signed on to run a job with minimum rates (RATE=MINIMUM) with a limit of 200 pages of printed output (PAGES=200). The order in which $SIGNON parameters are specified is not important. Note that there should be no blank spaces before or after the "=" sign.

The $SIGNON command is described more fully in the section "MTS Command Mode" in this volume. The remainder of this section discusses some of the $SIGNON parameters that are of particular interest to batch users.

## Job Selection and Rates

Both the job priority and the rate period are used to select jobs for execution and print processing.

The priority of the job is the position of the job within the execution queue or the print queue.

The job execution priority is assigned as an integer between 1 and 10 and is based on the CPU time estimate specified on the $SIGNON command. The job print priority is an integer between 0 and 15 and is based on the *actual* number of pages (images) to be printed. The following table summarizes the priority assignments for the NORMAL rate period:

**Table 1: Priority Assignments for Jobs**

| *Priority Class* | *Execution Time (seconds)* | *Images Printed* |
|---|---|---|
| 15 | ... | ≤5 |
| 14 | ... | ≤10 |
| 13 | ... | ≤20 |
| 12 | ... | ≤50 |
| 11 | ... | ≤100 |
| 10 | ≤1 | ≤200 |
| 9 | ≤2 | ≤350 |
| 8 | ≤4 | ≤500 |
| 7 | ≤8 | ≤650 |
| 6 | ≤16 | ≤800 |
| 5 | ≤32 | ≤1000 |
| 4 | ≤64 | ≤1500 |
| 3 | ≤128 | ≤2000 |
| 2 | ≤256 | ≤4000 |
| 1 | >256 | ≤6000 |
| 0 | ... | >6000 |

Jobs are selected in decreasing order of priority class and, within any given class, in a first-in, first-out (FIFO) basis. The page estimate does not affect the priority calculations for that job.

The rate period specifies the rate at which the job will be charged. Jobs for which RATE=LOW, RATE=DEFERRED, or RATE=MINIMUM is specified are automatically held for execution during the low-, deferred-, or minimum-rate periods. Such jobs are charged at a reduced rate. Jobs for which RATE=NORMAL is specified are eligible for execution at any time.

The rate period schedule is as follows:

| | |
|---|---|
| Low Rates (45%) | 6 pm - 10 pm Monday through Thursday |
| Deferred Rates (65%) | 10 pm - 12 am Monday through Thursday |
| | 12 am - 2 am Tuesday through Friday |
| | 6 pm - 10 pm Friday |
| | 7 am - 6 pm Saturday and Sunday |
| Minimum Rates (80%) | 2 am - 7 am Tuesday through Friday |
| | 12 am - 7 am Friday to Saturday |
| | 6 pm - 7 am Saturday to Sunday |
| | 6 pm - 7 am Sunday to Monday |
| Normal Rates | All other times |

The percentages given above reflect the CPU-rate discount that is applied to sessions run during these periods. Other resources may have differing discounts.

Terminal jobs are always executed at the lowest available rate at sign-on time. The rates for terminal jobs are determined by the sign-on time and remain in effect for the duration of the session. The rate for a terminal job may not be changed by specifying the RATE option on the $SIGNON command.

Batch jobs are executed at the current rate at sign-on time if the RATE specification is omitted from the $SIGNON command. A higher or lower rate period may be specified by using the RATE option. If a higher rate period is specified, the job will be executed before other batch jobs of lower rate periods. If a lower rate period is specified, the job will be held until the specified rate period is in effect. However, if the job cannot be executed during the next available lower rate period, it will be executed as soon as possible thereafter (i.e., it will not be held over until the lower rate period recurs).

Batch jobs and *PRINT* jobs submitted at LOW, DEFERRED, or MINIMUM rates may be printed anytime, but are assigned priorities lower than would be the case for similar-sized, NORMAL-rate jobs. The priority decrements are as follows:

| | |
|---|---|
| LOW-rate jobs | 2 lower than NORMAL |
| DEFERRED-rate jobs | 3 lower than NORMAL |
| MINIMUM-rate jobs | 4 lower than NORMAL |

For example, a 50-page job would be priority 12 for NORMAL rates, 10 for LOW rates, 9 for DEFERRED rates, and 8 for MINIMUM rates.

The public file *RATES gives the current rate structure for each of the different classes of accounts. You can obtain this information by issuing the command

```
$COPY *RATES
```

## Estimating Time and Pages

The amount of time and pages required is tied closely to the type of job being run; however, there are a few general guidelines to follow when making your estimates. If you have run the job previously using either batch or interactive mode, the last page of your listing will usually have an accounting of the pages printed and time used—you should refer to this for approximations.

If you are currently running a job interactively, you can obtain an estimate of the CPU time required by noting the CPU time used which is given on the “Execution terminated” message.

Note the following when you are estimating pages: the Resource Manager counts a single sheet, printed on both sides, as two pages of output. If you are using a format that prints two images on a page, then a single sheet would contain 4 pages of output. Also, remember to include all of the hardcopy that you are directing to *PRINT* in your estimates.

If you are submitting a large job which you anticipate could run for several minutes (or even hours), then do some smaller test runs first to get an accurate estimate. Finally, remember that if a job aborts due to exceeding time or page limits, *you will be charged for resources used*. Therefore, if you do not know what to estimate, get some advice from the ITD Consultants (764–HELP) instead of submitting a job several times on a trial and error basis using up computer dollars in the process.

## Assigning a Job Name

You can uniquely identify your batch job by assigning a job name by issuing a command of the form

```
$SIGNON * JOBNAME=jobname
```

The job name is a string of up to eight alphanumeric characters, starting with a letter; for example, ANDY or EPA001. This is useful if you have jobs that you submit on a regular basis. It also makes it easier to remember your job name when you are using the $SYSTEMSTATUS command.

You must use care when assigning names to jobs that are submitted on the UB-MTS system. All printed output from those jobs is assigned a new job number when it is transferred to the UM-MTS system for printing. Thus if you have several jobs with the same name, it may be difficult (or impossible) to distinguish those jobs since you will not know the new job number assigned after the transfer.

## Mounting Magnetic Tapes

If your batch job requires magnetic tapes to be mounted, use the TAPES option on the $SIGNON to improve the chances that sufficient tape drives will be available when your job starts:

```
$SIGNON * TAPES=n
```

Howvever, even with this, your job may start executing when all the drives are in use. In this event, the operator must stop your job and rerun it later, which could have undesired results (see “Reruning Batch Jobs” in this section). Also, you should use a single $MOUNT command to mount all tapes that must be simultaneously available.

## Setting Print Routes and Delivery Codes

The ROUTE option determines where your output is printed; the DELIVERY option implies that your job will be delivered to another site on campus. Either of these options can be altered with your $SIGNON command. A more complete explanation is given in the section “Getting Printed Output”.

**Table 2: Options Available on $SIGNON**

| *Option* | *Description* |
|---|---|
| ADDRESS="line1;line2;..." | Specify the campus mail address for delivered output (when DELIVERY=MAIL is set). This option applies only to page-printer output, not to line-printer or local-printer output. |
| CARDS=n | Specify maximum number of punched cards for job. The default is n=0. |
| COPIES=n | Specify number of copies of printed output. The default is n=1. |
| DELIVERY={station \| MAIL \| NONE} | Specify a delivery station to which output should be delivered by messenger service (see tee file *DELIVERY for further details). If MAIL is specified, the output will be delivered by campus mail or the US Postal Service. The default is NONE. |
| FORMAT={LANDSCAPE \| PORTRAIT \| TWOUP \| format-name} | Specify format for page-printer output. The default is LANDSCAPE. |
| JOBNAME={jobname \| DEFAULT} | Assign a job name of 1 to 8 alphanumeric characters to the current job. The first character must be a letter. DEFAULT specifies the default format of "RM" plus six digits. |
| {LANDSCAPE \| PORTRAIT \| TWOUP} | Specify orientation of page-printer output (synonymous with FORMAT for the corresponding values). The default is LANDSCAPE. |
| MARGIN={n.nn \| NO} | Set the left margin to "n.nn" inches. MARGIN=NO resets the margins to the default for the current format (0.5 for PORTRAIT and 0.65 for LANDSCAPE). |
| NOMESSAGES or NOMSGS | Suppress check of mailbox for messages. |
| {ONESIDED \| TWOSIDED} | Specify printing on one or both sides of the paper for page-printer output. The default is TWOSIDED. |
| OVERLAY={NONE \| SHADED \| LINED} | Specify an overlay for page-printer output. The default is NONE. |
| PAGES=n | Specify maximum number of printed pages for job. The default is n=50. |
| PAPER={PLAIN \| 3HOLE \| LABEL24 \| LABEL33} | Specify paper type for page-printer output. The default is PLAIN. |

| | |
|---|---|
| PLOTTIME={t \| tM} | Specify plot time limit. The default is t=0. |
| PRINT={T3 \| TN} | Specify line-printer character set. The default is T3. |
| PRINTER={PAGE \| LINE} | Specify the type of printer. The default is PAGE. |
| QUICK | Combination of NOMESSAGES and SHORT. |
| RATE={NORMAL \| LOW \| DEFERRED \| MINIMUM} | Specify job rate group for execution. |
| RERUN={YES \| NO} | Specify whether job should be restarted if it is stopped during execution for any reason. The default is YES. |
| ROUTE=station | Specify a station for output. |
| SEPCOPY={YES \| NO} | Specify whether each copy will have separate head and tail sheets (YES) or whether all copies will be printed together as one job with a single head and tail sheet (NO). The option is only effective for page-printer output and if COPIES=n is specified. If each copy is more than 50 pages, then SEPCOPY=YES is forced. The default is NO. |
| SHIFT={YES \| NO} | Specify whether page-printer output is shifted away from the binding edge. The default is SHIFT=NO. |
| {SHORT \| LONG} | Specify condensed signon message. The default is LONG. |
| SIGFILE={ON \| OFF} | Specify whether sigfile processing is to be in effect. The default is ON. |
| TAPES=n | Specify number of tape drives required at any one time. |
| TIME=n | Specify maximum CPU time in seconds. The default is 3. |
| TWOSIDED={YES \| NO} | Specify printing on one or both sides of the paper for page-printer output. The default is YES. |
| WAITUNTIL="time and/or date" | Specify when job will be run. |
| "comment" | An arbitrary comment, enclosed within quotes, may be used to identify the job. |

## PRINTED OUTPUT FOR BATCH JOBS

Every batch job creates printed output. In some cases this output may only include a head sheet, a tail sheet, and job statistics. Typically, the output consists of the above plus a log of the batch session. This log contains all of the commands that were submitted to *BATCH* plus the system responses to these commands. By default, all printed output from the job is directed to *PRINT*.

### Head Sheet, Tail Sheet, and Job Statistics

The first and last pages of the printed job are called the *head sheet* and *tail sheet*. These are used to identify each job, and to separate it from those that printed immediately before and after. The head sheet is printed in very large type and includes: your userID, the job name and number, the time and date it printed, and the delivery code. The tail sheet contains the same information in a much smaller format and also includes a summary of job statistics such as: CPU time, disk storage, lines read, pages printed, and so on for the current job. It also lists accounting charges for each resource used by the job, and gives an approximation of the remaining balance in your account.

Suppose you submitted a job by using *BATCH* as follows:

```
# $COPY *SOURCE* *BATCH*
> $SIGNON * PAGES=100
  *BATCH* assigned job number 629384
> $RUN CATMAKER INPUT=CATALOG PRINT=MYTEXT SERCOM=MYLIST
> $SIGNOFF
> $ENDFILE
# *BATCH* RM629384 released HOST=UM RATE=NORMAL
```

This is what you might expect your job statistics to look like:

```
ID:                         1ABC
Project Number:             1AAA
Signon Time:                11:57:06 Tue Jan 16/90
Signoff Time:               11:57:33 Tue Jan 16/90
Last Signon Time/Date:      08:13:12 Mon Jan 15/90
  Charging Rate:      Batch,Normal,Univ/Gov't,UM,IBM 3090-600E

Elapsed Time          0.067 minutes
Cpu Time              0.726 seconds              $.50
Active VM Integral    0.633 page-minutes
Wait VM Integral      0.128 page-hours
Lines Read                3 lines
Page Printer Pages        3 pages
Page Printer Lines      125 pages
Page Printer Image        3 images               $.12
Page Printer Sheet        2 sheets               $.02
Disk I/O                394
     Computing Cost                              $.64

Disk Storage           5760 page-hours           $.30
Remaining Balance                             $927.04

Job name:                   RM629384
Job number:                 629384
Host:                       UM
Devicetype:                 9700
Twosided:                   YES
Format:                     LANDSCAPE
Paper:                      PLAIN
```

```
        Entered from AF1C at:        11:57:07 Tue Jan 16/90
        Printed on PTR2 at:          12:26:59 Tue Jan 16/90
```

The above example tells you a number of things about your job. The job ran at 11:57 and was charged at normal rates. The total elapsed time was .067 minutes whereas it only used .726 seconds of CPU time. Elapsed time is always higher than CPU time due to the fact that even when a job is executing it will spend a certain amount of time waiting for resources, disk I/O, etc. The job read 3 lines; these are the 3 lines that were submitted to *BATCH* from your terminal. There were 3 images printed on 2 sheets of paper. All of the other output was put into permanent files, as indicated by the parameters PRINT=MYTEXT and SERCOM=MYLIST from the original job. If you had directed all of your output to *PRINT*, you would then expect the lines printed to be much higher. The job was finished at 11:57 and was printed at 12:26. The total cost for the job was $.99. If you had printed the files MYTEXT and MYLIST, there would have been additional charges.

## Changing Print Options

There are a number of techniques for changing your printed output. You can alter the print options for your entire batch job on the $SIGNON command (see Table 2 for details). This is typically done for setting global options; for example, you can set the delivery station using the DELIVERY option.

The other techniques that you will probably find useful is creating multiple print streams within your batch job. This will allow you to print several files each with different print options within a single batch job. A full explanation of creating user defined pseudodevices (multiple print streams) is provided in the section “Getting Printed Output”. The following is a simple example to illustrate how you might use this technique:

```
$SIGNON * PAGES=200
$CREATE *LIST* TYPE=PRINT OPEN='FORMAT=TWOUP'
$CREATE *TEXT* TYPE=PRINT OPEN='PORTRAIT ONESIDED'
$RUN CATMAKER INPUT=CATALOG PRINT=*TEXT* SERCOM=*LIST*
$SIGNOFF
```

This batch job consists of three separate print jobs. The first job is the log of your batch run which is directed to *PRINT*. The second print job is your program listing which will be printed in two-up format; that is, there will be two pages printed on each side of the sheet. Finally, the third print job is the output from your program run which will be printed on one side of the page (ONESIDED) in portrait orientation (PORTRAIT).

All of the options that are available for regular print jobs also apply to batch print jobs. Further details and examples of altering the appearance of your printed output are provided in the section “Getting Printed Output”.

## Cancelling Hardcopy Output

You can eliminate hardcopy output from your batch job by using the $CANCEL command. For example, suppose you want to run the previous job with the listing and CATMAKER output placed into permanent files. The only printed output would be the job statistics which you could then cancel as follows:

```
        # $COPY *SOURCE* *BATCH*
        > $SIGNON * PAGES=100
          *BATCH* assigned job number 629384
        > $RUN CATMAKER INPUT=CATALOG PRINT=MYTEXT SERCOM=MYLIST
        > $CANCEL *PRINT*
        > $SIGNOFF
        > $ENDFILE
        # *BATCH* RM629384 released HOST=UM RATE=NORMAL
```

You should use this form of $CANCEL carefully, as it will eliminate *all* printed output from your job including error messages.

## OTHER THINGS YOU SHOULD KNOW

Let us quickly look at a few more features which might be used to your advantage. These include:

(1) rerunning batch jobs

(2) conditional sign-offs

(3) sigfile processing

### Rerunning Batch Jobs

If there is a system failure (or a system shutdown where the operator decides to stop your job), then all jobs that are currently executing will be rerun *from the beginning*. This could present a serious problem if you had already emptied or destroyed input files that were used during an earlier part of your job. If you are adding data to the end of a file, the same information could be added twice, which may not be the desired effect.

Your job will also be automatically rerun if it requires tape drives that were not available while it was running. This problem is less likely to occur if you specify the TAPES option on the $SIGNON command and mount all tapes using a single $MOUNT command. Since tape mounts are not allowed on the UB-MTS system between 10 a.m. and 4 p.m., jobs submitted during that time requiring tape mounts should specify the WAITUNTIL option on the $SIGNON command.

In order to protect yourself against possible problems in the event of a system failure, design your batch job so that it can be rerun from the beginning. If you create a file within the job, empty it right after creation so that it will not contain anything if the job is rerun.

When a job is rerun, whatever was done before the interruption is *not* undone when it restarts. You are not explicitly informed when a rerun has occurred, although you might find some clues in your batch job listing. For example, if you get the following system response while trying to create a new file:

```
        File NEWFILE already exists
```

it is probable that your job was rerun.

One consolation is that you are only charged for the final complete run, and not for the aborted first attempt. If you do not want your job to be rerun, you have the option of specifying RERUN=NO with your $SIGNON command.

## Conditional Sign-off

In a batch job, all commands are processed sequentially regardless of previous errors. However, you can use the $IF command after any $RUN command to test for an error condition. Then, depending on the value of the *return code* set by the program, you can decide whether to sign off or continue. This does not offer as much flexibility as interactive computing, but it does let you terminate a job that is running into trouble.

The following is a simple example:

```
$IF RUNRC > 4, $SIGNOFF
```

The return code from the last executed program is tested. If it is greater than 4, the user will be signed off the system. Thus, if a program gives an indication of success or failure through values of the return code, this type of test may be used to abort a batch job if the program fails to execute properly.

You can also conditionally $RUN another program based on the success/failure of previous steps. A complete conditional facility is available by using MTS command macros which are described in *MTS Volume 21: MTS Command Extensions and Macros*, Reference R1021.

## Sigfile Processing

In MTS, it is possible to set up a special file called a *sigfile* that will contain a series of commands to be automatically processed whenever you sign on. You can check to see if you have a sigfile by issuing the command

```
$DISPLAY SIGFILE
```

If MTS responds with the name of a file that is your current sigfile. You should print out a copy of this file and familiarize yourself with its contents. If you do not already have a sigfile, the rest of this section will explain briefly how you can set one up.

The following steps will create a sigfile called SIGS:

```
# $CREATE SIGS
# $EDIT SIGS
: INSERT
? $SET ROUTE=NUBS
? $SET COST=ON
?
```

This file will route all your printed output to NUBS and will also turn on the MTS cost display option. The next step is to identify this file as your sigfile by issuing the following command:

```
$SET SIGFILE=SIGS
```

The next time you sign on to your userID (and every time thereafter), the two commands in SIGS will automatically be executed. If you decide that you want to *turn off* sigfile processing for all subsequent signons, issue the following command:

```
$SET SIGFILE=OFF
```

This is a simple example that is intended to introduce the concept of sigfile processing. It is also possible to run a program from within your sigfile which will execute a series of commands *conditionally*, depending on such factors as what type of device you are signed onto, or what day of the week it is. If you want to know more, please see the description of *SIGSETUP in *MTS Volume 2: Public File Descriptions*, Reference R1002.

# GETTING PRINTED OUTPUT

Printed output can be produced during a terminal session or as a result of submitting a batch job. The various terms *printout, hardcopy*, or simply *output* are most often used to describe the printed results. Earlier we discussed some of the attributes that apply specifically to batch jobs; the following will cover the details of creating a print job from either a terminal or batch session.

## WHAT IS *PRINT*?

In the introduction to this document we described pseudodevices and how they are used. *PRINT* refers to the pseudodevice that is used to produce your printed output. By default, *PRINT* is defined as one of the Xerox 9700 page printers.

You create printed output in a variety of ways. You can use the $LIST command:

```
$LIST filename ON *PRINT*          (the ON is optional)
```

or if the named file contains carriage-control characters, you can use $COPY:

```
$COPY filename TO *PRINT*          (the TO is optional)
```

or you can send output directly from a job that is executing, e.g.,

```
$RUN *PASCALVS INPUT=MYPROG OBJECT=MYOBJ PRINT=*PRINT*
```

In all of the above cases the Resource Manager follows the same sequence of events. It assigns the job a number and, if you do not assign one explicitly, a default job name. It evaluates the requirements of the job, and then submits it to the print queue for the designated device. Once in the queue the job will:

(1) print immediately, or

(2) print after waiting for jobs that were before it in the queue, or

(3) print when the resources are available (e.g., label stock is loaded into the Xerox 9700 page printer).

If the print device is not functioning when the job is submitted, the Resource Manager will maintain a queue of all jobs until the machine is back in order. This can sometimes cause quite a backlog when the system is busy. The operations staff have the option of holding back very large print jobs, especially if the queue is long. A print job that generates a large amount of output may be printed in smaller segments. Of course, the user picks it up as one complete job.

## WHEN TO USE $COPY AND $LIST

Some files contain special characters called carriage-control characters in the first column. These special characters act as instructions to the printer to control vertical spacing on the page. These are usually inserted by text-processing programs; however, they can be added by the user. A common mistake when getting a printout of a file is to use the $COPY command when the file does not contain carriage-control characters. The result is often that the output may be only one or two lines per page, thus generating an unexpectedly large volume of output. The clearest indication that you have this problem is when the first character of some (or all) of the lines is missing in the printed output.

The easiest way to check for carriage-control characters is to examine the file using the editor and check the first character of each line. If the first character is part of the actual text or data, *do not* use $COPY to send that file to the printer; use $LIST instead. On the other hand, if the first character of each line is a blank or a carriage-control character, the printer will use it to control spacing as indicated. Further information about carriage-control characters is given in Appendix H of the section "Files and Devices" in this volume.

The $LIST command also is preferable if you are printing files with long lines since they will be wrapped onto the next line. With the $COPY command, they will be truncated instead.

## PRINTING OPTIONS

All output is sent to the printer with certain default printing options in place. If you do nothing more than direct output to the printer, your output will have the following attributes:

(1) orientation is landscape

(2) typeface is Xerox 1200 fixed-pitch

(3) printing is on both sides

(4) printing with no overlay

(5) printing is not shifted from the "binding" edge

(6) paper is 8 1/2 by 11, standard weight, plain (not 3-hole punched)

All of these options can be altered using the $SIGNON, $SET, $CONTROL, or $9700 commands. The choice of which command to use depends on the options required, and how long you want them to be in effect. It is not practical to illustrate here all of the ways that you can change your print options. However, we will discuss each of the four commands briefly and give an example of how they might be used.

### Setting Print Options for the Entire Session

The $SIGNON command can be used to set options for an entire session in batch mode. Setting global print options for interactive mode is done using $SET. If you are interested in setting print options for batch processing, you should read the section on the $SIGNON command entitled "Changing Batch Defaults".

The $SET command is used to set print options for the remainder of the session—that is, once you have set your options they are in effect until you reset them or sign off. The main advantage of using $SET is that you have the option of putting your commands into a file called a sigfile and then have that file automatically executed each time you sign on. The following is a simple example of how you might use this technique.

Suppose you want all of your output to be printed with the page oriented in portrait (like this document) instead of in landscape (which is the default format). Assuming that you have a sigfile called MYSIGFILE, you would include the appropriate $SET command as follows:

```
# $EDIT MYSIGFILE
: INSERT
? $SET FORMAT=PORTRAIT
```

Now the next time you sign on (and for all sign-ons thereafter), all of your output will have portrait orientation. If you want to change this, you can reset the option using:

```
$SET FORMAT=LANDSCAPE
```

More details on creating and testing sigfiles is given the section “Sigfile Processing” earlier in this document. Sigfile processing is also useful for setting print routes and delivery codes (more about these in a later section).

## Setting Print Options for the Current Print Job

You can use $CONTROL to set print options for a single print job. For example, suppose you were printing a special report and you want to use paper that is lined to make it more readable. The output for your report is stored in a file called REPORT1.

```
# $CONTROL *PRINT* OVERLAY=LINED
  *PRINT* assigned job number 357578
  *PRINT* RM357579 held
# $COPY REPORT1 TO *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM357578 released to CNTR 10 images 5 sheets RATE=NORMAL
          OVERLAY=LINED
```

After you enter the $CONTROL command, the Resource Manager assigns a job number. You are also notified that your job is being *held*. What this means is that nothing will be printed until you issue $RELEASE *PRINT* or sign off. The Resource Manager displays the specified options once the job has been released.

You can also use $CONTROL to change several options at once. For example, if you want to print a file on 3-hole punched paper at Michigan Union Station (UNYN), you would enter the following:

```
# $CONTROL *PRINT* PAPER=3HOLE ROUTE=UNYN
  *PRINT* assigned job number 357595
  *PRINT* RM357595 held
# $COPY REPORT1 TO *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM357595 released to UNYN 10 images 5 sheets RATE=NORMAL
          PAPER=3HOLE ROUTE=UNYN
```

The $CONTROL command can also be used to change the format of your output. For example, to print a file using portrait orientation set with a shaded overlay:

```
# $CONTROL *PRINT* FORMAT=PORTRAIT OVERLAY=SHADED
  *PRINT* assigned job number 357596
  *PRINT* RM357596 held
# $COPY REPORT1 TO *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM357596 released to CNTR 10 images 5 sheets RATE=NORMAL
```

```
                 FORMAT=PORTRAIT OVERLAY=SHADED
```

A complete list of formats and overlays is available in *Using the Xerox 9700 Page Printer*, Reference R1038.

When you use $CONTROL be aware that each time you issue the command the options you set will affect everything that you have printed but not yet released. For this reason we suggest that you always release your print job when you are finished with a particular print option.

If a line of the form

$9700CONTROL option **...**

appears in a file that is copied to *PRINT*, then it will act as if a $CONTROL *PRINT* command with the specified options was issued. For example, if the file contains

```
$9700CONTROL ONESIDED PAPER=PLAIN
line 1
line 2
  ...
```

then copying this to *PRINT* will act the same as if you first issued the command

```
$CONTROL *PRINT* ONESIDED PAPER=PLAIN
```

and then copied the file containing the lines

```
line 1
line 2
  ...
```

to *PRINT*.

Note that only legal $CONTROL command options may be specified on the $9700CONTROL line. As with the $CONTROL command, all this does is set up the conditions that will be in effect at the *start* of the print job. If the same option is used more than once on $9700CONTROL lines in a print job, the last one specified will be the one in effect.

If do not explicitly $RELEASE your print job, it will be automatically released when you sign off. Also once you issue $RELEASE, all of your print options are changed back to their default values with the exception, of course, of those that were changed using $SET.

If you want to find out if a print job is currently being held, you can issue the command

$DISPLAY *PRINT*

The message

```
*PRINT* RM357604 routed to CNTR 20 images 10 sheets RATE=NORMAL
```

will be printed indicating that you have 10 sheets of output waiting to be printed. If you decide that you do not want your printout, you can $CANCEL it instead of issuing $RELEASE (more about this in a later section). More examples of using $CONTROL are given in the section “Managing Batch and Print Jobs”.

## Setting Print Options Using $9700 Commands

The $9700 command differs from previous commands (including the $9700CONTROL command) since it is used to change print options *within the data being printed* as opposed to changing the print options for an entire job. In this way, you can alter the printed appearance of your job on a line-by-line or a page-by-page basis.

The $9700 commands are inserted directly into the data stream instead of being issued from MTS. They will allow you to change options such as typeface, fonts, and page orientation. The use of $9700 commands is explained in detail in *Using the Xerox 9700 Page Printer*, Reference R1038. If you are interested in altering print options in this manner, you are advised to read this document, or alternatively you should consider using one of the text-processors available on MTS that automatically insert the appropriate $9700 commands.

## Summary of $SIGNON, $SET, $CONTROL, and $9700

When you submit a batch job, you can assign your print options for the entire session using the $SIGNON command. If you want a particular print option to always be in effect for either batch or interactive mode, you should create a sigfile and put in the appropriate $SET commands. If you want to specifically change the print options for a single job, the $CONTROL command is the one to use. The $9700 command is quite specialized; its use should not be attempted without further reading as noted above.

A table of $SIGNON parameters is given above in “Changing Batch Defaults”. Similar tables for $SET and $CONTROL can be found in the section “Managing Batch and Print Jobs”.

## PRINT ROUTES AND DELIVERY CODES

You can use ROUTE option to specify where you want your output printed; the DELIVERY option implies that your job will be printed at one location and then delivered to another site by messenger. Let’s look at two simple examples to illustrate the use of these options.

Suppose you are working in the Chemistry Building and you want to have your output printed at the nearby North University Building Station (NUBS). You enter your commands as follows:

```
# $SET ROUTE=NUBS
# $EMPTY MYOBJ
# $RUN *PASCALVS INPUT=MYPROG OBJECT=MYOBJ PRINT=*PRINT*
  *PRINT* assigned job number 629384
  *PRINT* RM629384 released to NUBS 18 images 9 sheets RATE=NORMAL
```

The listing produced by the compiler *PASCALVS will be printed at NUBS.

Alternatively, suppose you are working on the U-M Dearborn Campus and want the Dearborn messenger to pick up the output for you at the Computing Center. You would submit your job as follows:

```
# $SET DELIVERY=DBRN
# $EMPTY MYOBJ
# $RUN *PASCALVS INPUT=MYPROG OBJECT=MYOBJ PRINT=*PRINT*
  *PRINT* assigned job number 629384
  *PRINT* RM629384 released to CNTR 18 images 9 sheets RATE=NORMAL
          DELIVERY=DBRN
```

Note that in the above examples that the ROUTE and DELIVERY options will be in effect for the entire session (this is because you used $SET). If you want to change ROUTE or DELIVERY for a particular job, you would use $CONTROL. If you are submitting a job in batch, you can change the print route or delivery code using $SIGNON (see "Changing Batch Defaults").

A complete list of available print routes and delivery codes is maintained online; they can be obtained as follows:

```
$COPY *ROUTE
$COPY *DELIVERY
```

If the DELIVERY=MAIL option is specified, one of two things will happen:

(1) For Xerox 9700 page-printer output, if the ADDRESS option is specified on either the $CONTROL or $SET command, the output will be delivered via campus mail. For example:

```
$SET DELIVERY=MAIL ADDRESS="Jon Dough;ITD;5074 Fleming 1340"
```

(2) For line-printer output, or for page-printer output that does not specify the ADDRESS option, the output will be sent to the ITD Mail Librarian for delivery via the US Postal Service at a cost of $7.50 for each mailing in addition to actual shipping costs. The charge will be billed to your userID. To make arrangements for mail delivery, send your job number and mailing address to:

ITD Mail Librarian
535 W. William Street
Ann Arbor, MI 48103

or send an electronic message via the MTS Message System to Mail_Librarian.

Output from the local printers is never delivered, even if the DELIVERY option is specified.

## MULTIPLE COPIES

If you want to print more than one copy of your entire job (not including head and tail sheets), use the COPIES option with the $CONTROL command and specify the number desired.

```
# $CONTROL *PRINT* COPIES=3
  *PRINT* assigned job number 357615
  *PRINT* RM357615 held
# $COPY filename *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM357615 released to CNTR 40 images 20 sheets per copy
          RATE=NORMAL COPIES=3
```

If you require each copy to have a head and tail sheet (i.e., to print as a separate job), include the SEPCOPY=YES option on the $CONTROL command:

```
$CONTROL *PRINT* COPIES=3 SEPCOPY=YES
```

Note that if each copy is larger that 50 pages, then each copy will always have a head and tail sheet, since in this case each copy is separately scheduled for printing.

The SEPCOPY option applies only to page-printer output. Line-printer output and local printer output are always printed with SEPCOPY=YES in effect.

## CANCELLING A PRINT JOB

It is possible to cancel a job that you have submitted for printing if it is in the queue or is currently printing. First, get the job name and number as follows:

```
$SYSTEMSTATUS QUEUE *
```

The system responds,

```
1ABC:RM316640 (316640) is awaiting print, P8, after 10 jobs.
```

Then you enter,

```
$CANCEL RM316640
```

The system will respond with

```
 Job 1ABC:RM316640 cancelled.
```

As you can see your output was successfully cancelled. If you have not released *PRINT*, use $CANCEL as follows:

```
$CANCEL *PRINT*
```

When you use $CANCEL as shown in the last example, you will be refunded the printing portion of the original charges for your output. Further details on using $CANCEL are given in the section "Managing Batch and Print Jobs".

## CREATING YOUR OWN PRINT DEVICE

Occasionally you may find that you are changing print options frequently. If this is the case, you might find it useful to create your own print pseudodevices (the concept of pseudodevices was explained in the "Introduction") with the options that you use for particular types of print jobs. It is possible to do this using the $CREATE command.

For example, suppose you are working on a paper at the Church Street computing site (CHUR). The paper is in an MTS file called THESIS. You want to produce two types of printed output: the first is a file listing that you want printed on the Xerox 4045 local printer at the site; the second is your final text output, with portrait orientation, printed on the Xerox 9700 page printer at the Michigan Union Station (UNYN). You can do this by creating two new pseudodevices called *DRAFT* and *FINAL* as follows:

```
# $CREATE *DRAFT* TYPE=PRINT OPEN="ROUTE=CHUR"
  *DRAFT* has been created
# $CREATE *FINAL* TYPE=PRINT OPEN="FORMAT=PORTRAIT ROUTE=UNYN"
 *FINAL* has been created
# $LIST THESIS ON *DRAFT*
  *DRAFT* assigned job number 357400
  *DRAFT* RM357400 released to CHUR 12 pages
# $COPY THESIS TO *FINAL*
```

```
        *FINAL* assigned job number 357401
        *FINAL* RM357401 released to UNYN 11 images 6 sheets RATE=NORMAL
                FORMAT=PORTRAIT
```

The output from your job has been sent to the specified printers. You are also free to use *DRAFT* or *FINAL* anytime again in the current session. If you want to use *DRAFT* and *FINAL* without explicitly creating them for each session, you should use $CREATE in a sigfile to define them.

When you create print pseudodevices, you specify the attributes by using the OPEN option. Any options that are available with the $CONTROL command may also be used with OPEN. It is also possible to specify several options at once provided they are separated by blanks:

```
     $CREATE *ONE* TYPE=PRINT OPEN="FORMAT=PORTRAIT ONESIDED"
```

You have created a print device called *ONE* which has the attributes of PORTRAIT and ONESIDED.

## LOCAL PRINTERS

Several Campus Computing Sites have Xerox 4045 or Hewlett-Packard LaserJet local printers. These are small page printers which are slower, have fewer built-in fonts, and may be restricted in other capabilities such as printing on only one side of the paper. They are best for printing short file listings, draft copies of documents, and copies of messages.

The Xerox 4045 and H-P LaserJet only support the PORTRAIT, LANDSCAPE, MARGIN, and OVERLAY=LINED $CONTROL options; they do not recognize $9700 command lines.

To send output to a local printer, you must specify the ROUTE parameter on the $CONTROL or $SET command

$CONTROL *PRINT* ROUTE=station

For example, to print the file named DRAFT in portrait orientation on the Xerox 4045 at the Dana Building Computing Site:

```
     $CONTROL *PRINT* PORTRAIT ROUTE=DANA
     $COPY DRAFT *PRINT*
```

The following local printers are available at the Campus Computing Sites.

| *Route Code* | *Printer Type* | *Location* |
|---|---|---|
| AROX | Xerox 4045 | Angell Hall Courtyard |
| CHUR | Xerox 4045 | 611 Church, 4th Floor |
| DANA | Xerox 4045 | Dana Building (Natural Resources) |
| FRZE | Xerox 4045 | Frieze Building |
| SPH2 | Xerox 4045 | School of Public Health |
| UGLS | Xerox 4045 | Undergraduate Library |

Local printing to the Xerox 4045 and H-P LaserJet printers is charged for at the same rates as for the Xerox 9700 page printers.

# MANAGING BATCH AND PRINT JOBS

The Resource Manager enables you to modify certain features of any job you submit for processing. It also provides a mechanism for keeping track of jobs once they have been submitted. This section contains a summary of the MTS commands that can be used.

| | |
|---|---|
| $CANCEL | removes a job from the processing queue. |
| $CONTROL | changes the attributes of a specified pseudodevice. |
| $CREATE | lets you define pseudodevice names. |
| $DISPLAY | shows the current state of a pseudodevice or set item. |
| $LOG | puts a record of your session in a file. |
| $RELEASE | releases a job to the execution or print queue. |
| $SET | changes certain global conditions that apply to a job. |
| $SYSTEMSTATUS | keeps track of jobs that have been submitted during the last 8 hours. |
| $VIEW | lets you view the results of a batch or print job. |

Note: Most MTS commands may be abbreviated. However, for clarity abbreviations are used sparingly here.

## THE $CANCEL COMMAND

The $CANCEL command lets you remove a job from the processing queue. You can cancel a job that is not done, i.e.,

(1) executing or waiting to execute

(2) printing or waiting to print

(3) held by a $CONTROL command

The general form of the command is:

```
$CANCEL {jobname | jobnumber} [USER=userid]
```

or

```
$CANCEL {*PRINT* | *BATCH*}
```

The first format is used for jobs that have been released for processing. In this case you must know the *job name* or *job number* (these terms were explained in the "Introduction"). Your job name and job number can be obtained by using the $SYSTEMSTATUS command. If the job was submitted from a different userID, you must specify it with the USER parameter; you will be prompted for the password.

The second form is used when the job is still being held (i.e., you have not released it). In this case you must cancel the appropriate pseudodevice name (e.g., *PRINT*).

The effects of cancelling a job are as follows:

(1) If the job has already been executed and printed, cancelling it will have no effect as the job

is finished anyway.

(2) If you cancel the pseudodevice name *before* releasing the job to the processing queue, the job is cancelled and you are not charged.

(3) If you cancel a print job after it is released but before it starts printing, it will not be printed. Note: Your session charges will include the printing charges, but these will be automatically refunded at a later time.

(4) If your print job was in the middle of printing when you cancelled it, it will stop printing at that point and you will be charged for the portion that was printed.

(5) If you cancel a batch job before it starts executing, you will not be charged for it.

(6) If a batch job is executing, or has already executed but has not yet printed, cancelling the job will not cancel the printout. In order to cancel the print job you must issue a second $CANCEL. In any event you will still be charged for the job up to the point where it was cancelled.

Some typical examples:

(1) You have just copied 400 pages to the printer, and suddenly realize that it was the wrong file. The $SYSTEMSTATUS command indicates that your job (138475) is still waiting to print, so you can use:

```
$SYSTEMSTATUS 138475
1ABC:RM138475 awaiting print, P8, after 10 jobs
$CANCEL 138475
Job 1ABC:RM138475 cancelled.
```

The system response gives the user's userID (in this example, 1ABC) before the job name. The job will not be printed and you will be automatically refunded for it.

(2) You have submitted a batch job (RM112984), which has already started to execute, and then you realize that you have used the wrong format:

```
$CANCEL RM112984
Job 1XYZ:RM112984 cancelled.
$CANCEL RM112984
Job 1XYZ:RM112984 (112985) cancelled.
```

When the job began execution, it created a print stream (112985) that was not cancelled the first time. The second $CANCEL command got rid of it.

(3) After controlling *PRINT* so that it will print in portrait mode, you copy your job to it. However you do not know whether it should be printed onesided, so you decide to cancel it for now:

```
$CANCEL *PRINT*
*PRINT* RM198765 cancelled.
```

(The job name RM198765 was assigned by the $CONTROL command.) The *print* cost for images and sheets will be refunded, but you will still incur charges for CPU time used by the $LIST or $COPY command.

(4) You have submitted a batch job to run overnight and you decide you want to cancel it, but you do not know the job name and number. You obtain it as follows:

```
$SYSTEMSTATUS QUEUE *
1ABC:RM116982 awaiting execution, P10, after 35 jobs.
```

and now you enter:

```
$CANCEL 116982
Job 1ABC:RM116982 cancelled.
```

Your batch job has now been cancelled.

Notes:

(1) Use the $SYSTEMSTATUS command to find the job names and current status of any jobs you have submitted within the last 8 hours.

(2) Be sure you specify the correct job name when there is more than one waiting in the queue. If you have assigned the same job name to more than one job, cancelling that job name will cancel *all* jobs with that name. You can avoid this problem by cancelling the job number instead.

## THE $CONTROL COMMAND

The $CONTROL command lets you modify the attributes of a single print or batch job. You will notice that many of the options are identical to $SET options. The essential difference between $CONTROL and $SET is that $CONTROL changes options for a particular job—*only until the job is released or you sign off*. The $SET command will affect all jobs until the option is reset or you sign off. When you need a particular set of options for a single job, $CONTROL is the command to use. It is frequently used to change the way output is printed on the Xerox 9700 page printer.

The general form of the $CONTROL command is:

```
$CONTROL *pdn* keyword
```

or

```
$CONTROL *pdn* keyword=option
```

where **pdn** is a pseudodevice name from the list given below, and *keyword* (with *option* where applicable) is selected from Table 3 below. The following is a list of predefined pseudodevice names that are processed by the Resource Manager.

| *Pseudodevice* | *Facility Provided* |
|---|---|
| *BATCH* | For submitting a batch job. |
| *PRINT* | For printing output. |
| *PUNCH* | For producing 80-byte records for BITNET files. |
| *IMPORT* | For retrieving BITNET files that have been received from another system. |
| *EXPORT* | For sending BITNET files to another system. |

The first three pseudodevices have been described in other sections of this document (e.g., *BATCH* was explained in the section “Batch Processing”, *PRINT* was explained in the section “Getting Printed Output”, etc). Each of these sections also contains more examples of using $CONTROL for each particular pseudodevice.

More information about BITNET is available in the document *BITNET on MTS*, Reference R1039.

**Table 3: $CONTROL Keywords and Options**

| *Keyword* | *Description* |
|---|---|
| ADDRESS="line1;line2;**...**" | Specify the campus mail address for delivered output (when DELIVERY=MAIL is set). This option applies only to page-printer output, not to line-printer or local-printer output. |
| CANCEL ***...*** | Cancel the job that is currently held for the specified pseudodevice. |
| COMMENT="text" | Specify a comment that will be associated with the job. This comment will be printed on the head sheet of jobs submitted to *PRINT* or *BATCH*. |
| COPIES=n | Specify number of copies of output for the given pseudodevice. The default is 1. |
| DELIVERY={station ǀ MAIL ǀ NONE} | Specify a delivery station to which output should be delivered by messenger service (see the file *DELIVERY for further details). If MAIL is specified, the output will be delivered by campus mail or the US Postal Service. The default is NONE. |
| FORMAT={LANDSCAPE ǀ PORTRAIT ǀ TWOUP ǀ format-name} | Specify format for page-printer output. The default is LANDSCAPE. |
| HOLD | Explicitly hold the job directed to the specified pseudodevice. |
| JOBNAME={jobname ǀ DEFAULT} | Assign a job name of 1 to 8 alphanumeric characters to the job directed to the specified pseudodevice. The first character must be a letter. DEFAULT specifies the default format of “RM” plus six digits. |
| {LANDSCAPE ǀ PORTRAIT ǀ TWOUP} | Specify orientation of page-printer output (synonymous with FORMAT for the corresponding values). The default is LANDSCAPE. |
| MARGIN={n.nn ǀ NO} | Set the left margin to “n.nn” inches. “n.nn” must be less than the current page width (8.5 for portrait orientation, 11.0 for landscape). MARGIN=NO turns off the margin override and resets the margins to the default for the current format (0.5 for PORTRAIT and 0.65 for LANDSCAPE). |
| NUMBER={(b,l,c) ǀ NO} | NUMBER=(b,l,c) numbers pages automatically, starting with number “b”, |

printing the number on line “l”, ending in column “c”. NUMBER=NO (the default) disables automatic page-numbering. The page number is always printed in the first font of the current format or FONTLIST specification.

OVERLAY={NONE | SHADED | LINED}
Specify an overlay for page-printer output. The default is NONE.

PAGES=n Specify a page limit for a print job. The default is no page limit.

PAPER={PLAIN | 3HOLE | LABEL24 | LABEL33}
Specify paper type for Xerox 9700 page-printer output. The default is PLAIN. The obsolete ANY option is synonymous with PLAIN.

PRINTER={PAGE | LINE}
Specify the type of printer. The default is PAGE. Applies only to *PRINT* and *BATCH*.

RELEASE Release for processing the job that is currently held for the specified pseudodevice (see also the $RELEASE command).

SEPCOPY={YES | NO}
Specify whether each copy will have separate head and tail sheets (YES) or whether all copies will be printed together as one job with a single head and tail sheet (NO). The option is only effective for page-printer output and if COPIES=n is specified. If each copy is more than 50 pages, then SEPCOPY=YES is forced. The default is NO.

ROUTE=station Specify a station for output at which output is to be printed (use the command $COPY *ROUTE to obtain a list of valid station codes).

SHIFT={YES | NO} Specify whether page-printer output is shifted away from the binding edge. The default is SHIFT=NO.

TWOSIDED={YES | NO}
Specify printing on one or both sides of the paper for page-printer output. The default is YES.

{TWOSIDED | ONESIDED}
Specify printing on one or both sides of the paper for page-printer output. The default is TWOSIDED.

Some typical examples:

(1) The next job you send to *BATCH* is to be printed onesided on 3-hole punched paper. The file called BATRUN should contain MTS commands for a batch job, including a $SIGNON command.

```
# $CONTROL *BATCH* ONESIDED PAPER=3HOLE
  *BATCH* assigned job number 345678
  *BATCH* RM345678 held
# $COPY BATRUN *BATCH*
# $RELEASE *BATCH*
  *BATCH* RM345678 released HOST=UM ROUTE=CNTR TWOSIDED=NO
```

```
                            PAPER=3HOLE
```

The job is assigned a number and is automatically held until the $RELEASE command is given. This demonstrates how more than one attribute may be specified on a $CONTROL command.

(2) You want to direct the next printout to the Xerox 9700 page printer at the Michigan Union Station (UNYN):

```
# $CONTROL *PRINT* ROUTE=UNYN
  *PRINT* assigned job number 654321
  *PRINT* RM654321 held
# $LIST DATAFILE ON *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM654321 released UNYN 9 images 5 sheets RATE=NORMAL
```

(3) You want your output printed with portrait orientation and a shaded overlay:

```
# $CONTROL *PRINT* FORMAT=PORTRAIT OVERLAY=SHADED
  *PRINT* assigned job number 654324
  *PRINT* RM654324 held
# $COPY ESSAY TO *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM654324 released to CNTR 20 images 10 sheets
          RATE=NORMAL FORMAT=PORTRAIT OVERLAY=SHADED
```

A complete list of formats and overlays is available in *Using the Xerox 9700 Page Printer*, Reference R1038.

Notes:

(1) If a held job is not explicitly released, the keywords given on the $CONTROL command remain in effect for all output subsequently directed to that job during the current session.

(2) If you enter additional $CONTROL commands for a job that is already held, the effect is cumulative. In the event of a conflict of options for a specific keyword, the last value assigned will apply to the whole job (including any output that may have been generated before the last $CONTROL command).

(3) You can use the $CONTROL command with user-defined pseudodevice names.

(4) You can use $DISPLAY to see what is currently held for a particular job.

(5) You can use $CANCEL to cancel a job that is being held.

## THE $CREATE COMMAND

The $CREATE command is normally used for creating files, but it also lets you define your own pseudodevice name with the attributes that you most commonly use. You should consider creating your own pseudodevice if you find that you are using the $CONTROL command frequently to modify the attributes of:

```
*PRINT*
*BATCH*
*PUNCH*
*IMPORT*
*EXPORT*
```

With $CREATE you can define several print pseudodevices so you can easily produce print jobs each with a different set of predefined attributes.

Furthermore, you can use the $CONTROL command to alter or add to the attributes of a user-defined pseudodevice. In this case, the pseudodevice is assigned a job number and held until it is explicitly released.

The general form of the $CREATE command for pseudodevices is:

```
$CREATE *pdn* TYPE=device OPEN="keywords"
```

where

| | |
|---|---|
| *pdn* | is an arbitrary pseudodevice name, consisting of 3 to 16 characters. The first and last characters must be asterisks (*), for example, *PRINT*. |
| device | is one of: PRINT, BATCH, PUNCH, EXPORT, or IMPORT. This represents the type of device that is to be modified. |
| keywords | is one (or more) of the keywords given in Table 3 above, including any *options* that are appropriate. These attributes are applied as if a $CONTROL command were given each time the device is opened for processing. |

Some typical examples:

(1) You frequently produce notices that are printed with the attributes PORTRAIT and PAPER=PLAIN. It would be more convenient if you did not have to control *PRINT* every time you printed one, so you define a new pseudodevice name with those attributes:

```
$CREATE *NOTE* TYPE=PRINT OPEN="PORTRAIT PAPER=PLAIN"
```

Now, any time you want to copy a notice to the printer, simply specify *NOTE* instead of *PRINT*:

```
# $COPY CLASSLIST *NOTE*
  *NOTE* assigned job number 146531
  *NOTE* RM146531 released to CNTR 6 images 3 sheets
         RATE=NORMAL FORMAT=PORTRAIT PAPER=PLAIN
```

This is especially useful when created in a sigfile.

(2) You have already defined the pseudodevice name *NOTE*, but you want to copy one notice with a *shaded overlay*:

```
# $CONTROL *NOTE* OVERLAY=SHADED
  *NOTE* assigned job number 187654
  *NOTE* RM187654 held
# $COPY OUTLINE *NOTE*
# $RELEASE *NOTE*
  *NOTE* RM187654 released to CNTR 6 images 3 sheets
         RATE=NORMAL FORMAT=PORTRAIT PAPER=PLAIN OVERLAY=SHADED
```

The pseudodevice *NOTE* now has the attributes of portrait orientation, plain paper, and a framed overlay. This example illustrates that when you alter the attributes of a pseudodevice that you have created the attributes are *added* to those already present for that device.

Notes:

(1) You cannot create a pseudodevice name that is already defined.

(2) Use the $FILESTATUS command to see what pseudodevice names are currently defined:

```
$FILESTATUS *?*
```

This will list all the names (including predefined ones), but none of the attributes.

(3) A user-defined pseudodevice is like a temporary file: it exists for the current session only. You can use $CREATE in your sigfile to define one or more pseudodevice names automatically at the beginning of each session.

(4) You can remove a user-defined device by using the $DESTROY command.

## THE $DISPLAY COMMAND

The $DISPLAY command lets you display options that have been set with the $SET command, or establish whether a given pseudodevice name is defined or active.

A complete description of the $DISPLAY command is given in the section "MTS Command Mode" in this volume. Only those items that specifically relate to Resource Manager job handling are included here.

The general form of the command is:

$DISPLAY keyword

or

$DISPLAY *pdn*

where *keyword* is one of the $SET command keywords, and **pdn** is a pseudodevice name.

### Displaying $SET Command Keywords

Any RM keyword that may be used on the $SET command may also be used on the $DISPLAY command to determine its current setting. For example, if you had used

```
$SET FORMAT=PORTRAIT
```

in your sigfile, you could display this information at any time during the current session:

```
# $DISPLAY FORMAT
# FORMAT=PORTRAIT
```

If the item you display is set to the default value, the system would tell you that the default is in effect:

```
# $DISPLAY OVERLAY
# OVERLAY is defaulted.
```

See Table 4 later in this section for a list of keywords and their default settings.

## Displaying Pseudodevice Names

If you have started a job for a particular pseudodevice but not yet released it (for example, by using the $CONTROL command), you can display the job name, and other information for the given pseudodevice name. For example:

```
# $DISPLAY *PRINT*
# *PRINT* RM123456 routed to CNTR 4 images 2 sheets RATE=NORMAL.
```

You can see that you have a print job with 4 images on 2 sheets waiting to be released to CNTR at normal rates.

If you want to see which pseudodevices are currently being held you can use:

```
# $DISPLAY *?*
# *NOTE* RM113441 routed to UNYN 6 images 3 sheets RATE=NORMAL
# *PRINT* RM113440 routed to UNYN 4 images 4 sheets RATE=NORMAL
          FORMAT=PORTRAIT TWOSIDED=YES
```

You currently have two jobs for the Xerox 9700 page printer at UNYN that are waiting to be released.

You may display both predefined and user-defined pseudodevice names. If the specified pseudodevice is currently active, the following information is displayed for the given device type:

| *Device Type* | *Information Displayed* |
|---|---|
| PRINT | Job name and number, print route, delivery route (if any), number of images and sheets, charge rate, number of copies (if more than one), and format parameters (if any). |
| BATCH | Job name and number, time estimate (if any), delivery route (if any), host, print route, and format parameters (if any). |
| PUNCH | Job name and number, delivery route (if any), number of records, and charge rate. |

Notes:

(1) If the specified pseudodevice name is defined but is not currently active for your userID, the system sends an appropriate response:

```
# $DISPLAY *PRINT*
# *PRINT* is defined but is not active.
```

(2) You can display your current session statistics using the command:

```
$DISPLAY TAILSHEET
```

## THE $LOG COMMAND

This command can be used to keep a record of input and output operations on a file or device. A popular use of this command is for recording your terminal session in a file. When you are logging your terminal session, the I/O operations are the commands that you are typing and the device is *MSOURCE* (the keyboard that you are using). It is useful to keep a record of your terminal session in the event that an unexpected error occurs and you need a *hardcopy* of the sequence of commands that were entered.

The general form of the $LOG command is:

$LOG fdname1 ON fdname2 options

where *fdname1* and *fdname2* may be files or devices and *fdname1* will default to *MSOURCE*, but *fdname2* must be specified. Options can be one of:

READS
WRITES
CONTROL
BINARY
ALL
ASIS
SYMBOLIC

The first five options determine which operations are logged, the last two determine the format of the $LOG output. If you do not specify options they will automatically default to READS, WRITES, and ASIS. These options are more fully explained in the description of the $LOG command in the section “MTS Command Mode” in this volume.

Some typical examples:

(1) You want to keep a record of your terminal session in the file LOGFILE.

```
$LOG *MSOURCE* on LOGFILE
```

The file LOGFILE will contain all commands entered from your keyboard and all the output sent to your terminal.

(2) You are submitting a job to batch and you want to keep a record of your session in a file instead of having it printed. The commands for your batch run are in a file that looks like this:

```
$SIGNON *
$EMPTY LOGFILE
$LOG *MSINK* ON LOGFILE
$RUN SPSS:xxx INPUT=DATA PRINT=MYPRINT
$CANCEL *PRINT*
$SIGNOFF
```

Our file contains six lines. First you have your $SIGNON command, next you $EMPTY the file that you are using to log your session in. Note that you must $CREATE LOGFILE before you submit this run. You cannot use a temporary file as in the previous example since you are logging in batch and it would disappear once the job was completed. The next command instructs the system to log your job in LOGFILE. You then enter your commands for this run. In this example you are running the statistical package SPSS:xxx. Finally, you enter $CANCEL *PRINT* which will cancel all hardcopy output from this job. Because $CANCEL will cancel all printouts for your job, you should use it with caution. In this instance the only hardcopy that you are expecting is the log of your batch run which you are storing in a permanent file anyway.

Notes:

(1) If you are logging terminal I/O and you $LIST the logfile or $EDIT the logfile and issue a PRINT /F command, the list is (potentially at least) infinite, since each line that is listed or printed is also added to the end of the logfile. In general, when using $LIST you should specify the line range to avoid this problem, or alternatively, you can turn logging off first (see below).

(2) You can use the $DISPLAY command to see which files or devices are currently being logged:

```
# $DISPLAY LOGSTATUS
# Logging *MSINK* on LOGFILE
#   30 lines read
#   192 lines written
```

In this example you are informed that you are logging your terminal session on the file –LOG. You have read 30 lines which have resulted in 192 lines of output into the file –LOG.

(3) You can stop logging by issuing the $LOG OFF command:

```
# $LOG OFF
# Logging of *MSINK* on LOGFILE terminated
```

## THE $RELEASE COMMAND

A job will be held if one of the following conditions apply:

(1) You have issued a $SET AUTOHOLD=ON command.

(2) You have issued a $CONTROL *pdn* for a particular pseudodevice.

In all of the above situations, the job is held until you issue a $RELEASE command. If you neglect to use $RELEASE, the system will do so automatically when you sign off. However, until the job is released (one way or another), it does not get submitted to the processing queue.

The general form of the $RELEASE command is:

$RELEASE *pdn*

When a job is released, the job name and number and attributes applied using either $SET or $CONTROL commands are displayed.

Some typical examples:

(1) You want to copy a file to *PRINT* and print it in portrait format:

```
# $CONTROL *PRINT* PORTRAIT
  *PRINT* assigned job number 151357
  *PRINT* RM151357 held
# $COPY THESIS *PRINT*
# $RELEASE *PRINT*
  *PRINT* RM151357 released to CNTR 72 images 36 sheets
          RATE=NORMAL FORMAT=PORTRAIT
```

(2) You have given a $CONTROL command, but changed your mind before copying anything to the printer:

```
# $CONTROL *PRINT* PAPER=3HOLE
  *PRINT* assigned job number 124680
  *PRINT* RM124680 held
# $RELEASE *PRINT*
  *PRINT* RM124680 cancelled: no lines.
```

Nothing is printed, and the job is abandoned.

Notes:

(1) If you have used $SET AUTOHOLD=ON, you will need to explicitly release any job sent to any pseudodevice (e.g., *PRINT*, *BATCH*), regardless of whether you used a $CONTROL command. The $SET AUTOHOLD=ON is a useful command to put into your sigfile because it explicitly holds any files sent to a *pdn* and thus gives you the option of cancelling a job before it is released.

(2) If you did not copy anything to the printer before releasing it, the job would be automatically cancelled (see example 2).

(3) If there was no print job active, the system would respond that the given pseudodevice name *was not held*.

(4) If you neglect to $RELEASE a job, the system will do so automatically when you sign off from the current session.

(5) Once you $RELEASE your job the options for that pseudodevice are reset back to their default values; with the exception, of course, of those that have been changed using $SET.

(6) You can use $DISPLAY to see what is currently held, for a particular device:

```
$DISPLAY *PRINT*
```

or for all devices:

```
$DISPLAY *?*
```

(7) If you have a number of pseudodevices active, you can release them all at once by typing the following:

```
        $RELEASE *?*
```

## THE $SET COMMAND

The $SET command lets you set various *global* conditions. Any option specified on a $SET command remains in effect until the end of the current session, or until it is reset to a new value. Many of the options described here apply to jobs that are directed to a printer. Some of them are valid only for the Xerox 9700 page printer.

The $SET command is particularly useful for options that you use all the time. In this case you can put your $SET commands into a special file called a sigfile and then have that file executed each time you $SIGNON.

The general form of the $SET command is:

$SET option

where *option* is selected from Table 4 below.

Note: Only the options that specifically relate to RM job handling are included here; for complete documentation of the $SET command, see the section "MTS Command Mode" in this volume.

**Table 4: $SET Options**

| *Option* | *Description* |
|---|---|
| ADDRESS="line1;line2;..." | Specify the campus mail address for delivered output (when DELIVERY=MAIL is set). This option applies only to page-printer output, not to line-printer or local-printer output. |
| AUTOHOLD={OFF \| ON} | Specify that all subsequent jobs sent to *BATCH*, *PRINT*, and *PUNCH* will automatically be held until explicitly released. This will also apply to all user-defined pseudodevices. The default is OFF. |
| COMMENT="text" | Specify a comment to be printed on the head sheet of the job. |
| COPIES=n | Specify number of copies of output for print jobs. The default is 1. |
| DELIVERY={station \| MAIL \| NONE} | Specify a delivery station to which output should be delivered by messenger service (see the file *DELIVERY for further details). If MAIL is specified, the output will be delivered by campus mail or the US Postal Service. The default is NONE. |
| FORMAT={LANDSCAPE \| PORTRAIT \| TWOUP \| format-name} | Specify format for page-printer output. The default is LANDSCAPE. |

JOBNAME={jobname | DEFAULT}
: Specify a job name of 1 to 8 alphanumeric characters for all subsequent jobs sent to the printer. The first character must be a letter. DEFAULT specifies the default format of "RM" plus six digits.

MARGIN={n.nn | NO}
: Set the left margin to "n.nn" inches. "n.nn" must be less than the current page width (8.5 for portrait orientation, 11.0 for landscape). MARGIN=NO turns off the margin override and resets the margins to the default for the current format (0.5 for PORTRAIT and 0.65 for LANDSCAPE).

NUMBER={(b,l,c) | NO}
: NUMBER=(b,l,c) numbers pages automatically, starting with number "b", printing the number on line "l", ending in column "c". NUMBER=NO (the default) disables automatic page-numbering. The page number is always printed in the first font of the current format or FONTLIST specification.

OVERLAY={NONE | SHADED | LINED}
: Specify an overlay for page-printer output. The default is NONE.

PAGES=n
: Specify a page limit for a print job. The default is no page limit. It applies only to jobs directed to *PRINT* or user-defined pseudodevices with TYPE=PRINT.

PAPER={PLAIN | 3HOLE | LABEL24 | LABEL33}
: Specify paper type for Xerox 9700 page-printer output. The default is PLAIN.

PRINTER={PAGE | LINE}
: Specify the type of printer. The default is PAGE.

ROUTE=station
: Specify a station for output (use the command $COPY *ROUTE to obtain a list of valid station codes).

SEPCOPY={YES | NO}
: Specify whether each copy will have separate head and tail sheets (YES) or whether all copies will be printed together as one job with a single head and tail sheet (NO). The option is only effective for page-printer output and if COPIES=n is specified. If each copy is more than 50 pages, then SEPCOPY=YES is forced. The default is NO.

SHIFT={YES | NO}
: Specify whether page-printer output is shifted away from the binding edge. The default is SHIFT=NO.

TWOSIDED={YES | NO}
: Specify printing on one or both sides of the paper for page-printer output. The default is YES.

Some typical examples:

(1) You have decided that *all* output directed from the current session to *PRINT* should be printed in portrait format, on one side only:

```
        $SET FORMAT=PORTRAIT TWOSIDED=NO
```

All subsequent output for the Xerox 9700 page printers will be printed in portrait mode, on one side of the paper. This demonstrates how more than one option may be used on a $SET command.

(2) You are working in the Church Street (CHUR) computing site and you want to print all your output for the current session to the the Xerox 4045 printer located there:

```
        $SET ROUTE=CHUR
```

Anything sent to *PRINT* will now be printed at that computing site. Remember that this type of printer cannot handle many of the characters and specifications that are available only on the Xerox 9700 page printers.

(3) You want to ensure that none of your print jobs exceed 20 pages of output so you enter:

```
        $SET PAGES=20 AUTOHOLD=ON
```

Now whenever you $COPY or $LIST a file to *PRINT*, only 20 pages (images) will be copyed after which an error message:

```
        # *PRINT* assigned job number 348902
        # ***Local Page Limit Exceeded
```

will be printed. This option can be overidden for a particular print job by using the $CONTROL command.

Notes:

(1) The $SET options remain in effect for the whole session, or until explicitly reset.

(2) Using $SET does not initiate any particular “job”.

(3) Values that have been set may be displayed (see the $DISPLAY command).

(4) If you submit a batch job from a terminal session or from another batch job, the attributes of the job will be defaulted from any values that have been $SET in the submitting job unless overridden by the $SIGNON command in the submitted job.

## THE $SYSTEMSTATUS COMMAND

The $SYSTEMSTATUS command lets you find out the current status of any job (or jobs) that you have submitted for processing within the last 8 hours, or that has not finished processing. It can also tell you how busy the computer is overall.

The general form of the command is:

```
    $SYSTEMSTATUS USERS
```

or

```
      $SYSTEMSTATUS QUEUE option
```

where *option* selects the job(s) for which you want queuing information.

## The USERS Keyword

This form of the command gives you a general idea of the system load. The information displayed includes:

(1) number of terminal users signed on

(2) number of active batch jobs

(3) number of active non-MTS jobs

(4) number of virtual pages in use

(5) number of real pages in use

## The QUEUE Keyword

This displays the status of individual jobs that you have submitted for processing. It also provides summary information about queue lengths. You must be signed on to the userID that submitted the job in order to see its current status.

The general form of the command is shown above, and the available *options* are listed below.

| *Option* | *Information Displayed* |
|---|---|
| jobnumber | The current status of the specified job. |
| jobname | The current status of the specified job. |
| USER | The status of all jobs submitted by the current userID, but not yet processed. |
| * | The status of all jobs processed for the current userID during the preceding 8-hour period. This will include jobs that have printed, or were cancelled, or are still waiting to be processed. This will also include BITNET files that are being held in *IMPORT*. |
| ALL | A summary including the number of active and queued jobs, the number of pages waiting to print for all printers, and the number of plot minutes in the queue. This information represents the whole system, not just the current userID. |
| PROUTE=code | A summary of all jobs awaiting print at a specified print station. |
| IMPORT | A summary of all BITNET files that are awaiting import. |

Some typical examples:

(1) If you just want to get an idea how busy the system is:

```
# $SYSTEMSTATUS USERS
  There are 267 terminal users, 2 batch jobs, 216 available
  lines, and 60 non-MTS jobs using 20150 private virtual pages
  and 9110 real pages. Additionally, there are 1270 shared
  virtual pages.
```

(2) The owner of the userID 1ABC submitted two jobs and wants to see if they have finished printing.

```
# $SYSTEMSTATUS QUEUE USER
  1ABC:RM116982 is awaiting print, P14, after 0 jobs.
```

This tells you that one job is next in the queue to be printed. In order to see the status of all jobs (included those that have printed within the last 8 hours) you should enter:

```
# $SYSTEMSTATUS QUEUE *
  1ABC:RM116982 is awaiting print, P14, after 6 jobs.
  1ABC:RM118973 is awaiting purge, was printed on PTR2 at
    10:33:00 Mon Jan 15/90.
  1ABC:RM113243 was printed on PTR2 at 09:24:11 Mon Jan 15/90.
```

The first job is awaiting print and there are six jobs ahead of it in the queue. The second entry represents a job that has been printed but has not yet been *purged*. For an explanation of the the term “purged”, see “When is a Job Purged?” in this section. The last entry represents a job that has been printed and has also been purged.

(3) How many jobs are waiting to print on the Church Street (CHUR) computing site printer?

```
# $SYSTEMSTATUS QUEUE PROUTE=CHUR
  1 active job, 3 queued jobs, representing 81 pages.
```

(4) You want to track a job that was submitted on the UB-MTS system and tranferred to the UM-MTS system for printing. From the UB-MTS system, you can enter

```
# $SYSTEMSTATUS QUEUE RM123456
  1ABC:RM123456 is awaiting purge, was sent at 12:10:15
    Mon Jan 15/90.
```

This shows the time the job was sent to UM-MTS for printing. If you have an account with the same userID on the UM-MTS system, you can enter

```
# $SYSTEMSTATUS QUEUE RM123456
  1ABC:RM123456 (345678) is awaiting purge, was printed on PTR2
    at 12:20:20 Mon Jan 15/90.
```

Note that the UB-MTS job (numbered 123456) was given a new job number of 345678 when it was transferred to the UM-MTS system. However, the job name RM123456 remained the same.

## Explanation of Responses

The following list summarizes the responses you can normally expect when querying the status of a job.

(1) *job name* is awaiting execution, P*m*, after *n* jobs

This indicates a batch job that is waiting in the execute queue. It also tells you the priority of the job, and the number of batch jobs ahead of it. (For more information on priorities, see the section "Batch Processing".)

(2) *job name* is executing

This indicates a batch job that is currently executing.

(3) *job name* is awaiting print, P*m*, after *n* jobs

This indicates a job that is waiting in the print queue. It also tells you the priority of the job, and the number of print jobs ahead of it.

(4) *job name* is printing

This indicates a job that is currently printing.

(5) *job name* is awaiting purge, was printed on PTRn at **...**

This message gives the time and date at which the job was printed. The job is finished, but has yet to be purged. (See Note 3 at the end of this section.)

(6) *job name* was executed at **...**

This indicates the job has been executed.

(7) *job name* was printed on PTRn at **...**

This indicates the job was printed and has been purged from the system.

(8) *job name* was cancelled at **...**

This indicates the job was cancelled (see the $CANCEL command).

(9) *job name* not locatable

This indicates you are requesting information for a job that the RM does not have information for. Perhaps it finished more than eight hours ago, or was submitted under another userID.

(10) *job name* was sent at **...**

This indicates the job was sent to another system for processing. This message applies to print, punch, or BITNET jobs that require a device that is attached to another system.

All print jobs from the UB-MTS system are sent to the UM-MTS system for printing.

Notes:

(1) $SYSTEMSTATUS always looks for jobs for the current userID. It automatically prefixes each job name with this userID.

(2) Note the difference between the following QUEUE options:

```
# $SYSTEMSTATUS QUEUE USER
  No jobs found for ID 1ABC.
# $SYSTEMSTATUS QUEUE *
  1ABC:RM119435 was cancelled at 15:36:23 Wed Jan 10/90.
```

The * will find jobs that were purged or cancelled earlier in the day, whereas "USER" locates only those that are active or pending.

(3) The Xerox 9700 page printer maintains its own internal print queue. The time given as printed is actually the time the Resource Manager copied the job to the printer. If there is a heavy backlog inside the printer or it is out of paper, there may be a lag between the time given and the time the job really gets printed.

### When is a Job Purged?

The expression "awaiting purge" simply means that the job is finished, but still exists on the system. So, for example, if operators discovered your output was printed while the Xerox 9700 page printer was malfunctioning, they could reprint the job when the problem was fixed. This saves you the task of rerunning the job.

After a set amount of time, the Resource Manager purges jobs which have already printed in order to make room for new ones being submitted. The length of time a job remains in the purge queue varies depending on the type of job.

## THE $VIEW COMMAND

The VIEW command allows users to look at batch and print jobs that are queued by the Resource Manager for execution or printing.

For example, you can look at the contents of the print job 123456 by first entering the command

```
VIEW 123456
```

at the MTS "#" prompt. Then, you can use MTS File Editor commands (including visual mode) to examine the job. You can copy sections of the job to other files, cancel it, or reroute it. If the job is queued for execution, you will see the commands that make up the batch job. If the job is queued for printing, you will see the output produced by the job.

Print jobs must be released to the Resource Manager before they can be viewed. That is, you cannot give the commands

```
# COPY file *PRINT*
```

```
> *PRINT* assigned job number 123456
# VIEW 123456
```

but you can give

```
# COPY file *PRINT*
> *PRINT* assigned job number 123456
# RELEASE *PRINT*
  *PRINT* RM123456 released to ...
# VIEW 123456
```

You can view print output only while it is queued for printing. Once printed, it can no longer be viewed.

You can send jobs to the VIEW command by using the ROUTE=VIEW option on the $CONTROL or $SIGNON commands. Jobs submitted with this routing will not print but will remain queued in the system for up to 11 days until rerouted or canceled (both described later).

For example, to use ROUTE=VIEW for a print job, use the commands

```
CONTROL *PRINT* ROUTE=VIEW
COPY file *PRINT*
RELEASE *PRINT*
```

You can then use the VIEW commands described below. For batch jobs, place the parameter ROUTE=VIEW on your $SIGNON command.

If you currently submit batch jobs and only examine the printed output to determine the success or failure of the job, then VIEW can be of use to you. If you submit your job with a ROUTE=VIEW, you can examine the output from your terminal. Having looked at it, you can then cancel it.

If you submit a batch job that is going to generate a lot of output, it might be worth first routing the output to VIEW. You can then examine the output and, if the job is correct, reroute it to the printer.

The VIEW command requires a job number to identify the job. You can find the job number by using the LOCATE command. To see what jobs you have in the system, enter the command:

```
LOCATE
```

or, to see only the jobs with ROUTE=VIEW:

```
LOCATE VIEW
```

You will see a message such as

```
RM299444 (234567) waiting print, posn 0, Route=VIEW.
RM299593 (234568) waiting print, posn 1, Route=VIEW.
```

Then, you can use the VIEW command to view a job:

```
VIEW 234567
```

You will see a message such as

```
* Print job 234567 RM234567 1ABC   3 pages submitted at
    11:34:32 Tue Jul 30/91
```

The asterisk “*” is a prompt that indicates the VIEW command is active. To examine the job in visual mode, enter the following command at the “*” prompt:

```
V
```

Subsequently, you could cancel the job with the command

```
CANCEL
```

To exit the VIEW command, enter

```
STOP
```

at the “*” prompt. For more help on the VIEW command, enter

```
HELP
```

while you are in the VIEW command.

You can reroute jobs from VIEW to a printer by entering the following command at the “*” prompt:

```
ROUTE station
```

For example,

```
ROUTE CNTR
```

will route the output of a job to the CNTR batch station. You can send jobs to other printers, such as the page printer at NUBS, by substituting NUBS for CNTR. Please note that you will probably not want to view text-formatted jobs, such as .DVI files, since their output normally is not readable using the File Editor.

If you work under more than one MTS userID, please note that you can only view jobs from the userID on which you originally executed them.

# SERVER USE OF MTS

An *MTS network server* is a program on an MTS computer (the host) that provides a service to a user or program (the client) on another computer. The host and the client are connected by the UMnet/Michnet Computer Network.

The MTS network server mechanism is an important part of the evolution of computing at the University of Michigan. It makes possible a future *server-enhanced networking environment*, in which the various kinds of computers attached to the UMnet/Michnet network cooperate to deliver a variety of multiple-computer services to users and to each other.

The MTS network server mechanism originally was created to make it easier for MTS programmers to coordinate the activity of multiple computers (at least one of them running MTS). As a side benefit, the mechanism makes it possible for the UMnet/Michnet Computer Network to provide certain "free" information services.

## MTS NETWORK SERVERS CURRENTLY IN USE

The following table shows the characteristics of some "public" MTS network servers that are currently in use.

Public MTS Network Servers

| Name | Host | Description | Sponsor | Client |
|---|---|---|---|---|
| FINGER UM | UM | Display UM user info | ITD | Both |
| FINGER UB | UB | Display UB user info | ITD | Both |
| FTP | UM, UB | File Transfer Protocol | ITD | Pgm |
| HELP | UM, UB | Online help for MichNet | ITD | User |
| MIRLYN | UM, UB | UM Library catalog | Univ Libraries | User |
| NETMAILSITES | UM, UB | List of remote mail hosts | Merit Network | User |
| TDAY | UM, UB | T-Day EBCDIC translation | ITD | Pgm |
| TIMEDATE | UM, UB | Display time and date | ITD | Both |
| UMLIBHOURS | UM, UB | UM Library open hours | Univ Libraries | User |
| UM-CIC | UM, UB | Campus events database | Campus Info Ctr | User |
| UM-ITD-SUGGEST | UM, UB | ITD Suggestion Box | ITD | User |
| UM-UHS-INFO | UM, UB | Health info query | Univ Health Svc | User |
| UM-UMIPS | UM, UB | UM Info. Posting System | ITD | User |
| UM-WEATHER | UM, UB | Weather information | AOSS Dept | User |

A more complete list of available servers is given by the HELP server.

Each server has a *name* that lets the network direct a request for the server to the proper host and lets the host direct the request to the proper server command file.

A server resides on a *host*, has a *sponsor*, and is intended for access by *users*, *programs*, or *both*.

## CLASSIFYING SERVERS

There are three ways to classify an MTS network server: by *client type*, by *funder*, and by *provider*.

### By Client Type

A server is said to be *user-oriented* if it is intended to be accessed directly by an interactive user. Examples include the servers UM-CIC, UM-UHS-INFO, and UMLIBHOURS. A UMnet/Michnet user can access these servers to get (respectively) a list of upcoming campus events, an answer to a health question, and a schedule of open hours at a University library.

A server is said to be *program-oriented* if it is intended to be accessed by a client program on another computer. Examples include the servers FTP and TDAY. The FTP (File Transfer Protocol) server on each machine exchanges files with corresponding servers on other machines, and the TDAY server is invoked by the T-Day translation program.

Note that program-oriented servers are often accessed by user-oriented programs as, for the above examples, $FTP and *TDAY88TRANS. However, it is the program, not the user, that directly accesses the program-oriented server.

A server can be both user- and program-oriented. For example, FINGER and TIMEDATE. It is also possible for a program to access a user-oriented server, and a user (with sufficient devotion) to access a program-oriented server.

User-oriented servers and program-oriented servers differ in several respects:

(1) **Purpose**. The purpose of a user-oriented server is to provide a centralized, usually free service to users anywhere on the network, even to users who have not signed on to a host computer. The purpose of a program-oriented server is to provide a streamlined, reliable way for two or more computers to undertake coordinated activity.

(2) **Format**. All exchanges with a user-oriented server consist entirely of text; exchanges with a program-oriented server may be text or binary.

(3) **Style**. A good user-oriented server offers “friendly” prompts and provides help if requested; a good program-oriented server sends output that is predictable and easily parseable.

Most public user-oriented servers are accessible at the “Which Host?” prompt and are free.

### By Funder

Any server’s use of MTS resources on its host computer must be paid for by a userID on that host. If the userID belongs to the client, the server is said to be *charged* for; if the userID belongs to someone else (the server’s *sponsor*), the server is said to be *free*.

Charged servers and free servers differ in that the client must send a charged server a *logon record* (see next section) to identify the userID and password.

### By Provider

Apart from the question of who pays for using a server is the question of who provides and supports the server program itself. *Public* servers are officially provided and supported by ITD; *private* servers are not.

Public and private servers differ in that:

(1) Practically speaking, a public server is under the stewardship of ITD, while private servers are not. ITD takes responsibility for ensuring that a public server works correctly and that it is documented accurately. ITD may provide this support itself, or by arrangement with another person or organization, typically the server's sponsor. For example, the University Health Service maintains the public server UM-UHS-INFO, and the Campus Information Center maintains the public server UM-CIC.

(2) Technically speaking, a public server is defined by having an entry in the host computer's *server table*. This table has one generic entry PRIVATE that handles all private servers.

(3) From the user's point of view, a public server (like a public file) is not associated with a particular userID, while a private server (like a private file) is.

(4) Public servers can be free or charged; private servers are always charged.

Note that the terms "public" and "private" imply nothing about who can access the server. They are analogous to the common terms "public sector" and "private sector," and consistent with long-standing MTS terminology for "public files" and "private files."

## COMPARING INTERACTIVE, BATCH, AND SERVER SESSIONS

How does an MTS network server session differ from the familiar interactive and batch sessions? The following table shows how interactive, batch, and server sessions progress from beginning through middle to end.

| Phase | Interactive Session | Batch Session | Server Session |
|---|---|---|---|
| Beginning | Write MTS banner | Write MTS banner | |
| | Read a $SIGNON | Read a $SIGNON | Generate a $SIGNON |
| | Write a signon banner | Write a signon banner | |
| Middle | Read project sigfile | Read project sigfile | Read project sigfile |
| | Read user sigfile | Read user sigfile | Read user sigfile |
| | Read interactive commands | Read batch commands | Read server command file |
| End | $SIGNOFF | $SIGNOFF | Generate a $SIGNOFF |

Notice how a server session shares some characteristics of an interactive session and some characteristics of a batch session. In particular:

(1) All three types of sessions begin with a $SIGNON (explicit or implicit), then process a project sigfile (if any) and user sigfile (if any) before processing any other commands, and end with a $SIGNOFF (explicit or implicit).

(2) Like a batch session, a server session generates an implicit $SIGNOFF if none is provided.

(3) Unlike the interactive and batch sessions, a server session writes nothing when it is invoked.

This last point is perhaps the major advantage of using a server session, if the client is a program: the server mechanism guarantees that the first output the client sees is the result the client asked for. That is, the client program need not parse the MTS headers and signon headers and possible other messages from UMnet/Michnet and MTS to search for the desired result.

## ACCESSING A SERVER

This section explains the various methods by which a client user or program can access an MTS network server.

After a general description of the client's *logon record*, some of the various access methods are presented.

### The Logon Record

To access a user-funded server, that is, a charged public server or a private server, the client must supply a *logon record*. A logon record consists of a sequence of *information items* separated by spaces. Each item is of the form *keyword=value*.

Certain items are required, and others are optional. The required items are:

| *Keyword* | *Description* |
|---|---|
| ID | The userID on the MTS host of the user to whom the server's use of resources will be charged. |
| PW | The password that corresponds to the userID. |
| CMDFILE | The name of the server's command file (private servers only). |

For example, a logon record for a charged public server might look like:

```
ID=WXYZ PW=GRUNCH
```

A logon record for a private server might look like:

```
ID=WABC PW=HUNGRY CMDFILE=WABC:MAILDROP
```

### Using the "Which Host?" Prompt

This method can be used by any client that can access the UMnet/Michnet network. It is most suited for human users, especially in the case of free public servers.

To access a *free public server* using the "Which Host?" prompt, type the server's name in response to that prompt. For example, here is a sample TIMEDATE server session:

```
Which Host? TIMEDATE
16:16:29 EST
Wed Jan 20/88
%H1E:CH0445-AF5E:UM24 timeda Connection closed
```

To access a *charged public server* using the “Which Host?” prompt: type the server’s name; press the Return key; and then type a *logon record* that supplies a userID to be charged and its password. For example, here is a sample server session charged to an imaginary user with userID MONA and password LISA:

```
Which Host? servername
ID=MONA PW=LISA
output here
%H1E:CH043D-AF1D:UM4E servername Connection closed
```

To invoke a *private server* using the “Which Host?” prompt: type the word “PRIVATE” followed by the name of the MTS host on which the server resides; press Return; and then type a logon record that supplies the userID of the owner of the server, the corresponding password, and the name of the server command file. For example, here is a session with an imaginary private server SHOWFILES owned by our imaginary user MONA:

```
Which Host? PRIVATE UM
ID=MONA PW=LISA CMDFILE=SHOWFILES
5 ML.CC             Create=Oct16/87
5 ML.CITI           Create=Nov02/87
%H1E:CH043D-AF1D:UM4E privat Connection closed
```

When typing a logon record at a terminal, it is useful to press the Escape key before and after the password to keep it from printing.

## Using the %GRAB Command

This method can be used by any client that can access a host computer on the UMnet/Michnet network. It is used from within a host session rather than at the “Which Host?” prompt, but is otherwise identical.

To access a *free public server* using the %GRAB command, type “%GRAB” followed by the server’s name. For example:

```
#%GRAB TIMEDATE
16:16:29 EST
Wed Jan 20/88
%H1E:CH0445-AF5E:UM24 timeda Connection closed
```

To access a *charged public server* using the %GRAB command: type “%GRAB” followed by the server’s name; press the Return key; and then type a *logon record* that supplies a userID to be charged and its password. For example:

```
#%GRAB servername
ID=MONA PW=LISA
output here
%H1E:CH043D-AF1D:UM4E servername Connection closed
```

To invoke a *private server* using the %GRAB command: type “%GRAB”, followed by the word “PRIVATE”, followed by the name of the MTS host on which the server resides; press Return; and then type a logon record that supplies the userID of the owner of the server, the corresponding password, and the name of the server command file. For example, here is a session with an imaginary private server SHOWFILES owned by our imaginary user MONA:

```
#%GRAB PRIVATE UM
ID=MONA PW=LISA CMDFILE=SHOWFILES
5 ML.CC            Create=Oct16/87
5 ML.CITI          Create=Nov02/87
%H1E:CH043D-AF1D:UM4E privat Connection closed
```

## Using the $MOUNT Command

This method can be used by any client that is signed on to an MTS session in the UMnet/Michnet network. It is often used from within a $SOURCE-able file of MTS commands.

To invoke a *free public server*, use a $MOUNT command of the form:

```
MOUNT MNET *pdn* DEST=host SERVER=server
```

where “pdn” is a pseudodevice name of your choosing, “host” is the name of the MTS host on which the server resides, and “server” is the name of the server (see the discussion on server names under “Comparing MTS Network Servers Currently in Use” above). Note that server and client may reside on the same host.

Here is an example of accessing the free public TIMEDATE server interactively using the $MOUNT command:

```
#MOUNT MNET *A* DEST=UM SERVER=TIMEDATE
#mnet *a* dest=um server=timedate
#*A*: Mounted on AD00
#COPY *A*
>16:25:00 EST
>Tue Jan 26/88
AD00: Connection CLOSED
#RELEASE *A*
#"*A*": MNET dismounted.
```

To invoke a *charged public server* or a *private server*, issue the MOUNT command as above and then follow it with a logon record as described above under the “Which Host” method. One way to do this is to copy it to the server’s pseudodevice:

```
COPY *SOURCE* TO *pdn*
ID=userid PW=password CMDFILE=cmdfile
{end-of-file}
```

The output of the server can then be read using a command of the form:

```
COPY *pdn*
```

Once the server completes, release the pseudodevice name with a command of the form:

```
RELEASE *pdn*
```

Here is an example of accessing our imaginary private SHOWFILES server owned by userID MONA:

```
#MOUNT MNET *A* DEST=UM SERVER=PRIVATE
#mnet *a* dest=um server=private
#*A*: Mounted on AD00
#COPY *SOURCE* TO *A*
>ID=MONA PW=LISA CMDFILE=SHOWFILES
>$ENDFILE
#COPY *A*
>5 ML.CC             Create=Oct16/87
>5 ML.CITI           Create=Nov02/87
AD00: Connection CLOSED
#RELEASE *A*
#"*A*": MNET dismounted.
```

For further information about other methods for accessing servers and for creating servers, see *Using and Creating MTS Network Servers*, Reference R1073.

# FILES AND DEVICES

Files and devices are used by programs for the input and output of data. A file is a logical entity that contains data. A device is a physical entity such as a card reader, a magnetic-tape unit, or a magnetic-disk unit that transmits data to and from a file or a program. The general specification of a file or device is called a *file or device name*, and is abbreviated as *FDname*. The purpose of this section is to describe the kinds of files and devices available, to give rules for specifying FDnames, and to illustrate their use.

## FILES

A *file* is an ordered set of zero or more lines. A *line* is a string of one or more characters (bytes). A line may contain up to 32,767 characters.

On-line storage in MTS is organized on the basis of files of information stored on magnetic disk units. It is common practice to refer to such files as "file storage," "disk files," or simply "files." These files are always available when MTS is operating (except in cases of hardware failure) and are considered to be "on-line." Other storage media such as punched cards and magnetic tapes may be used to store information but information stored in this manner is usually not referred to as file storage. Furthermore, such information is considered to be "off-line" since some manual intervention is required to gain access to the information. Disk files may contain source decks, object decks, data sets, output listings, writeups, etc.

Files are classified in two ways:

(1) the category of the file, and
(2) the organization of the file.

These two characteristics are established when the file is created. The category is specified by the form of the file name. The type of organization is specified by a keyword on the CREATE command.

### Categories of Files

*Public* files are files containing components of the system, such as language translators and utility programs. Public files are usually permitted such that they can be read by all users, but are protected against modification. *MTS Volume 2: Public File Descriptions*, Reference R1002, contains descriptions of the public files. Public files are also called system files or library files, and their names always begin with an asterisk (*). Most users cannot create public files.

*Private* files are files belonging to a specific user and may be accessed only by that user, unless permission is given to others for accessing them. There are two types of private files, *permanent* and *temporary*. Permanent private files must be explicitly created by the user with the CREATE command (or by calling the subroutine CREATE). Once created, they exist until the user explicitly destroys them with the DESTROY command (or by calling the subroutine DESTROY). For the descriptions of these subroutines, see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

*Temporary* files may be created explicitly (via the CREATE command or the CREATE subroutine) or implicitly. Implicit creation of a temporary file occurs when the file does not exist (i.e., has not been previously created either explicitly or implicitly) but is referenced in a command or through a subroutine call. Implicitly created files always have the default file characteristics. Temporary file names always begin with a minus "–" sign. Temporary files must be created explicitly if other than the default characteristics are required. The default characteristics are discussed later and are also given in the CREATE command description in this volume. Temporary files may be explicitly destroyed at any time before the user signs off; all remaining temporary files that still exist are automatically destroyed by MTS when the user signs off.

## Organization of Files

There are two types of file organization, the *line* file and the *sequential* file.

The line file is the most commonly used file type in MTS. A line file is an ordered set of zero or more lines. Each line may consist of 1 to 32,767 characters (bytes). Each line has associated with it a unique *line number*. Although the line number is stored on the disk with the line, it is not a part of the contents of the line. The lines are numerically ordered and are in the range –2147483.648 to 2147483.647. By specifying its line number, any line in the file may be directly accessed.

A sequential file is an ordered set of zero or more lines. Each line may consist of 1 to 32,767 characters (bytes). The lines of a sequential file do not have line numbers associated with them; therefore, the lines in the file may not be directly accessed. The lines may be accessed only sequentially. Sequential files are rarely used any more. They were useful in the past when the length of line-file lines was restricted to 255 characters.

The file organization is established at the time the file is created by specifying the TYPE keyword with the CREATE command. For example,

```
    CREATE SFILE TYPE=SEQ
```

creates the sequential file named SFILE. If the TYPE keyword is omitted, the type defaults to LINE. All temporary files that are implicitly created (by reference as opposed to by the CREATE command) are line files. If a temporary sequential file is desired, it must be created explicitly by the CREATE command or by the CREATE subroutine.

## File Names

Permanent private file names consist of one to twelve characters. The name may contain the letters A–Z (case is insignificant), the digits 0–9, and the following special characters:

< > $ * – % # / . _ !

The name cannot begin with an asterisk (*) and should not begin with the special flag characters ">" or "#". If more than twelve characters are specified, only the first twelve are used; the rest are ignored. Examples of legal file names are:

```
    PHYLE
    A_LONG_NAME
    STATPROG.SOU
```

Internally, the name of the file consists of the user's four-character userID followed by the external twelve-character name. In this way, the names of one user's files are always different from those of other users.

If a user wishes to access a permanent file belonging to another user, the file name must be prefixed by the userID of the owner. The userID and file name are separated by a colon (:). Thus, if user AAAA wants to read the file NEWS belonging to user BBBB, it must be referred to as

```
BBBB:NEWS
```

However, user AAAA may read the file only if user BBBB has given user AAAA read access to the file. See the section "Shared Files" for further details about accessing files.

Temporary file names consist of one to eight characters prefixed by a minus sign "–". This prefix is called the temporary-file character. The legal characters for a temporary file name are identical to those for a permanent file name. Temporary file names are restricted to a maximum of 9 characters (including the leading minus sign). Examples of legal temporary file names are:

```
-T
-LOAD
```

Public file names begin with an asterisk (*), cannot end with an asterisk, and contain a maximum of sixteen characters (including the asterisk). Examples of legal public file names are:

```
*FORTRANVS
*CONSULTINGHOURS
```

## DEVICES

A *device* is a physical unit of hardware, such as a card reader, a magnetic tape unit, or a telephone line adapter (or port) that a user's terminal can call. All input of data into the system and output of data from the system is ultimately performed by devices.

### Device Names

Each device has a unique four-character *device name* and a three- or four-character *device type*. All similar devices have the same type. A complete list of devices is given in the section "System Device List" in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003. Since the user generally does not know and should not care which specific devices are being used for his job, pseudodevice names are used in their place. Device names are generally used only by the operators and the system programmers. Since the user occasionally sees device names, a brief mention is made here.

### Pseudodevice Names

A *pseudodevice name* (or *pdn*) refers to a file or device when the actual name for the file or device is not available or when the user desires to have a single name to refer to a set of file or device names.

For batch use, pseudodevice names are needed for the card reader, the line printer, and the card punch since the actual device names for these units are not available to the user. For conversational use, pseudodevice names are needed for the terminal (input and output) since the terminal port name is not available to the user. These pseudodevice names are predefined and are given below.

Pseudodevice names are also needed to refer to user-mounted volumes such as magnetic tapes since the actual tape drive device names are not available to the user. These pseudodevice names are not predefined, but are defined by the user at the time these volumes are mounted. See the MOUNT command description in this volume and in *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019, for further details of user-mounted media and their associated pseudodevice names.

A pseudodevice name begins with an asterisk, ends with an asterisk, and has from one to fourteen characters in between. The same characters that are legal for file names are legal for pseudodevice names. Examples of legal pseudodevice names are:

```
*P*
*TAPE*
*T12*
```

There are eight pseudodevice names which are predefined for the user. These are the following:

| | |
|---|---|
| *MSOURCE* | is defined as the master source (or input) file or device. For batch mode, it is the card reader; for conversational mode, it is the keyboard of the terminal. *MSOURCE* may not be redefined by the user. |
| *MSINK* | is defined as the master sink (or output) file or device. For batch mode, it is the line printer; for conversational mode, it is the printer (or display screen) of the terminal. *MSINK* may not be redefined by the user. |
| *SOURCE* | is defined as the current source (or input) file or device. Initially, the system defines *SOURCE* to be the same as *MSOURCE*. Thus, for batch mode, *SOURCE* is the card reader, and for conversational mode, *SOURCE* is the keyboard of the terminal.<br><br>The user can redefine *SOURCE* by using the SOURCE command. If *SOURCE* has been redefined by a user, an attention interrupt at a terminal, or an end-of-file on *SOURCE* when attempting to read a command, redefines *SOURCE* back to *MSOURCE*. |
| *SINK* | is defined as the current sink (or output) file or device. Initially, the system defines *SINK* to be the same as *MSINK*. Thus, for batch mode, *SINK* is the line printer, and for conversational mode, *SINK* is the printer (or display screen) of the terminal.<br><br>The user can redefine *SINK* using the SINK command. If *SINK* has been redefined by a user at a terminal, an attention interrupt redefines *SINK* back to *MSINK*. |
| *DUMMY* | is defined as an infinite wastebasket for output (lines are discarded) and an empty file for input (every time a line is requested, an end-of-file condition is returned). *DUMMY* is particularly convenient for specifying that output is to be ignored. |
| *PUNCH* | is defined as the punch file or device. For batch mode, *PUNCH* is the card punch. If the user has specified a card limit on this SIGNON command, output written to *PUNCH* is punched on cards. For conversational mode, output written to *PUNCH* creates a new batch |

| | |
|---|---|
| | queue entry to punch the output on cards. A receipt number for retrieving the punched output will be printed at the terminal. |
| *PRINT* | is defined as the print file or device. In batch mode, this is the same as *MSINK*. In conversational mode, output written to *PRINT* creates a new batch queue entry to print the output. A receipt number for retrieving the printed output will be printed at the terminal. |
| *BATCH* | is defined for both batch and conversational mode. Output written to *BATCH* creates a new batch queue entry to process the input provided as a separate batch job. A receipt number for picking up the batch job's output will be printed at the terminal (if submitted from a terminal), or in the printed batch output of the submitting job (if submitted from batch). |
| *IMPORT* | is defined for importing data using BITNET connections. |
| *EXPORT* | is defined for exporting data using BITNET connections. |

## SIMPLE FDNAMES

A *simple FDname* is one of the following:

(1) a file name, or
(2) a pseudodevice name.

An FDname that is not a pseudodevice name is treated as a file name. To specify that an FDname is only a file name, it may be prefixed with the file-name character "#". This is required only when the first character of the file name is a special flag character such as "–" or ">"; e.g.,

```
  -T  refers to a temporary file with name T
 #-T  refers to a permanent file with name -T
```

## SUBSETS OF FILES

### Line Numbers

Line numbers are used to specify individual lines associated with a file or device. Line numbers exist in two formats: external and internal. The external format is normally used by the user to specify an FDname with a line-number range; the external format is converted by the system to the internal format and is used to process the line number during I/O operations. The internal format is also required for certain I/O subroutines.

The external format of a line number may be one of the following:

(1) ±nnnnn.nnn

where "n" is a decimal digit (0 through 9). The minimum and maximum line numbers are –2147483.648 and 2147483.647. When writing a line number, leading plus signs, leading zeros, trailing decimal points, and trailing zeros after decimal points may be omitted. Examples of line numbers of this form are:

```
        5    5.1    5.13    5.137    32505.137    -32505.137
```

(2) FIRST or *F

which has the value of the first (numerically least) line number in the file. If the file is empty, the value is zero.

(3) LAST or *L

which has the value of the last (numerically greatest) line number in the file. If the file is empty, the value is zero.

(4) FIRST±m or *F±m
LAST±m or *L±m
MIN+m
MAX−m

where "±m" is a number of the form "±nnnnn.nnn" as described above, and MIN and MAX are the numbers −2147483.648 and 2147483.647, respectively. The value of this is the sum or difference of the two components; thus LAST−1 does not necessarily specify the line number of the next-to-last line, but merely a line number 1 less than that of the last line.

The internal form of a line number is a fullword binary integer whose value is 1000 times the external form. Thus, a line number whose external form is 1 is stored internally as 1000 (decimal) or 000003E8 (hexadecimal). The internal form of a line number must be supplied to the input/output subroutines when requesting an indexed operation, and the internal form of the line number of the line that was read is returned after a sequential read operation (see the description of READ and WRITE in *MTS Volume 3*).

## Line-Number Ranges

A subset of a file or device may be specified by appending a *line-number range* to the FDname. This line-number range is given in the form

(b,e,i)

where "b", "e", and "i" specify the beginning line number, the ending line number, and the increment, respectively. Any or all of these items may be omitted. Trailing commas resulting from the omission of any of these items may also be omitted, but leading and intermediate commas are required.

The bounds of the subset of the file or device are determined by the beginning line number and the ending line number. All lines between the beginning line number and the ending line number are included in the subset. If "b" is omitted, the subset extends to line 1 of the file or device. If line numbers less than 1 are desired, an explicit beginning line number must be specified or the line number FIRST (or *F) must be used. If "e" is omitted, the subset extends to the end. For example,

```
A(10,20)
```

specifies all lines in file A with line numbers between 10 and 20, inclusive.

```
A(10)
```

specifies all lines in the file with line numbers ≥ 10.

```
        A(,20)
```

specifies all lines in the file with line numbers ≥ 1 and ≤ 20.

```
        A(*F,20)
```

specifies all lines in the file with line numbers ≤ 20.

For devices such as magnetic tapes, the line-number range serves as a count of records. For example,

```
        *TAPE*(10,20)
```

specifies the next eleven records from the *current* record on the tape file. The first record read will be given the line number 10 (this is useful only when doing an indexed I/O operation to another file or device). If the user desires to position a tape to the tenth record from the current position, the CONTROL command should be used.

The subset of the file may be further restricted by specifying an increment. The increment indicates which lines between the beginning line number and the ending line number are to be included in the subset. The increment and the beginning line number are used according to the formula

$$b + n*i$$

to generate the list of lines which are included in the subset. Values for "n" are chosen such that the generated list of lines are within the beginning and ending lines of the subset. If "b" is omitted, the value of 1 is used in the formula. If "i" is omitted, all lines between the beginning line number and the ending line number are included in the subset. This is equivalent to having an increment of .001. For example,

```
        A(10,20,2)
```

specifies the lines in the file with line numbers 10, 12, 14, 16, 18, and 20. Any other lines between line 10 and line 20 are excluded from the subset.

```
        A(,20,2)
```

specifies all lines in the file with odd line numbers ≥ 1 and ≤ 20. In this case, the ending line number 20 is excluded from the subset.

```
        A(,,2)
```

specifies all lines in the file with odd line numbers ≥ 1.

A subset of the file also has a direction associated with it. This direction specifies the order in which lines are processed by a sequential read or write operation. The direction is determined by the sign of the increment and the presence of the @BKWD FDname modifier. If the increment is positive and the @BKWD FDname modifier is not present, the direction is forwards. The direction is also forwards if the increment is negative and the @BKWD FDname modifier is present. The direction is backwards if either (but not both) the increment is negative or the @BKWD FDname modifier is present. A forwards subset consists of the lines whose line numbers "#" are in the range

$b \le \# \le e$

A backwards subset consists of the lines whose line numbers are in the range

$b \ge \# \ge e$

For example,

```
A(10,1)@BKWD
```

specifies all lines in the file starting at line 10 through line 1, in that order.

```
A(10,1,-.001)
```

is the same as above.

```
A(10,1,-1)
```

specifies all lines in the file with line numbers 10, 9, ..., 1, in that order.

```
A(1,10,-.001)@BKWD
```

specifies all lines in the file with line numbers from 1 to 10, in that order. This is the same as

```
A(1,10)
```

If the ending line number is less than the beginning line number, the direction specified must be backwards; otherwise, the subset is null. Thus,

```
A(10,1)
```

specifies a null subset of file A even though the file may contain lines between lines 10 and 1.

The @BKWD I/O modifier and the BKWD modifier used with subroutine calls differ in their meanings. See the section "Input/Output Operations" and Appendix A for a further description of the @BKWD FDname and I/O modifiers.

## I/O MODIFIERS

Modifiers are used to modify the action of a specific I/O call or general I/O use. Modifiers may be used in I/O subroutine calls (INPUT, PRINT, READ, etc.), in macro calls setting up the corresponding I/O subroutine calls, or as parts of FDnames given in MTS commands. Modifiers control such functions as uppercase conversion, logical carriage control, machine carriage control, record trimming, etc.

When modifiers are used with FDnames, they are appended to the FDname. Each modifier consists of an at sign (@) followed by the modifier name. A modifier name may be preceded by a minus sign (–) or a not sign (~) to reverse its meaning. For example,

```
COPY A *SINK*@-CC
```

copies the file A to *SINK* with the –CC modifier specifying no logical carriage control.

Modifiers used with I/O subroutine calls are specified in one or more fullwords of modifier bits which are passed to the subroutine performing the I/O operation. This is further described below.

In general, there are three levels of precedence in the use of modifiers. In the first level of precedence are the modifiers specified on a call to one of the I/O subroutines. If the modifier is not specified by the subroutine call, the second level of precedence, in which the modifier name is a part of the FDname, applies. Note that the group of modifiers which can only control the action of a *specific* I/O call (for example ERRRTN, NOTIFY, NOATTN, and NOEC) are not valid at this level of precedence. If the action of the modifier is not specified by the second level, the third level of precedence, which consists of the default specifications, applies. The default specification depends upon the type of FDname referenced in the I/O call and the settings of global options. These defaults are given in the explanation of modifier bits below. Modifier specifications given at the first level of precedence override specifications given at the second and third levels (except for the BKWD modifier). Modifier specifications given at the second level override specifications given at the third level. This precedence scheme is illustrated in the diagram below. Each modifier is treated independently in the above precedence process.

```
Level 1: Subroutine Call Modifiers ______
                                         |
                                         |
Level 2: FDname Modifiers _______________|
                                         |
                                         |
Level 3: Defaults _______________________|
                                         |
                                         |
                                Effective Modifiers
```

The examples below illustrate the three levels for controlling the TRIM modifier. In each example, the solid line indicates the controlling level of precedence.

```
CALL INPUT(REG,LEN,32768,LNUM) ________
     (32768 specifies TRIM)           |
                                      |
INPUT=FYLE@-TRIM on RUN command ......|
                                      |
Default is -TRIM for file ............|
                                      |
                                      |
                                    TRIM

CALL INPUT(REG,LEN,0,LNUM) ............
     (0 makes no specification)       .
                                      .
INPUT=FYLE@TRIM _______________________
                                      |
Default is -TRIM for file ............|
                                      |
                                      |
                                    TRIM
```

```
CALL INPUT(REG,LEN,0,LNUM) ...........
     (0 makes no specification)       .
                                      .
INPUT=FYLE ...........................
                                      .
Default is -TRIM for file ____________
                                      |
                                      |
                                    -TRIM
```

The action of the modifiers specified on a subroutine call is controlled by one or more fullwords of modifier bits given as one of the parameters to the subroutine. The action of the modifiers on the subroutine call apply only to that specific call. There are two classes of modifiers.

(1) Bits 0-6 of the first fullword are referred to individually and each specifies the options for a *specific* I/O call. If the bit is set, the modifier's action is enabled. If the bit is not set, the default specification is used (which normally means the modifier action is disabled).

(2) Bit 7 of the first fullword (and all subsequent fullwords) is the continuation bit. If this bit is set, another fullword of modifier bits follows the current fullword.

(3) Bits 8-31 of the first fullword and bits 30-31 of the second fullword are referenced in pairs and specify options for a general I/O use. For each option, one bit is used as an "ON" bit and the other as an "OFF" bit. If either of the bits, but not both, is set, the modifier action is as specified (except for the BKWD modifier). If neither, or both, of the bits is set, indicating a "don't care" condition at this level of precedence, the modifier appended to the FDname is used. If there is no modifier name appended to the FDname, the default specification for the FDname type is used. The normal programming practice is to leave the modifier bits set to zero on the subroutine call and to append the modifier names to the FDname unless the program depends upon the modifier bits being set for a specific subroutine call. The following example illustrates how this might be done first in assembly language and then in FORTRAN:

```
        CALL INPUT,(REG,LEN,MOD,LNUM)
          .
          .
REG     DS   20F
LEN     DS   H
        DS   0F            Fullword alignment for MOD
MOD     DC   X'00004000'   No trimming of input lines
LNUM    DS   F
```

Note that if the subroutine call is set up by a macro call, the modifier names rather than the bits are used in the macro parameter list. Thus, the above example would become

```
INPUT  REG,LEN,@-TRIM,LNUM
```

The equivalent FORTRAN code is:

```
INTEGER*2 LEN
DATA MOD/Z00004000/
CALL INPUT(REG,LEN,MOD,LNUM)
```

The action of modifiers applied to the FDnames is controlled by the modifier name (preceded by "@") appended to the FDname. The action of the modifiers appended to the FDname apply to all I/O calls referring to *that use* of the file or device. If the modifier name is preceded with (−) or (~), the other bit of the bit pair is set, which negates the action of the modifier name. If implicit or explicit concatenation to another FDname occurs, the modifiers must be applied to both FDnames even if the FDnames are the same. If the user at a terminal is prompted for an FDname, the full FDname including the modifiers and line-number range must be given with each request. The order of modifier names appended to an FDname is unimportant. Some examples are:

```
FILE1@I@UC        Specifies indexed and uppercase
FILE2@-TRIM       Specifies no trimming
*SINK*@-CC        Specifies no logical carriage control
```

If the modifier action is also specified on a subroutine call, the modifier action applied to the FDname is overridden (except for the BKWD modifier).

If the BKWD modifier is given on a subroutine call, it specifies that the I/O operation is to be done in the reverse direction to that associated with the subset of the file. If the −BKWD modifier is given on a subroutine call, it is ignored.

The list of all the modifiers available is given below. The complete description of the action of each of these modifiers is given in Appendix A.

| *Name* | *Function (1st Fullword)* | *Value (hex)* | *Value (decimal)* |
|---|---|---|---|
| S | Sequential I/O operation | 00000001 | 1 |
| I | Indexed I/O operation | 00000002 | 2 |
| EBCD | EBCDIC translation (card reader) | 00000004 | 4 |
| BIN | Binary format (card reader) | 00000008 | 8 |
| LC | No conversion to uppercase | 00000010 | 16 |
| UC | Conversion to uppercase | 00000020 | 32 |
| NOCC | No logical carriage control | 00000040 | 64 |
| CC | Logical carriage control | 00000080 | 128 |
| −PFX | No line number prefix | 00000100 | 256 |
| PFX | Line number prefix | 00000200 | 512 |
| −PEEL | No line number peeling (input)<br>No line number returned (output) | 00000400 | 1024 |
| PEEL | Line number peeling (input)<br>Line number returned (output) | 00000800 | 2048 |
| −MCC | No machine carriage control | 00001000 | 4096 |
| MCC | Machine carriage control | 00002000 | 8192 |
| −TRIM | No trimming of trailing blanks | 00004000 | 16384 |
| TRIM | Trim trailing blanks | 00008000 | 32768 |

| | | | |
|---|---|---|---|
| –SP | No special processing | 00010000 | 65536 |
| SP | Special processing | 00020000 | 131072 |
| –IC | No implicit concatenation | 00040000 | 262144 |
| IC | Implicit concatenation | 00080000 | 524288 |
| FWD | Forward sequential | 00100000 | 1048576 |
| BKWD | Backward sequential | 00200000 | 2097152 |
| –ENDFILE | Suppress $ENDFILE recognition | 00400000 | 4194304 |
| ENDFILE | Force $ENDFILE recognition | 00800000 | 8388608 |
| FDUBCONT | Second word of modifiers present | 01000000 | 16777216 |
| NOPROMPT | No prompting for replacement | 04000000 | 67108864 |
| MAXLEN | Maximum input/output length given | 08000000 | 134217728 |
| NOEC | No explicit concatenation | 10000000 | 268435456 |
| NOATTN | Attention left pending | 20000000 | 536870912 |
| ERRRTN | Error return | 40000000 | 1073741824 |
| NOTIFY | Notify when FDUB changes | 80000000 | –2147483648 |

| *Name* | *Function (2nd Fullword)* | *Value (hex)* | *Value (decimal)* |
|---|---|---|---|
| –LOG | No logging via LOG command | 00000001 | 1 |
| LOG | Allow logging via LOG command | 00000002 | 2 |
| –MACRO | Do not invoke macro processor | 00000004 | 4 |
| MACRO | Invoke macro processor | 00000008 | 8 |
| –MFR | Macro flag not required | 00000010 | 16 |
| MFR | Macro flag required | 00000020 | 32 |

## CONCATENATION OF FDNAMES

Although the maximum size of any single file is limited (depending on the amount of file space available in the system and to the user), the amount of data that may be referred to by a given FDname is effectively unlimited, because several FDnames may be concatenated. This concatenation may be done either *implicitly* or *explicitly*.

### Implicit Concatenation

Implicit concatenation is indicated whenever a line of the form

```
$CONTINUE•WITH FDname
```

or

```
$CONTINUE•WITH FDname RETURN
```

(where • represents exactly one blank) is read from any file or device. The dollar sign ($) is required and must be the first character of the line. When such a line is encountered, reading continues with the file or device *FDname*; the "$CONTINUE WITH" line is not passed as data to the program issuing the read operation.

In the first case (without the RETURN), reading continues with the specified FDname, and the lines following the "$CONTINUE WITH" line in the original file or device are ignored. This is analogous to an unconditional transfer statement in a programming language. For example, if file A contains the line

```
    $CONTINUE WITH C
```

then the FDname A specifies the portion of file A up to the implicit concatenation line followed by the entire contents of file C. The second case (with RETURN) is analogous to a subroutine call in programming languages—after reaching the end of the FDname that was continued with, reading resumes with the line following the "$CONTINUE WITH" in the original file or device. For example, if file A contains the line

```
    $CONTINUE WITH C RETURN
```

then the FDname A specifies the portion of file A up to the implicit concatenation line, the entire contents of file C, and the remainder of file A after the implicit concatenation line, in that order.

Implicit concatenation may be disabled by the IC global option or by the IC FDname modifier. When implicit concatenation is disabled, a "$CONTINUE WITH" line is read as a data line. The IC global option may be set by the IC option of the SET command or by the subroutine CUINFO; its initial value is ON (implicit concatenation enabled). Thus, the command

SET IC=OFF

disables implicit concatenation globally ("$CONTINUE WITH" is read as a data line, by default). The IC modifier overrides the setting of the IC global option for all references to FDnames to which the modifier is appended. Thus,

```
    LIST PHYLE@-IC
```

allows the user to determine if the file contains any "$CONTINUE WITH" lines.

## Explicit Concatenation

Several files and/or devices may be chained together by using *explicit concatenation*. This is done by giving the names of the files or devices (with optional modifiers and/or line-number ranges) in the order desired, connected by plus signs. For example,

```
    A(1,100)+B
```

specifies lines 1 through 100 of file A followed by the entire contents of file B.

If two or more consecutive simple FDnames in an explicit concatenation are identical, all but the first may be omitted. For example,

```
    A(1,1)+A(10)
```

may be abbreviated to

```
        A(1,1)+(10)
```

Each of these specifies line 1 of file A followed by the remainder of file A starting at line 10.

If a member FDname of an explicit concatenation uses implicit concatenation (i.e., contains a "$CONTINUE WITH" line), the implicit concatenation affects only that member of the explicit concatenation; the remaining members are not ignored. For example, if file A contains the line

```
        $CONTINUE WITH C
```

the FDname A+B specifies the portion of file A up to the implicit concatenation line, the entire contents of file C, and the entire contents of file B, in that order. Note that the FDname in the "$CONTINUE WITH" line may also be an explicit concatenation.

## FDNAMES

The term *FDname* can now be completely defined as either

(1) a simple FDname with an optional line-number range and/or modifiers, or

(2) an explicit concatenation of two or more simple FDnames, each with an optional line-number range and/or modifiers.

In either case, the files or devices specified by the FDnames may contain "$CONTINUE WITH" lines. In the cases where a restriction must be made to either a single file or device, the term FDname will not be used.

## INPUT/OUTPUT OPERATIONS

Input and output using files may be done either as *indexed* operations or as *sequential* operations. An indexed operation uses a line number to directly access a specific line. A sequential operation accesses the first line or the next line following the previous line that was read or written. Many devices, such as card readers and line printers, are intrinsically sequential. For these devices, indexed operations are not defined. Similarly, indexed operations have no meaning for a file whose organization is sequential.

The type of operation used for input and output depends on the setting of a pair of I/O modifier bits. These bits specify whether the operation is to be indexed or sequential. These bits may be set either by the SEQUENTIAL (S) or INDEXED (I) FDname modifiers or by the corresponding I/O modifier bits on an input/output subroutine call. The complete details of I/O modifier bit processing are given in Appendix A to this section.

For all input or output operations with *line* files, the system maintains a *current line pointer* that points just past the last record read or written, and a *current line number* that contains the line number of the last record read or written. When a sequential I/O operation is initiated, the current line pointer is used to determine the record that is to be read or written. After the operation has been completed, the current line pointer is set just past the line that was read or written. If the I/O operation is done in the backwards direction, the line pointer is set just before the line that was read or written (i.e., just past the line going in the backwards direction). In addition, the current line number is updated.

The BKWD FDname modifier, the sign of the increment specified in a line-number range, and the BKWD I/O modifier bit specified on an I/O subroutine call interact to determine the direction of an I/O operation. The BKWD FDname modifier and the sign of the increment determine a direction for the subset of the file specified by the line-number range. The rules for determining this direction are given in the section "Line-Number Ranges." If the direction is forwards, the I/O operation is done in the forwards direction; if the direction is backwards, the I/O operation is done in the backwards direction. The setting of the BKWD I/O modifier bit on an I/O subroutine call may be used to further control the direction of the I/O operation. If the BKWD modifier bit is set, the direction of the operation is reversed. The following table illustrates the use of these modifiers in controlling the direction of the I/O operation.

| *FDname Specification* | *BKWD I/O Modifier Bit* | *Direction of I/O Operation* |
|---|---|---|
| A(1,10) | 0 | Forwards |
| A(1,10) | 1 | Backwards |
| A(10,1,–1) | 0 | Backwards |
| A(10,1,–1) | 1 | Forwards |
| A(1,10,–1)@BKWD | 0 | Forwards |
| A(1,10,–1)@BKWD | 1 | Backwards |

## Sequential Operations with Line Files

If the I/O modifier bits specify a sequential operation or if they specify neither a sequential nor an indexed operation (the default), a sequential operation is performed.

A sequential operation specifies that the "next" record is to be read or written. The "next" record is determined by the value of the current line pointer. The setting of the BKWD I/O modifier bit indicates whether "next" is taken in the ascending or descending order. If the modifier bit is not set and the direction of the subset is forwards, ascending order is used; if the modifier bit is set and the direction of the subset is forwards, descending order is used. For a read operation, "next" means the record follows in ascending (descending) line number order from the current value of the line pointer. If an increment is explicitly given with the FDname, the line read is the first line in the file past the current line pointer that has a line number which is a multiple of the increment from the beginning line number. For a write operation, "next" means the first line number past the current line pointer that is a multiple of the increment from the beginning line number. If no increment is specified, a default of 1 is used for write operations.

For a sequential read operation with a line file, the line-number range determines which lines of the file may be read. The read operation begins with the beginning line number "b". If "b" is omitted, line number 1 is used. Note that this is not necessarily the first line of the file. All lines in the subset of the file as specified by the line-number range may be read. Any lines not in the subset, even if they are between the beginning line number and the ending line number, are not read (i.e., they are skipped). If an attempt is made to read a line in the subset which does not exist in the file, reading continues with the next line in the subset. For example, the command

```
    LIST A(10,20,2)
```

initiates read operations for lines 10, 12, 14, 16, 18, and 20 from line file A. Each line of this subset that exists in the file is listed. Lines in the subset that do not exist in the file and all other lines in the file that are not in the subset defined by (10,20,2) are not listed.

If a sequential read operation is not the first I/O operation on a line file (e.g., if an indexed read or any write operation was the last operation completed), the line read is the first line past the current line pointer in the appropriate direction that has a line number that is a multiple of “i” past the beginning line number “b”.

For a sequential write operation to a line file, the line-number range determines which lines of the file may be written. The first line written is the beginning line number “b”. If “b” is omitted, line number 1 is used. All lines in the subset of the file as specified by the line-number range may be written. Any lines not in this subset, even if they are between the beginning line number and the ending line number, are not changed. If an attempt is made to write to a line not in the subset, an end-of-file condition is generated. For example, the command

```
    COPY *SOURCE* A(10,20,2)
```

reads 6 lines from *SOURCE* and writes them to lines 10, 12, 14, 16, 18, and 20 of file A. Any other lines in the file that are not in the subset defined by (10,20,2) are not changed. When an attempt is made to write a seventh line, an end-of-file occurs.

## Indexed Operations with Line Files

An indexed operation is performed if the I/O modifier bits specify an indexed operation.

For an indexed operation, a line number must be given which specifies the line to be read by a read operation or the line to be written by a write operation. The line-number range specified on the FDname is checked to determine whether the given line number is in the subset of the file that may be read or written. If the line is not within this subset or the line contains the “$ENDFILE” delimiter, an end-of-file condition occurs. The rules for defining the subset of the file specified by the line-number range are given in the earlier section “Line-Number Ranges.” After an indexed operation is completed, the current line pointer and the current line number are updated. The current line pointer will point after (in the direction of the subset of the file) the line read or written unless the BKWD modifier was specified on the I/O call, in which case it will point before the line read or written.

Implicit concatenation with indexed operations is handled as follows. For an indexed read operation, if the line selected is a “$CONTINUE WITH” line and implicit concatenation is enabled, the concatenation will occur and the same line number will be used to read the new file or device specified by the FDname in the implicit concatenation line.

Indexed operations are not allowed with sequential files unless the SEQFCHK option is OFF. An error comment will be generated if an indexed operation is specified with an FDname; a return code of 20 will be given if an indexed operation is attempted with a sequential file in an I/O subroutine call.

## Sequential Operations with Sequential Files and Devices

For I/O operations involving sequential files or devices, the test to determine whether a given line is in the subset specified by the line-number range is the same as for line files. However, since lines in a sequential file or device have no line numbers inherently associated with them, the system “generates” line numbers internally using the beginning line number and the increment specified by the user and tests if the result exceeds the ending line number. The data lines can only be read or written sequentially, either forwards or backwards (depending on the inherent nature of the file or device).

Just as the current line pointer is maintained for line files, a current read pointer and a current write pointer are maintained for sequential operations on sequential files. In general, these two pointers are independent of each other, i.e., the read pointer controls only read operations and the write pointer controls only write operations. The read pointer initially points before the first line of the file and the write pointer initially points after the last line of the file. These pointers are updated appropriately after a sequential read or write operation to point to the next line to be read or written. For a sequential read operation, the read pointer may move either forwards or backwards depending on the setting of the BKWD I/O modifier bit.

For a sequential read operation from a sequential file or device, the line read is the next line after the current position of the read pointer in the appropriate direction or the line at the current position of the device (e.g., the current position of a magnetic tape or card reader).

For a sequential write operation to a sequential file or device, the line written is the next line after the current position of the write pointer (thus, new lines are always written at the end of the file, by default), or the line at the current position of the device (e.g., the current position of a magnetic tape or line printer). If the write pointer does not point to the end of the file, a sequential write operation will truncate the remainder of the file after the line written unless the SP modifier is specified (see below).

For sequential files, a beginning line number other than 1 is not allowed unless the SEQFCHK option is OFF. For devices, any beginning line number is legal.

The read and write pointers may be explicitly positioned to a particular line in a sequential file via the NOTE and POINT subroutines. This is further described in Appendix B to this section.

## The SP Modifier with Files

The SP (SPECIAL) modifier is a file- or device-dependent modifier that may be used to further control I/O operations. Only the use of SP as applied to I/O operations with files is described here. For details on using SP with terminals operating through the UMnet/Michnet Computer Network, see the appropriate sections of *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004, and *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019.

For read operations to both line files and sequential files, the SP modifier is used to skip to the next line without transmitting data. The current line number and current line pointer (or read pointer for sequential files) are updated. However, no data is read into the buffer provided by the I/O subroutine and a zero is returned for the length of the line read.

For write operations to sequential files, the SP modifier is used to replace the current line without truncating the file. The line at the current position of the write pointer is replaced by a new line. If the new line is shorter than the current line, it is padded on the right with blanks; if the new line is longer than the current line, it is truncated to the length of the current line. In either case, an error message is generated. If the write pointer is at the end of the file, no write operation is performed and an error message is generated. The SP modifier is ignored for write operations to line files.

## Explicit Concatenation with I/O Operations

The processing of the next member of an explicit concatenation is started whenever an end-of-file occurs for a read or write operation unless the NOEC modifier was specified. This occurs when the physical end of the file or device is encountered or when the line number specified for the operation is not within the subset of the file specified by the line-number range. Specifying NOEC inhibits the transfer to the next member of the explicit concatenation.

For an indexed read operation, an end-of-file returned to the program (when NOEC is not specified) indicates that none of the members of the concatenation contains the line specified for the indexed operation. A successful indexed read operation selects the specified line from the first member of the concatenation that contains that line.

For a sequential write operation, the line-number range specifications may be used to control the flow of data into several files or devices. For example,

```
COPY A B(1,10)+*SINK*(1,10)+B(11)
```

copies from file A (starting at line 1) 10 lines into line file B (starting at line 1), followed by 10 lines to *SINK*, followed by the remainder of file A into file B (starting at line 11).

If the file that is a member of an explicit concatenation is unable to expand during a write operation, the write operation transfers to the next member of the concatenation to write the line.

## USE OF FDNAMES

There are two ways that FDnames may be used for input/output operations. Most commonly, a program reads or writes information without knowing the names of the specific files or devices being used. This case is handled with logical I/O units. Alternatively, the program reads from or writes to a specific file or device, whose name is either built into the program or is obtained as data from the user. This case is handled with FDUB-pointers.

A summary of the system subroutines that use file names, logical I/O units, and FDUB-pointers is given in the section “Subroutines That Use Files and Devices” in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

### Logical I/O Units

When a program is coded, the names of the files and devices to be used for input and output are normally unknown; hence, it is impossible to specify them in the program source statements. Even if the names were known, it would be inconvenient to specify them in the program, since this would require retranslation every time a file or device name was changed. Thus, it is desirable to specify the location of the data at execution time rather than at translation time. To do this, a *logical I/O unit* is used. A logical I/O unit is a symbolic name which is used in a program to specify the source of data for input or the destination of output information. A logical I/O unit does not name a specific file or device; it simply serves as a reference. When a program is executed, it is necessary to first specify, for each logical I/O unit used by the program, the actual file or device to be used. For example, in FORTRAN, the statement

```
READ (5,100) P
```

requests input from logical I/O unit 5. Before this statement can be executed, the user must specify a file or device name to be used whenever logical I/O unit 5 is referenced. This is normally done on the RUN command by specifying a keyword of the form

unit=FDname

For example, to execute the program in the file PROGRAM which contains the above FORTRAN statement, the command

```
    RUN PROGRAM 5=DATAFILE
```

may be given; whenever the program requires input from logical I/O unit 5, data is read from the file DATAFILE.

The names of the logical I/O units and acceptable synonyms are given below. Minimum abbreviations for the synonyms are underlined. The synonyms will be used in this volume when appropriate.

| *Units* | *Synonyms* |
|---|---|
| SCARDS | INPUT |
| SPRINT | PRINT |
| SPUNCH | OBJECT |
| SERCOM | |
| GUSER | |
| 0 through 99 | |

For each of these, except 0 through 99, there is a subroutine of the same name to perform input and/or output on this unit. For 0 through 99, the subroutines READ and WRITE are provided (these subroutines may also be used with the named logical I/O units). These subroutines can be called from both assembly-language programs and higher-level language programs. For example, calling the subroutine INPUT (SCARDS) from a program causes a record to be read from the logical I/O unit INPUT (SCARDS). These subroutines are described in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

Since it is desirable to reduce the amount of typing necessary to specify the information required on a RUN command, some of the logical I/O units have default specifications. The following defaults are provided if no logical I/O unit assignment is given on the RUN command:

| *Units* | *Defaults* |
|---|---|
| INPUT | *SOURCE* |
| PRINT | *SINK* |
| OBJECT | *PUNCH*[1] |
| SERCOM | *MSINK* |
| GUSER | *MSOURCE* |
| 0–99 | none |

[1]*PUNCH* is the default for batch jobs only if the global card estimate is greater than 0; otherwise, there is no default. OBJECT (SPUNCH) is not defaulted for terminal jobs.

For example, if on a RUN command the logical I/O unit SCARDS is not specified as a particular file or device, it is assigned by default to the current input *SOURCE*. Since most of the translators in MTS use INPUT for source program input and PRINT for compilation listing output, it is often unnecessary to specify these logical I/O units when running the translators, especially in batch mode. For both batch and conversational mode, it is usually necessary to specify the logical I/O unit for the translator output of the resulting object deck. Logical I/O units 0 through 99 have no default specifications in MTS. There are, however, default specifications for units 5 and 6 within the

FORTRAN I/O library routines during program execution.

The following example given in assembler language illustrates how to perform an input/output operation using subroutines and logical I/O units. This example reads the name of a file from logical I/O unit 5 and performs an indexed write operation to place the line “***DATA LINE***” into line 10 of that file. For the complete descriptions of the subroutines used in this example, see *MTS Volume 3*.

```
        READ   5,FNAME,EXIT=ERROR              Read file name
        CALL   SETLIO,(UNIT,FNAME)             Assign file to I/O unit
        LTR    15,15                           Check for error
        BNE    ERROR
        WRITE  UNIT,REGION,@I,10000,EXIT=ERROR Write the data line
         .
         .
FNAME   DC     CL18' '
UNIT    DC     CL8'PRINT '
REGION  DC     C' ***DATA LINE***'             Data line
```

The following example is the FORTRAN equivalent of the example given above.

```
      INTEGER*4 MOD/2/,LNUM/10000/
      INTEGER*2 LEN/16/
      LOGICAL*4 REGION(4)/' ***','DATA',' LIN','E***'/
      LOGICAL*1 FNAME(18)
      LOGICAL*4 UNIT(2)/'PRIN','T   '/
      READ (5,50) FNAME
   50 FORMAT (18A1)
      CALL SETLIO(UNIT,FNAME,&100)
      CALL WRITE(REGION,LEN,MOD,LNUM,UNIT,&100)
        .
        .
  100 (print error comment)
```

The following example is the C87 equivalent of the example given above.

```
#include <mts.h>
main()
  { char fname[18];
    short inlen;
    static const short outlen=16;
    static const int unit5 = 5;
    static const unsigned int inmods = 0,
                              outmods = _INDEXED;
    int lineno, rc;

    MTSREAD(fname,&inlen,&inmods,&lineno,&unit5,_retcode rc);
    if(rc!=0) goto error;
    SETLIO("PRINT   ",fname,_retcode rc);
    if(rc!=0) goto error;
    lineno = 10000;
    MTSWRITE(" ***DATA LINE***",&outlen,&outmods,&lineno,"PRINT   ",
             _retcode rc);
    if(rc!=0) goto error;
    ...
error: /* print error message */
}
```

**FDUB-Pointers**

It is occasionally necessary for a program to read or write on a specific file or device whose name is not, or cannot, be assigned to a logical I/O unit before or after program execution begins. The FDname may be built into the program (as for example a fixed file of error messages) or may be input data to the program. The program uses a fullword quantity called a *FDUB-pointer* to refer to this FDname when doing input or output on the file or device. A FDUB-pointer is the location of a *F*ile or *D*evice *U*sage *B*lock, maintained by MTS to control the use of that file or device. When the subroutines READ or WRITE are called, the FDUB-pointer is used as a parameter instead of a logical I/O unit name or number.

A FDUB-pointer is obtained by calling the subroutine GETFD (see *MTS Volume 3*) giving it the FDname for the file or device. The FDUB-pointer returned by GETFD can then be used in calls to READ or WRITE to do input or output. It can also be used in calls to a number of other subroutines (see table below). When the program is finished with the FDUB-pointer, the FDUB can be released by calling the subroutine FREEFD.

The following example given in assembler language illustrates how to perform an input/output operation using subroutines and FDUB-pointers. This example reads the name of a file from logical I/O unit 5 and performs an indexed write operation to place the line "***DATA LINE***" into line 10 of that file. The subroutine GDINFO is also to determine if the file specified is a line file. For the complete descriptions of the subroutines used in this example, see *MTS Volume 3*.

```
        READ  5,FNAME,EXIT=ERROR      Read file name
        LA    1,FNAME
        CALL  GETFD                   Get FDUB-pointer for file
        LTR   15,15                   Check for error
        BNE   ERROR
        ST    0,FDUB                  Save FDUB-pointer
        CALL  GDINFO                  Get file information
        LTR   15,15                   Check for error
        BNE   ERROR
        USING GDDSECT,1               Pointer for GDINFO area
        CLC   GDTYPE,=C'FILE'         Check for line file type
        BNE   ERROR
        SR    0,0
        CALL  FREESPAC                Release the GDINFO area
        DROP  1
        WRITE FDUB,REGION,@I,10000,EXIT=ERROR  Write the data line
        L     0,FDUB
        CALL  FREEFD                  Release the FDUB-pointer
         .
         .
FNAME   DC    CL18' '                 File name
FDUB    DS    F
REGION  DC    C' ***DATA LINE***'     Data line
        COPY  *GDINFODSECT            GDINFO area copy section
         .
         .  (GDDSECT has symbol GDTYPE)
```

The following example is the FORTRAN equivalent of the example given above.

```
      EXTERNAL GETFD,FREEFD,FREESP,GDINF
      INTEGER*4 ADROF,FDUB,GDAREA(11),GDTYPE,FILE/'FILE'/
      INTEGER*4 MOD/2/,LNUM/10000/
      LOGICAL*4 REGION(4)/' ***','DATA',' LIN','E***'/
      LOGICAL*1 FNAME(18)
```

```
            INTEGER*2 LEN/16/
            EQUIVALENCE (GDAREA(2),GDTYPE)
            READ (5,50) FNAME
        50  FORMAT (18A1)
            CALL RCALL(GETFD,2,0,ADROF(FNAME),1,FDUB,&100)
            CALL GDINF(FDUB,GDAREA,&100)
            IF (GDTYPE.NE.FILE) GOTO 100
            CALL WRITE(REGION,LEN,MOD,LNUM,FDUB,&100)
            CALL RCALL(FREEFD,1,FDUB,0,&100)
              .
              .
       100  (print error comment)
```

The following example is the C87 equivalent of the example given above.

```
    #include <mts.h>
    #include <string.h>
    main()
      { char fname[18];
        short inlen;
        static const short outlen=16;
        static const int unit5 = 5;
        static const unsigned int inmods = 0,
                                  outmods = _INDEXED;
        void *fdub;
        struct GDDSECT *gdreg;
        int lineno, rc;

        MTSREAD(fname,&inlen,&inmods,&lineno,&unit5,_retcode rc);
        if(rc!=0) goto error;
        fdub = GETFD(fname,_retcode rc);
        if(rc!=0) goto error;
        gdreg = GDINFO(fdub,_retcode rc);
        if(rc!=0) goto error;
        if(memcmp(gdreg->GDTYPE,"FILE",4)!=0) goto error;
        FREESPAC(0, gdreg);
        lineno = 10000;
        MTSWRITE(" ***DATA LINE***",&outlen,&outmods,&lineno,&fdub,
                 _retcode rc);
        if(rc!=0) goto error;
        ...
    error: /* print error message */
    }
```

## ERROR PROCESSING

If the FDname for a file or device is incorrect (e.g., if it contains an illegal modifier or line-number specification), an initial error comment is issued when the FDname is accessed (e.g., when a RUN command is issued). If, however, the FDname is correct but the intended use is not (e.g., if an attempt is made to read from a file that does not exist), then the error condition is not recognized until the first time an attempt is made to use the file or device (e.g., when the subroutine READ or WRITE is called).

In either case, when an error condition exists, error comments are printed as described below when the first attempt is made to use the file or device, and then

(1) in batch mode, a return is made to MTS command mode or debug mode, and (in MTS command mode) the succeeding cards in the user's deck are skipped until an MTS

command ($ in column 1) is found, or

(2) in conversational mode, the message

```
        [error message]
        Enter a new file/device name, "CANCEL", or "HELP".
```

is printed. At this point, the user may either enter a replacement FDname (including the modifiers and/or line number ranges), or enter CANCEL in which case MTS will return to MTS command or debug mode, or else enter HELP in which case MTS will give an expanded error message.

If the file or device in error is part of an explicit concatenation, the replacement file or device replaces only the FDname printed in the message in the same relative position in the concatenation.

The four error messages that most often occur at "first use" time are as follows:

| | |
|---|---|
| "FDname" does not exist. | This means that "FDname" was interpreted as a file name and no file by that name exists. |
| "FDname" is not available. | This means that "FDname" is an existing file or *...*, but it cannot be accessed at the present time (e.g., because it is a file on a volume that is not available). |
| File "FDname" - required access not allowed. | This means that "FDname" is an existing file, but the access appropriate to the request is not permitted. |
| Invalid file specification "FDname". | This is produced in all other cases. This message is often accompanied by another message giving more information. |

As an example, consider the following portion of a terminal session. The numbers in brackets at the right are for reference only.

```
    #copy  *source* -a@snark                              (1)
    #Illegal FDname modifier                              (2)
    >line one                                             (3)
    >"-A@SNARK" is invalid.                               (4)
    >Enter a new file/device name, "CANCEL", or "HELP".   (5)
    ?cancel                                               (6)
    #                                                     (7)
```

In (1), the user enters a COPY command with an illegal modifier. The error message is printed immediately (2). However, the error occurred in the second FDname of the COPY command, which was not referred to until after the first line was read from *SOURCE* in (3). At that point, the illegal name is first used, the "first use" error message is printed (4), and the user is prompted for replacement (5). In (6), the user enters CANCEL, which causes return to MTS command mode ("#" prefix character) in (7).

## CREATING FILES

The category, organization, and initial and maximum size of a file are all established at the time the file is created. Files are created by using the CREATE command or by calling the CREATE subroutine from a program. The syntax of the CREATE command is given in the command description in this volume. The CREATE subroutine is described in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003. The following discussion applies to the specification of the file characteristics when using the CREATE command.

The CREATE command has the basic form:

```
CREATE filename
```

This command creates a file with the name "filename" (unless there already exists a file of the same name, or the user has exceeded the file-space allotment). There are several options that can be specified in the form of keyword expressions.

| | |
|---|---|
| SIZE=n | n is number of average (50 byte) lines |
| SIZE=nP | n is number of pages (1 page = 4096 bytes) |

For most files, this parameter is not needed. The default size for permanent files is the smallest possible (one page) which is large enough for about 60 average length lines. The default size for temporary files is 10 pages. Although files are expanded automatically (as explained later), they are expanded by 10% of their current size each time; consequently, the cost of this convenience is higher than initially creating the file with the necessary size. Thus, for larger files, it is desirable to specify an estimated size. When estimating size in terms of pages, it should be noted that there is a certain minimum overhead in addition to the data itself for each line or logical record in the file. The minimum overhead for each logical record depends on the organization of the file as follows:

| *Type* | *Minimum Overhead* |
|---|---|
| LINE | 10 bytes/line |
| SEQ | 6 bytes/line |

Information on the internal structure of files and approximate formulas for the size required for a file is given in Appendix C to this section.

Files are expanded automatically by the system if the following conditions are satisfied:

(1) The user has not exceeded the disk space allocation and allowed overage.

(2) Space is available on the direct-access volume on which the file was originally created.

(3) The maximum inherent size of a file (32,767 pages) has not been exceeded.

(4) The user-specified maximum expandable size (MAXSIZE) has not been exceeded.

If the first condition is not met, the error comment

```
File "filename": Allocated space exceeded
```

is printed. If any of the last three conditions is not met, the error comment

```
File "filename": Expansion unsuccessful
```

is printed. If a file cannot be expanded for any of the above reasons, it is left in a state such that the I/O operation which incurred the error should be repeated, if possible, after correcting the cause of the error. For line files, the following types of operations will have the indicated results, following a failure to expand a file:

| *Operation* | *Result* |
|---|---|
| Insertion | Line is not inserted |
| Replacement | Line is deleted, but new line not inserted |
| Deletion | Line is deleted |

For the replacement and deletion operations, the file space occupied by the deleted line may not be available for future allocation to subsequent lines.

| | |
|---|---|
| TYPE=LINE | Line file |
| TYPE=SEQ | Sequential file |

This specifies the organization of the file being created. The default is LINE.

| | |
|---|---|
| MAXSIZE=n | n is number of average (50 byte) lines |
| MAXSIZE=nP | n is number of pages (1 page = 4096 bytes) |

This specifies the maximum expandable size for the file. The default is 32767 pages.

## PUTTING INFORMATION INTO A FILE

There are basically two different methods for initially putting a set of lines from *SOURCE* into a file. The following descriptions are equally applicable to batch and conversational mode.

The first method is to use a COPY command, copying from the source stream *SOURCE* into the file, as for example,

```
COPY  *SOURCE* F
    .
    .
    .  lines to be put in file
    .
    .
{end-of-file signal}
```

where the end-of-file signal in batch must be a “$ENDFILE” delimiter (or the physical end of the card deck), or from a terminal, it can be either an end-of-file control character for that terminal or a “$ENDFILE” delimiter explicitly typed in.

With this method, all lines are transmitted into the file as typed except lines which consist of “$ENDFILE” or “$CONTINUE WITH”. The “$ENDFILE” line is recognized as an end-of-file delimiter and terminates the copy operation. For example, in order to get “$ENDFILE” lines into the file F, the command

```
COPY *SOURCE*@-ENDFILE F
```

should be used. This causes “$ENDFILE” lines to be treated as data lines. This also means that “$ENDFILE” cannot be used to terminate the copy. If the copying is being done from a terminal, an end-of-file control character must be used to terminate the copy. But if the copying is being done in batch, the COPY command should be terminated by

```
$CONTINUE WITH *DUMMY*
```

This is an alternate method of generating an end-of-file condition. If this method is not used, the COPY command must be the last command in the batch input stream, since all the remaining input lines will be copied into the file.

Note that implicit concatenation is normally enabled. If a “$CONTINUE WITH” line is to be copied rather than taking effect, implicit concatenation must be disabled for the copy, for example,

```
COPY *SOURCE*@-IC F
```

The second method is to use the MTS File Editor INSERT command, for example,

```
EDIT F
INSERT 1
   .
   .
   .  lines to be put into file
   .
   .
{end-of-file signal or null line}
STOP
```

For further details, see *MTS Volume 18: The MTS File Editor*, Reference R1018.

## MAKING CHANGES TO A FILE

There are two basic methods of making changes to a file: using the MTS commands to make changes to the file on a line-by-line basis, and using the MTS File Editor to make changes either on a line basis or on a context basis.

### Changes Using MTS Commands

MTS commands can be used to change one line of a line file, a group of lines of a line file, or to empty or destroy an entire file. Sequential files can be changed only by adding on to the end of the file, or by emptying or destroying the file. Single lines or groups of lines in a line file can be changed by using the same basic two methods that may be used to put information into a file.

The COPY command can be used to copy lines or groups of lines from a file to itself or another file. Thus,

```
COPY  A+B C
```

enters the contents of A, followed by the contents of B, into file C (or if C is sequential, adds to the end of C). If either A or B are line files, then the "contents" are only the lines with numbers greater than or equal to 1, which is not necessarily the entire contents of the file. If C is a line file, lines that are not replaced by the copy operation are not affected. Thus,

```
COPY A(1,5)+(99,99)+(1000) B
```

copies from A lines 1 through 5, 99, and 1000 through the end of A into B, starting at line 1 in B, if B is a line file, or adding to the end of B, if B is sequential. File A must be a line file in this example. Note that if the example were

```
COPY A(1,5) B
```

then A could be either a line file or a sequential file.

```
COPY A(10,15) A(100)
```

makes a copy in A of the lines in the range of 10 through 15, starting at line 100. Remember that the range of 10 through 15 is the same as 10.000 through 15.000 and can contain anywhere from 0 to 5001 lines. The increment between the copied lines starting at line 100 is 1 (the default).

A copy operation from a file to itself specifying overlapping line-number ranges can be done, but should be used with care. For example,

```
COPY A(1,9) A(2)
```

results in line 1 being replicated in lines 2 through 10, since line 1 is copied into line 2, and then line 2 (which is now a copy of line 1) is copied into line 3, etc.

Once a file has been created, it may be emptied, truncated, expanded, renumbered (line files only), renamed, or destroyed at any time. This applies both to permanent files and to temporary files. Permanent files exist until the user destroys them; temporary files exist until the user destroys them or signs off.

An exact copy of a file can be made by issuing the DUPLICATE command. For example,

```
DUPLICATE A B
```

writes an exact copy of the file A into the file B. The file B will be created automatically if it does not exist with the same characteristics as file A; if it does exist, it will be emptied first. Conversational users are prompted for confirmation before the file is emptied.

To empty a file, the command

EMPTY filename

should be issued. This discards the contents of the file, but preserves *all* storage space allocations (including expansions). Future references to the file will reuse the file space. If a command to empty a permanent file is issued from a terminal, the user must confirm that the contents of the file are to be discarded. The message printed is

```
File "filename" is to be emptied. Please confirm:
```

The user should respond with "OK" to empty the file. A response of "NO" or "CANCEL" cancels the command. Any other response is an error and the message is repeated. Confirmation is not requested for temporary files, nor for any files in batch runs.

It should be noted that the EMPTY command empties the whole file; parts of a file cannot be emptied by attaching a line-number range to the file name in the command. Thus,

```
        EMPTY A(10,50)
```

is an error, whereas

```
        EMPTY A
```

empties all of file A. To partially empty a file, the parts to be saved should be copied to another file. The original file may then be destroyed and the new file may be renamed to the original name.

To truncate a file, the command

> TRUNCATE filename

should be issued. Any unused space at the end of the file is deallocated. This decreases the size of the file and decreases the storage charge for the space used by the file. In the case of a line file, truncating does *not* compact or optimize the space used by the actual data in the file.

To change the absolute size of a line file, the user may issue the command

> CONTROL filename SIZE={n | nP}

or, to add to or subtract from the current file size

> CONTROL filename SIZEINC=[±]{n | nP}

To change the absolute maximum size of a line file, the user may issue the command

> CONTROL filename MAXSIZE={n | nP}

or, to add to or subtract from the maximum file size

> CONTROL filename MAXSIZEINC=[±]{n | nP}

To renumber a *line* file, the command

> RENUMBER filename

should be issued. The line numbers in the file are renumbered. By default, the entire file is renumbered starting at line number 1 and by increments of 1. An explicit line-number range to be renumbered as well as an explicit new beginning line number and increment may be specified (see the RENUMBER command description).

To rename a file, the command

> RENAME filename1 filename2

should be issued. The file “filename1” is renamed to “filename2”. If this command is issued from a terminal and “filename1” is a permanent file, confirmation is requested. The message is:

```
File "filename1" is to be renamed as "filename2". Please confirm:
```

The user should respond with “OK” to rename the file; confirmation is not requested for temporary files, nor for any files renamed during batch runs. A temporary file may be changed to a permanent file, and vice versa, by renaming it.

To destroy a file, the command

DESTROY filename

should be issued. All space allocated for the file is deallocated. If this command is issued from a terminal and the file is a permanent file, confirmation is requested:

```
File "filename" is to be destroyed. Please confirm:
```

The user should respond with “OK” to destroy the file. A response of “NO” or “CANCEL” cancels the command. Any other response is an error and the message is repeated. Confirmation is not requested for temporary files, nor for any files in batch runs.

## Changes Using the File Editor

The MTS File Editor can be used to edit files on a line-number basis. Single lines or a range of lines may be deleted, new lines may be inserted between existing lines, and existing lines may be replaced by new lines. The File Editor can also be used on a context basis, scanning for lines with certain character strings in them, replacing characters in a line with other characters, etc. For example, on a line-number basis, the following sample edit session replaces line 2, deletes line 7, and inserts two lines after line 8 of the file being edited (the first character is the prefix character; input from the user is in lowercase, File Editor output is in uppercase):

```
:replace 2 'new second line'
:      2     NEW SECOND LINE
:delete 7
:insert 8
?eighth and a quarter
?eighth and a half
? {null input line}
:
```

On a context basis, the following example illustrates a scan through the file searching for the characters “ABC#D”:

```
:scan /file 'abc#d'
```

and then the following command alters “C#D” to “XYZ”:

```
:alter /file 'c#d'xyz'
```

The File Editor is invoked from MTS command mode by the EDIT command. The Editor has its own command language. Complete details on using the Editor are given in *MTS Volume 18: The MTS File Editor*, Reference R1018.

### Changes Using CDUPATE

The program *CDUPDATE may be used to update a file. *CDUPDATE is a "context-directed" program that applies updates to an "old master file" to produce a new "updated master file." It is typically used in maintaining a program in the form of a "base-level source" and a set of updates to be applied to produce the current version. *CDUPDATE is further described in *MTS Volume 2: Public File Descriptions*, Reference R1002.

## RESTORING THE CONTENTS OF A FILE

Occasionally, a user will make changes to a file and then regret it. The user should realize that the File Editor makes changes to a file as the commands are entered; it does not make changes to a working copy. Thus, it is prudent to either make a copy of the file as backup before editing it, or else to use the Editor's checkpoint/restore facility. This facility allows the user to establish a base from which all future edit commands are checkpointed. The RESTORE edit command may then be used to restore all or a portion of the file back to its state when it was last checkpointed. The most recent change to the file can always be reversed by the UNDO command (see *MTS Volume 18: The MTS File Editor*, Reference R1018 for details).

All files are periodically saved on magnetic tape by ITD. This "file-save" is done on a partial basis daily and on a complete basis weekly. The daily file-save saves all files that were changed during the preceding day. The weekly file-save saves all files that exist in the system at the time of the file-save.

The user may restore a file which was accidentally changed, emptied, or destroyed from the file-save tapes by running the program *RESTORE. Files are available only for 6 weeks. See *MTS Volume 2: Public File Descriptions*, Reference R1002, for details.

## DISCOVERING THE CHANGES TO A FILE

Often a user, after extensively working on a file, ends up with a "current" copy and an "original" copy, and would like to know how they differ. In particular, the user would like a list of changes that must be applied to the original copy to make it into a current copy.

There are three programs available in MTS public files that generate "changes" in this manner: *UNEDIT, *APC, and *COMPARE. All differ on either the basis for deciding if something is changed, or the method for presenting the changes, or both. The remainder of this section gives examples illustrating how these programs differ. The complete descriptions of these programs are given in *MTS Volume 2: Public File Descriptions*, Reference R1002.

The program *UNEDIT generates MTS File Editor commands. Both the current and original copies should be line files and must not contain lines longer than 255 bytes. The entire contents of the lines are sequentially compared. The line numbers are used for the edit commands output, but are ignored in the comparing process. Thus, the effect of applying the output of *UNEDIT as commands to the File Editor working on the original generates, as a result, a file with lines that have the same content as the "current" copy and in the same order, but not necessarily with the same line numbers. In the following example, "original" and "current" are the original and current copies as before:

```
           Line Number       Contents

Original:     1               FIRST
              2               SECOND
              3               THIRD
              4               FOURTH
              5               FIFTH
              6               SIXTH
              7               SEVENTH
              8               EIGHTH
              9               NINTH
             10               TENTH

           Line Number       Contents

Current:      1               FIRST
              2               NEW SECOND
              3               THIRD
              4               FOURTH
              5               FIFTH
              6               SIXTH
              8               EIGHTH
              8.25            EIGHTH AND A QUARTER
              8.5             EIGHTH AND A HALF
              9               NINTH
             10               TENTH
```

Issuing the command

```
RUN *UNEDIT 0=original 1=current SPUNCH=changes
```

yields as the contents of "changes":

```
DELETE     2
INSERT     1
NEW SECOND
$ENDFILE
DELETE 7
INSERT 8
EIGHTH AND A QUARTER
EIGHTH AND A HALF
$ENDFILE
```

so that

```
SET ENDFILE=ON
EDIT original
$CONTINUE WITH changes
```

regenerates the current copy.

*UNEDIT produces on PRINT a printed record of the differences found between the two files if PAR=LIST is specified on the RUN command. This is often more useful than the actual edit commands.

The program *APC ("all-purpose compare") may be used to test for the equality of records between files. Using the same "original" and "current" files as in the UNEDIT example,

```
RUN *APC 0=original 1=current PAR=SYNCH
```

would generate the following output:

```
 Unit 0: ORIG lnr 1 len 5 col 1     Unit 1: CURR lnr 1 len 5 col 1
*************************   1 equal line  ****************************
    2     SECOND                       2     NEW SECOND
*************************   4 equal lines ****************************
    7     SEVENTH                   After line 6
*************************   1 equal line  ****************************
 After line 8                          8.25  EIGHTH AND A QUARTER
                                       8.5   EIGHTH AND A HALF

*************************   2 equal lines ****************************
Files have 4 mismatches
4 errors
```

The program *COMPARE is a general file comparison program much like *APC. *COMPARE uses a different and less expensive algorithm; however, the size of the files compared is limited to 65,534 lines. Using the same “original” and “current” files as in the above examples, the command

```
RUN *COMPARE 0=original 1=current
```

would generate the following output:

```
 Unit 0: original              Unit 1: current
 __________________________________________________________________

 =     1     FIRST                  1     FIRST                  =
       2     SECOND                 2     NEW SECOND
 =     3     THIRD                  3     THIRD                  =
 =======================   2 equal lines ==========================
 =     6     SIXTH                  6     SIXTH                  =
       7     SEVENTH
 =     8     EIGHTH                 8     EIGHTH                 =
                                    8.25  EIGHTH AND A QUARTER
                                    8.5   EIGHTH AND A HALF
 =     9     NINTH                  9     NINTH                  =
 =    10     TENTH                  10    TENTH                  =

 Old file "original" has 10 lines,
 and current file "current" has 11 lines.
 Both files have 4 mismatches.
 Total virtual storage used = 3 pages.
 CPU time = 0.017397 seconds.
```

## EXPANDING FILES

When information is placed into a file, the size of the file is increased to accommodate the new data if at all possible. The process of expanding a file requires

(1) verification that the user has not exceeded the file space allotment, and

(2) allocation of space on a disk storage unit for the new extent of the file.

Since these are relatively expensive and time consuming operations, allowing a small file to expand line by line into a large file increases the total cost and real time to record the data in the file.

The user may alleviate this problem by using the CONTROL command to explicitly assign an expansion factor characteristic *permanently* to the file. This command is given in the form

```
CONTROL filename EXPFAC={nP | n%}
```

The EXPFAC parameter has the following effect. If EXPFAC=nP is specified, the file will be expanded by "n" disk pages whenever the file requires more space. If EXPFAC=n% is specified, the file will be expanded by "n" percent of the current size whenever it requires more space. In either case, if the expansion amount is less than required to fit the new line into the file, an adequate amount will be allocated. Conversely, if the expansion amount would exceed either the user's file space allotment or the remaining available space on a disk storage unit, the request will be scaled down appropriately.

Using explicit expansion factors, a user can create a file of a small initial size and can cause it to expand by large amounts. This alleviates the cost and time penalty described above without forcing one to guess the ultimate size of a file and having to bear the effects of a poor choice. For example, suppose one is intending to write 20,000 20-byte lines into a file of an initial size of one page. If the file expanded 1 page at a time, approximately 20,000/(4096/20)=97 expansions would be required. Expanding 5 pages at a time would require only 1/5 of the number of expansions, or 19. If expanded by 100 percent of its current size, then only log(base 2)(20,000/(4096/20))=7 expansions would be necessary. The latter expansion characteristics are clearly preferable.

A default expansion factor can also be requested to cancel a previous one associated with a file. By requesting

CONTROL filename EXPFAC=<u>DEF</u>AULT

(where underlining indicates the minimum acceptable abbreviation), the current default expansion factor, 10 percent, is associated with the file. By default, all newly created files have the default expansion factor.

The FILESTATUS command may be used to display the expansion factor of a file by requesting the EXPFAC item. Executing programs may call the system subroutines CHGXF and GFINFO to change and retrieve the expansion factor, respectively; see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

## CONTROL OPTIONS FOR FILES

The CONTROL command and the CONTROL subroutine may be used with files. For the CONTROL command, the options are specified in the form

```
CONTROL filename option
```

and for the CONTROL subroutine, the options are supplied as the first argument to the subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003). The following table gives the control options available for files.

| *Option* | *Description* |
|---|---|
| EMPTY | Empty the file (same effect as the EMPTY command or EMPTY subroutine). |

| | |
|---|---|
| TRUNCATE | Truncate the file (same effect as the TRUNCATE command or TRUNC subroutine). |
| RENUMBER pars | Renumber the file (same effect as the RENUMBER command or RENUMB subroutine). "pars" is the line number parameters as specified in the RENUMBER command. |
| PKEY=key | Set the program key of the file. See the section "Program Keys" for further details. |
| SIZE={n \| nP} | Change the size of the file to "n" lines or "nP" pages. The size must be greater than or equal to the truncated size of the file and less than or equal to the maximum size of the file. |
| MAXSIZE={n \| nP} | Change the maximum size of the file to "n" lines or "nP" pages. The maximum size must not be greater than 32767 pages or less than the current size of the file. |
| SIZEINC=[±]{n \| nP} | Increment (or decrement) the size of the file by "n" lines or "nP" pages. |
| MAXSIZEINC=[±]{n \| nP} | Increment (or decrement) the maximum size of the file by "n" lines or "nP" pages. |
| {BUFFERS \| MAXBUFS}=n | This dynamically alters the maximum number of 4096-byte buffers used by the file system when reading or writing a file. For line files, "n" must be in the range of 3 to 100 inclusive; for sequential files, "n" must be in the range of 1 to 100 inclusive. This option is only effective for files that are currently open. The control information is not saved; if the file is closed and reopened, the default value will be used. In general, the file system dynamically allocates as many buffers for use in reading and writing a file as there are pages in actual use by the file (i.e., the truncated size) up to the maximum number of buffers specified. Generally, as more buffers are allowed, less physical disk I/O is required, but greater virtual memory is required. Large line files will benefit more than large sequential files from an increase in the maximum number of buffers allowed. The default for both line files and sequential files is 8. |
| EXPFAC={nP \| n%} | Explicitly assign the expansion factor permanently to a file. If "nP" is specified, the file will be expanded by "n" disk pages whenever the file requires more space. If "n%" is specified, the file will be expanded by "n" percent of the file's instantaneous size whenever it requires more space. |
| EXPFAC=DEFAULT | Restore the expansion factor to the default of 10 percent. |
| {SAVE \| NOSAVE} | If the SAVE option is specified, the file "FDname" will be saved periodically on the system file-save tapes. These files may later be restored by running the program *RESTORE. If the |

| | |
|---|---|
| | NOSAVE option is specified, the file will not be saved on the file-save tapes. In this case, there is no system backup in the case of accidental changing, emptying, or destroying. The default is SAVE. |
| TOUCH | The last data change time for the file is updated to be the current time. |

## SHARED FILES

MTS provides extensive file-sharing capabilities. When a file is created, the owner (i.e., the "userID" under which the file was created) has unlimited access to it and all other users have no access to it (unless the NEWFILEACCESS global SET option has been used). However, the owner may allow various levels of file access to other userIDs. Private files that may be accessed by userIDs different from the owner are called shared files. Sharing is implemented via the PERMIT command and the NEWFILEACCESS global SET option. In order to permit access to a file, access information must be given which specifies the type of access the owner wishes to allow and which userIDs are allowed to have this access. After a file is permitted, the userIDs that have access to it may refer to it by prefixing the file name with the userID and the colon (:) shared-file separator character.

### Access Types

There are six basic categories of access to a file. Each can be denoted in several *equivalent* ways.

| *Category* | *Name* |
|---|---|
| Read | READ |
| Write-expand | WRITEXP, WE, APPEND |
| Write-change and Empty | WRITECHG, WC, EMPTY |
| Truncate and Renumber | TRUNCATE |
| Destroy and Rename | DESTROY, RENAME |
| Permit | PERMIT |

Each of these six basic permit categories is independent of the others. Permitting a file using any one of the names for a category is the same as using any of the other names for the same category. All types of access listed in the category are allowed.

There are two different forms of write access:

*Write-expand* means that a user with this access can *add* new lines to the file, but cannot change the lines that are already there. For a sequential file, new lines always are added at the end; hence, the notation APPEND. For a line file, however, new lines may be inserted anywhere in the file.

*Write-change* means that the user with this access can *change* or *delete* lines that are already there, but cannot add new ones. He also can *empty* the file.

To give a user general write access to a file, the owner must specify both WE and WC (this combination can be specified as WRITE or W).

Permit access means that the users so authorized can change the permit status of the file. There is one special case with regard to permit access: the owner of a file always has permit access for that file. Otherwise, the owner could get into a situation where he had a file that he could not do anything with (except pay for it).

In general, it will never be necessary to explicitly give a file truncate and renumber access. Write-expand access will allow a file to be truncated and read-write access (see below) will allow a file to be renumbered.

Several commonly used combinations of categories have been given names of their own; they are:

| *Combination* | *Name* |
|---|---|
| Write-change and Write-expand | WRITE |
| Read and Write-change | RWCHG |
| Read and Write-expand | RWEXP |
| Read, Write-change, and Write-expand | RW |
| Everything except Permit | FULL |
| Everything | UNLIM |
| Default access | DEFAULT |
| No Access | NONE |
| Read, Write-change, and Write-expand from the File Editor | EDIT |
| Read from the RUN command | RUN |

Other combinations must be described in terms of either the basic categories or the combinations given above. The combinations are specified by means of a parenthesized list. Thus, to give a user read and destroy access to a file, the owner would specify (R,D).

## Accessors

Access to a file may be granted to one or more individual userIDs or to groups of userIDs by specifying one of the following:

xxxx

ID=xxxx

where “xxxx” must be from one to four characters. If less than four characters are specified followed by a question mark (?), the file is permitted to the group of userIDs that begin with those characters (not including the trailing “?”); otherwise, the file is permitted to the single userID specified.

A file may also be permitted to all userIDs belonging to a specific projectID “yyyy” by specifying

PROJECT=yyyy

where “yyyy” must be from one to four characters. If less than four characters are specified followed by a question mark (?), a group of projects, all beginning with the characters specified, are given the associated access to the file.

In addition, several specific categories may be used:

OTHERS gives the associated access to all userIDs not specifically permitted via the projectID or userID. This does not include the owner. When a file is created, the access for others is initially NONE. If the owner wishes to change the access to a file, using PERMIT, and fails to specify an explicit accessor, by default, the specified access is given to OTHERS.

ALL gives all userIDs (other than the owner) the associated access to the file, *including* those specifically permitted. To accomplish this, ALL destroys all previously specified permit access to specific userIDs and projectIDs, and also changes the access for OTHERS to that specified. Thus, ALL enables the user to eliminate the access granted to specified userIDs and projectIDs; it does not affect the owner’s access.

ME gives the specified access to the userID issuing the permit command.

OWNER gives the specified access to the userID of the owner of the file (usually the same as ME).

Often it is desirable not only to allow access to ones files by specific userIDs or projectIDs, but also to allow access to data files, for example, by specific programs (object files). To accomplish this, files may have a *program key* attribute associated with them; this association is set by the CONTROL command. Data files may be permitted access by object files (with the appropriate program key attribute) by specifying in the PERMIT command

PKEY=key

where “key” is a program key. See the section “Program Keys” for further details on the use of program keys in MTS.

## The PERMIT Command

The basic form of the PERMIT command is

PERMIT filename [access [accessor]]

where “filename” is the name of the file to be permitted, and “access” and “accessor” specify the access information to be given for the file. If the “accessor” is omitted, it is assumed to be OTHERS; if both "access" and "accessor" are omitted, they are assumed to be READ and OTHERS, respectively. Thus,

```
PERMIT FYLE READ OTHERS
PERMIT FYLE READ
PERMIT FYLE
```

all have the same effect.

In addition, the user may specify several "access-accessor" pairs. The "access" and "accessor" of a pair must be separated by blanks, and the pairs must be separated by commas. Blanks may occur before the comma, after the comma, or both. The form of the PERMIT command thus becomes:

PERMIT filename [access [accessor]] [,access [accessor]] **...**

For example,

```
PERMIT FYLE RW 2AGA, READ OTHERS
```

In addition, the "filename", "access", and "accessor" may each consist of a parenthesized list of items, rather than just a single item. The items in such a parenthesized list must be separated by commas, blanks, or both. Thus, the following commands are valid:

```
PERMIT FYLE READ (ME,OTHERS)
PERMIT FYLE READ (2BCA,2BCB,2BCC)
PERMIT (FILEA,FILEB) (RW,DES) 2AGA, R P=ABCD
PERMIT DBMSFILE RW PKEY=DBMSPGM
```

These are processed left to right for each file. Hence, each file in the list is processed in the same manner. This does not mean that each file has the same permit status; that depends on the previous status of each file. Merely issuing a PERMIT command for a file does not necessarily delete the previous specific sharing information, unless ALL is specified. Each "access-accessor" combination specified either replaces a previous sharing entry, adds a new entry, or deletes an entry (if DEFAULT is specified), depending on whether or not one was there before.

A special access type DEFAULT may be used to remove a specific access granted to a userID, projectID, or program key and thus, by default, leave only the access associated with OTHERS to the specified userID, projectID, or program key. For example, if the user first issues

```
PERMIT FYLE READ OTHERS, UNLIMITED (WXYZ,WABC)
```

and later issues

```
PERMIT FYLE DEFAULT WXYZ
```

the second command would remove WXYZ's unlimited access and leave it with read access. Note that there is nothing special about the owner in this context. Thus,

```
PERMIT FYLE DEFAULT OWNER
```

means that the owner has the same access as OTHERS (of course, the owner always retains permit access). DEFAULT may be abbreviated as DEF.

One additional option on the PERMIT command is available. The sharing capabilities enable the user to build up very complex sharing lists for a file. In such cases, it is useful for the user to be able to copy sharing information from one file to another, so that it does not have to be reentered. This is possible with the LIKE option. The form is

    PERMIT filename1 LIKE filename2

which copies the specific sharing information, the default access, and the owner access from "filename2" to "filename1" replacing any sharing information already associated with the file. The user must have permit access to *both* files to be able to do this. Additional sharing information may be applied to "filename1" by adding the word EXCEPT and the "access-accessor" pairs, as in the normal form of the PERMIT command:

    PERMIT filename1 LIKE filename2 EXCEPT access [accessor]

The additional sharing information is applied to "filename1" *after* the information from "filename2" has been applied. Moreover, either "filename1" or "filename2", or both, may be a parenthesized list of files. The information given after EXCEPT may enhance or restrict access to "filename1" depending on the "access-accessor" lists. Several "access-accessor" pairs may be given in the same manner specified above.

Files may be permitted to an initial substring of a userID or projectID by specifying from one to three characters followed by a question mark (?). Therefore, it may happen that a single userID and associated projectID may potentially have more than one type of access. In these cases, the actual access is resolved (in the absence of program keys) according to the following rules:

(1) If a file is permitted to a given userID and also to its associated projectID, the userID always takes precedence. Furthermore, if a file has been permitted in such a way that more than one match may occur on a userID, then the PERMIT command that specifies the greatest number of characters that match the userID takes precedence. For example, for a file that has been permitted both

```
PERMIT filename READ 2AG?
```

and

```
PERMIT filename RW 2AGA
```

then the user with userID 2AGA has RW access to that file, but user 2AGB has only READ access.

(2) If a file is not permitted to a specific userID, but is permitted to its associated projectID, then the user implicitly has the access to that file associated with that projectID. ProjectIDs, like userIDs, may be specified by 1 to 4 characters, and thus the same potential for conflicts exists when determining accessibility. As in the case of userIDs, the ambiguity is resolved according to the greatest number of matching characters. Thus, for example, in a file that is permitted both

```
PERMIT filename READ P=2BC?
```

and

```
PERMIT filename NONE P=2BCB
```

all userIDs that have projectIDs 2BCA, 2BCC, 2BCD, etc., have READ access to that file, while userIDs that have projectID 2BCB have no access.

(3) If a file has not been specifically permitted for either userID or projectID, the access that has been specified for OTHERS is used. The default access for OTHERS is NONE.

See the section "Program Keys" for details on using the PERMIT command with program keys.

The NEWFILEACCESS option of the SET command establishes a new default access to be given to all new files created by the CREATE command. This command is given in the form

SET NEWFILEACCESS='access accessor'

where "access" and "accessor" is same access/accessor pair that may be specified with the PERMIT command. For example, the command

```
SET NEWFILEACCESS='READ P=ABCD'
```

will give READ access to project ABCD for all new files created. This access may be removed later, either for all userIDs in the project or for an individual userID. For example, issuing the command

```
PERMIT DATA DEFAULT ABC1
```

removes READ access from the individual userID ABC1 for the file DATA.

The initial default access is UNLIM OWNER, and may be restored by giving the command

SET NEWFILEACCESS=OFF

The FILESTATUS command can be used to determine the access to a file.

## PROGRAM KEYS

For most applications, it is sufficient to permit a file to other userIDs or projectIDs with one of the simple access types described above. For example, if the file has been permitted "READ OTHERS", other userIDs can list or copy the file, run the program that resides in the file, or run a program that reads the file. Alternatively, if a file has been permitted "WRITE" to a specific userID, that userID may copy information into the file, edit the file, empty the file, or run a program that changes the contents of the file.

However, there are situations where this level of control is not sufficient. For example, a file may exist that has information about each person in a group. It may be necessary to prevent others from obtaining confidential information about specific persons in the group, but it may be necessary to allow the extraction of statistical information about the group as a whole. Simple READ access does not give this kind of selectivity. As a second example, the operations of a database management system involve a management program that must be able to selectively read and perhaps write the files that contain the database, but not all users (perhaps none at all) should have direct read or write access to the files, since the integrity of the database might depend on the access implemented by the database system.

This more sophisticated control over file access is accomplished in MTS by the use of *program keys*.

## Overview of Program Keys

Every file, whether a private file or a public file, has two attributes associated with it that are relevant for sharing programs. The first attribute is the program key of the file and the second attribute is the access list describing how and by whom the file may be accessed.

By default, the program key is set to *EXEC when the file is initially created. The access is set to UNLIM for the owner and NONE for others. For example, if user WABC creates the file DBMSPGM, which is to contain a database management program, the default file-access structure is as follows:

```
        Program
 ┌────────────────┐
 │                │
 │ WABC:DBMSPGM   │
 │                │
 └────────────────┘
  |
Pkey:
  *EXEC
Access List:
  WABC - UNLIM
  OTHERS - NONE
```

With this access structure, only user WABC (the owner) has access to the program in the file DBMSPGM, and therefore only user WABC may run the program. All other users are denied access to the program. The setting of the program key is irrelevant in this case.

User WABC may allow other users to run this program by permitting the file READ to OTHERS as follows:

```
PERMIT DBMSPGM READ OTHERS
```

The file-access structure can now be illustrated as:

```
        Program
 ┌────────────────┐
 │                │
 │ WABC:DBMSPGM   │
 │                │
 └────────────────┘
  |
Pkey:
  *EXEC
Access List:
  WABC - UNLIM
  OTHERS - READ
```

Unfortunately, these attributes do not restrict the other users from using the file DBMSPGM in other ways. For example, any other user may examine the file by using the EDIT command or may make a copy of the file by using the COPY command.

This deficiency can be overcome by permitting the file so that it can be accessed only by the RUN command. As with a file, each MTS *command* has a program key associated with it. For example, the program key for the EDIT command is *EDIT, while the program key for the RUN command is *MTS.RUN. The program keys for all the MTS command are listed later in this section. Unlike a file, there is no access list associated with an MTS command; each MTS command is always available to all users.

```
           Command
      ┌────────────────┐
      │                │
      │  EDIT Command  │
      │                │
      └────────────────┘
       |
      Pkey:
        *EDIT



      ┌────────────────┐
      │                │
      │  RUN Command   │
      │                │
      └────────────────┘
       |
      Pkey:
        *MTS.RUN
```

User WABC may permit the file DBMSPGM so that it can be accessed only by the RUN command by issuing the command

```
       PERMIT DBMSPGM READ PKEY=*MTS.RUN
```

The file-access structure is now illustrated as follows:

```
           Command                  Program
      ┌────────────────┐       ┌────────────────┐
      │                │       │                │
      │  RUN Command   │       │ WABC:DBMSPGM   │
      │                │       │                │
      └────────────────┘       └────────────────┘
       |                        |
      Pkey:                    Pkey:
        *MTS.RUN                 *EXEC
                               Access List:
                                 WABC - UNLIM
                                 OTHERS - NONE
                                 PKEY=*MTS.RUN - READ
```

With this access structure, all other users still may access the file, but now only by the RUN command. This type of access transforms the program into a “run-only” program.

Instead of making the “run-only” program available to all users, user WABC may restrict access to a selected group of users such as an individual user, several individual users, or a project. For example, user WABC may permit the file DBMSPGM so that it can be accessed only by user WXYZ and only by the RUN command by issuing the command

```
       PERMIT DBMSPGM READ WXYZ&PKEY=*MTS.RUN
```

Since this is a common way to use program keys, a shorthand abbreviation is available:

```
       PERMIT DBMSPGM RUN WXYZ
```

The file-access structure is now illustrated as follows:

```
        Command                  Program
  ┌───────────────┐       ┌───────────────┐
  │               │       │               │
  │  RUN Command  │       │ WABC:DBMSPGM  │
  │               │       │               │
  └───────────────┘       └───────────────┘
   |                       |
  Pkey:                   Pkey:
    *MTS.RUN                *EXEC
                          Access List:
                            WABC - UNLIM
                            WXYZ &
                              PKEY=*MTS.RUN - READ
                            OTHERS - NONE
```

With this access structure, only user WXYZ can run the program in the file DBMSPGM. All other users are denied access.

Another level of sophistication may be added to this example by assuming that the program in DBMSPGM is to read and write a data file named DATA, also belonging to user WABC. If user WABC merely grants RW access to the file DATA by issuing the command

```
PERMIT DATA RW OTHERS
```

there is still the problem that other users may access the data file by means other than running the database management program, which may be undesirable if the data is confidential.

This deficiency may be overcome by having user WABC permit the data file so that it can be accessed only by the program in the file DBMSPGM. This is done in two steps. First, the program key of DBMSPGM is changed by using the CONTROL command to a nondefault value, such as DBKEY:

```
CONTROL DBMSPGM PKEY=DBKEY
```

Internally, the program key DBKEY is made unique by prefixing it with the owner's userID (so that it cannot be confused with the use of the same program key by another user). Second, the file DATA is permitted so that it may be accessed only by programs executing with program key DBKEY by giving the command

```
PERMIT DATA RW PKEY=DBKEY
```

This actually permits the data file such that any user who is allowed to run the program (in this case, only the owner WABC and user WXYZ) will have RW access to the file DATA. The file-access structure can now be illustrated as follows:

```
        Command                 Program                 Data File
   ┌────────────────┐      ┌────────────────┐      ┌────────────────┐
   │                │      │                │      │                │
   │  RUN Command   │      │ WABC:DBMSPGM   │      │   WABC:DATA    │
   │                │      │                │      │                │
   └────────────────┘      └────────────────┘      └────────────────┘
    |                        |                       |
   Pkey:                   Pkey:                   PKey:
     *MTS.RUN                WABC:DBKEY              *EXEC
                           Access List:            Access List:
                             WABC - UNLIM             WABC - UNLIM
                             WXYZ &                   OTHERS - NONE
                               PKEY=*MTS.RUN - READ   PKEY=WABC:DBKEY - RW
                             OTHERS - NONE
```

With this access structure, user WXYZ is now able to run the program DBMSPGM which in turn is able to read and write into the data file DATA. However, user WXYZ is unable to examine or make copies of either the program or the data file.

User WABC may extend the access of the program and data file to other users by issuing additional PERMIT commands. For example

```
PERMIT DBMSPGM RUN WRST
```

will allow user WRST to also run the database program (with RW access to the database).

A variation of this scheme can be shown by assuming that user WABC desires to allow several different users to run the database program, but desires to restrict each of the users to a different type of access to the data file DATA. For example, user WXYZ is to have RW access to the data file and user WRST is to have only READ access to the data file. In order to establish this type of scheme, user WABC would issue the following commands:

```
PERMIT DBMSPGM RUN (WXYZ,WRST)
CONTROL DBMSPGM PKEY=DBKEY
PERMIT DATA RW WXYZ&PKEY=DBKEY
PERMIT DATA READ WRST&PKEY=DBKEY
```

The last two PERMIT command are necessary to restrict the type of access that each of the users running the database program is to have to the data file.

For this case, the access structure is now as follows:

```
       Command                 Program                 Data File
  ┌────────────────┐      ┌────────────────┐      ┌────────────────┐
  │                │      │                │      │                │
  │  RUN Command   │      │ WABC:DBMSPGM   │      │   WABC:DATA    │
  │                │      │                │      │                │
  └────────────────┘      └────────────────┘      └────────────────┘
   |                       |                       |
  Pkey:                   Pkey:                   PKey:
    *MTS.RUN                WABC:DBKEY              *EXEC
                          Access List:            Access List:
                            WABC - UNLIM             WABC - UNLIM
                            WXYZ &                   WXYZ &
                              PKEY=*MTS.RUN - READ     PKEY=WABC:DBKEY - R
                            WRST &                   WRST &
                              PKEY=*MTS.RUN - READ     PKEY=WABC:DBKEY - READ
                            OTHERS - NONE            OTHERS - NONE
```

The final example illustrates how program keys can be used to safeguard a user's own files from accidental change or destruction. In this case, user WABC desires to restrict the source file DBMSPGM.S for the database program to be changeable only by the EDIT command. This can be done by issuing the command

```
PERMIT DBMSPGM.S READ ME, RW ME&PKEY=*EDIT
```

The access structure is illustrated as follows:

```
       Command                Data File
  ┌────────────────┐      ┌────────────────┐
  │                │      │                │
  │  EDIT Command  │      │ WABC:DBMSPGM.S │
  │                │      │                │
  └────────────────┘      └────────────────┘
   |                       |
  Pkey:                   Pkey:
    *EDIT                   *EXEC
                          Access List:
                            WABC &
                              PKEY=*EDIT - RW
                            WABC - READ
```

This prevents user WABC from accidentally destroying the file with a DESTROY command or overwriting the file with a COPY command; only the EDIT command is now allowed to be used to make changes to the file. Other MTS commands that read the file still can access the file; for example, the RUN command can be used to compile the program.

The user may ascertain the current access list and program key for a file by issuing the FILESTATUS command, for example

```
FILESTATUS DATA ACCESS PKEY
```

The above examples illustrate only the most simple uses of program keys for sharing programs. In order to devise more complex examples, the user will have to understand the concept of the current program key, the rules for file-access evaluation, and the methods by which programs are protected when running with program keys. These are discussed in detail in the following sections.

## File Program Keys

Every file in the MTS file system has a *file program key* associated with it. When a file is created, it is assigned the file program key *EXEC; this value is stored in the system catalog along with other information such as the file type and the maximum size of the file.

The file program key allows MTS to distinguish between different programs. For example, it may be useful to permit a file so that it can be accessed from one program, but not accessed from another. If this mechanism is to work, the program must reside in a single file (without $CONTINUE WITH lines). In this case, the file program key for the file becomes the current program key when the file is run.

The owner of the file, or any userID that has permit access to it, may change the file program key using the CONTROL command. For example, if the user whose userID is WABC has a file named PROGRAM and enters the command

```
CONTROL PROGRAM PKEY=DBMSPGM
```

the file program key is changed to DBMSPGM. A user-provided file program key must be from one to eight characters long and may contain no blanks. The same characters that can be used for file names may be used for program key names. Internally, the program key is prefixed by a userID.

When a file program key is assigned via the CONTROL command, the user ordinarily specifies only the external portion of the program key and the system assigns the userID prefix. If the user provides a userID prefix for the program key, the system requires that it match the userID of the user who entered the CONTROL command. Thus, if WABC is signed on and enters the command

```
CONTROL PROGRAM PKEY=WABC:DBMSPGM
```

the command is accepted. If a userID prefix is specified on the CONTROL command but is different from the userID currently signed on, an error comment is produced. Since the internal form of a program key includes a userID as a prefix, different users may assign the same *external* program key to one (or more) of their files but such program keys are internally distinct. Program keys need not be unique for each file under a userID; in fact, it may be desirable for a user to assign the same program key to several different files.

Unless a user explicitly assigns a program key to the file, it retains the default program key *EXEC that is assigned when the file is created.

If a program already has a file program key other than *EXEC, the user may permit files to that program key (see *MTS Volume 2: Public File Descriptions*, Reference R1002, for a listing of the file program keys for public files).

## File Access Via Program Keys

The PERMIT command may be used to allow (and restrict) access to data files by other files (programs) with certain program keys. The general form of the PERMIT command is

PERMIT filename access accessor

where “filename” is the name of the file whose access control information is being altered, “access” is the type of access (read, write, etc.), and “accessor” specifies the userID, projectID, program key, or combination thereof that is to be permitted “access” to “filename”.

Thus, data files may be permitted to specific programs via their associated file program keys. For example, if the user WABC has a file named DATA and enters the command

```
PERMIT DATA READ PKEY=DBMSPGM
```

the program with associated program key DBMSPGM (object file PROGRAM in the previous example) would be granted read access to file DATA.

Ordinarily, when the accessor parameter of a PERMIT command is a program key, the userID prefix of the program key need not be specified. When no userID prefix is given, the system assumes that the userID prefix is that of the user who enters the command. However, if a user wishes to permit a data file to be accessed by a program whose program key has a userID prefix that differs from his own userID, this may be done by providing an explicit userID prefix. For example,

```
PERMIT DATA READ PKEY=WXYZ:OBJ
```

is a valid form for the PERMIT command that grants read access to the file DATA by any program that resides in a file having a program key WXYZ:OBJ.

In certain instances, the user may want to control more precisely the access to certain files. For example, it may be desirable to allow certain programs to access certain files only if the programs are invoked by specific userIDs or by userIDs that belong to a specified projectID. To establish this level of control, the “accessor” parameter of the PERMIT command may be expressed as the logical “and” of a userID (or projectID) and a program key. For example, if the user WABC enters the command,

```
PERMIT DATA RW ID=WXYZ&PKEY=DBMSPGM
```

the program with program key DBMSPGM (object file PROGRAM belonging to WABC in this example) would be granted read and write access to the file DATA only if it is invoked by userID WXYZ. The ampersand (&) is used to represent the logical operator “and” in the accessor expression. To allow read and write access by programs with program key DBMSPGM belonging to and invoked by WXYZ, the command

```
PERMIT DATA RW ID=WXYZ&PKEY=WXYZ:DBMSPGM
```

must be given.

To accommodate program keys, the PERMIT command accepts several types of accessor expressions. In the following list of accessor expressions, uppercase letters represent characters that must be entered exactly as shown; the lowercase letters “xxxx” represent a user-provided userID or project number, and “key” represents a user-provided program key. The following are the legal accessor expressions involving program keys:

PKEY=key
xxxx&PKEY=key
ID=xxxx&PKEY=key
P=xxxx&PKEY=key
PROJECT=xxxx&PKEY=key

In these expressions, “key” may be a simple external program key or an external program key prefixed by a particular userID. If a userID prefix is specified, it must be followed immediately by a colon and the external program key. For example:

```
    ID=WXYZ&PKEY=WABC:DBMSPGM
```

is a valid form for the accessor parameter of a PERMIT command.

As stated in the description of the PERMIT command, “xxxx” may be from one to four characters long; “key” may be from one to eight characters long and may be prefixed by an explicit userID. Permitting a file to a group of userIDs, projectIDs, or program keys by specifying only the first characters of the name followed by a question mark (?) is called permitting the file to an initial substring of the userID, projectID, or program key.

## Command-Processor Program Keys

Each of the MTS command processors has been assigned a specific *command-processor program key*. For most of the MTS commands, the command-processor program key is of the form

*MTS.command

truncated, if necessary, to 12 characters. For example,

| *Command* | *Program Keys* |
|---|---|
| COPY | *MTS.COPY |
| DESTROY | *MTS.DESTROY |
| DUPLICATE | *MTS.DUPLICA |
| EMPTY | *MTS.EMPTY |
| LOG | *MTS.LOG |
| RUN | *MTS.RUN |
| UNLOCK | *MTS.UNLK |

The only exceptions are as follows:

| *Command* | *Program Keys* |
|---|---|
| ACCOUNTING | *ACCOUNTING |
| CALC | *CALC |
| DEBUG | *SDS |
| EDIT | *EDIT |
| FILEMENU | *FILEMENU |
| FSMESSAGE | *FSMESSAGE |
| LIST | *LIST |
| LOCKSTATUS | *MTS.LSTATUS |
| MESSAGE | *MESSAGESYST |
| NET | *NET |
| RERUN | *MTS.RUN |
| SDS | *SDS |
| SYSTEMSTATUS | *SYSTEMSTATU |

During the execution of the command, the current program key is changed to the command-processor program key in question.

The PERMIT command may be used to permit a file to be accessed only by a particular command processor. For example, if the file PROGRAM is permitted using

```
        PERMIT PROGRAM READ PKEY=*MTS.RUN
```

then only the RUN processor could read the object file PROGRAM. The file could be read during RUN command processing for the purpose of loading the program but read access would be denied the LIST, COPY, EDIT, and all other command processors. If one assumes that no other PERMIT commands were entered to specify additional access restrictions, the program in the file PROGRAM becomes a "run-only" program available to everyone. This allows the program to be run. However, it is not possible to copy or list the program or to gain any information about the internal details of the program. Similarly, the EDIT command processor in MTS has been assigned the program key *EDIT. If a user enters the command

```
        PERMIT DATA READ ME, RW ME&PKEY=*EDIT
```

access to the file DATA would be limited to read and write access *while* EDITing the file. By first specifying READ access to ME, the default access for the owner of a file (i.e., UNLIMITED access) is changed to read. Thus, the file can only be changed by the MTS File Editor (secure from the owner's accidental emptying or destruction, although the owner would still be able to read the file, e.g., LIST it) until another PERMIT command is used to modify its access restrictions.

The two shorthand access types, RUN and EDIT, may be used with the PERMIT command. Specifying

```
        PERMIT file RUN xxxx
```

is equivalent to specifying

```
        PERMIT file DEFAULT xxxx, READ xxxx&PKEY=*MTS.RUN
```

where "xxxx" is a userID, P=xxxx, OTHERS, ALL, ME, or OWNER. Likewise, specifying

```
        PERMIT file EDIT xxxx
```

is equivalent to specifying

```
        PERMIT file DEFAULT xxxx, RW xxxx&PKEY=*EDIT
```

These access types may be combined with each other and with the present file accesses, e.g.,

```
        PERMIT DATA (WE,RUN,EDIT) P=WABC
```

## The Current Program Key

MTS maintains a piece of information called the *current program key* which is simply a character string stored in system storage.

The value of the current program key changes during a session in response to the commands issued and programs executed by the user. The intent of changing the current program key is to maintain a program key value that corresponds, in some way, to what the job is doing, so that file-access decisions can be made appropriate to the job's current activity.

For most commands, the current program key retains the value of the associated command-processor program key throughout the execution of the command. For example, when the COPY command is executed, the current program key is set to *MTS.COPY. However, for commands that initiate or resume execution of a program (e.g., RUN and RERUN), the current program key will change as the command executes. For such commands, the current program key will be changed at the moment execution of the program begins to a value associated with the program being executed (i.e., the *file program key*).

When a RUN or RERUN command is issued, the current program key is initially set to *MTS.RUN. However, under certain circumstances it may be changed while loading the program to *MTS.ETC.RUN. This change is intended to prevent users from violating the intent of the “run-only” access provided by the program key *MTS.RUN.

The circumstances under which the RUN command-processor program key is changed, and other considerations of run-only programs, are discussed in the section “Protection of Run-Only Programs.”

The loading of a program is performed with the program key for the command used to load it (e.g., *MTS.RUN), which therefore controls which files may be accessed by the loader. The current program key is changed again, as described below, only when execution is about to begin or continue. The program key used during execution determines which files may be accessed by the executing program.

(1) Under certain carefully controlled conditions, the current program key is set, during program execution, to the file program key associated with the file from which the program was loaded. The conditions under which the current program key is set in this way are described below in the section “Protection of Programs with Nondefault Program Keys.” Basically, they amount to the requirement that the program must be run as a complete self-contained program, and must not be “tampered with” after loading.

(2) When a program begins execution under any circumstances that do not meet the requirements for the use of a nondefault program key, the current program key is set to the key *EXEC (the same as the default file program key).

If the object file has a nondefault file program key, a message will be issued indicating that the key *EXEC is being used instead of the file program key. If, under these conditions, the file program key is required to access a certain data file, access will be denied.

(3) The program key for a user program is determined at the time the main program is initially loaded. Calls to the LINK, LOAD, or XCTL subroutines do not change the program key.

## File-Access Evaluation

Since files may be permitted to numerous combinations of userIDs, projectIDs, and program keys, the system must follow a precisely defined procedure to determine whether a particular combination of userID, projectID, and program key is to have access to a file.

For each file, the system maintains an access list that it uses to determine access rights. This list contains the userIDs, projectIDs, and program keys which have been permitted access to the file. Assuming that access has not been granted to initial substrings of userIDs, projectIDs, or program keys, this list is scanned as follows:

(1) First, the list is checked to determine if the userID attempting to access the file appears explicitly in the access list, either alone or in combination with program keys.

If the userID has been specified in conjunction with a program key that is equal to the current program key, the access specified for that combination of userID and program key applies.

If the userID has been specified alone (not in combination with a program key), the specified access applies.

(2) If the userID is not specified in the access list, the list is checked to determine if an entry specifies the projectID of the user attempting the access (either alone or in combination with program keys). If a match occurs, access rights are determined in a similar manner as for userIDs.

(3) If the list contains no entries that match the userID or projectID of the user attempting to access the file, the list is checked for the current program key as set by the program or command processor accessing the file.

(4) Finally, the access specified for OTHERS is used if access rights were not determined by any of the above procedures.

The above is only a simplified version of the algorithm used. When initial substrings are used in specifying access for userIDs, projectIDs, and program keys, the complete algorithm is as follows:

(1) First, the list is checked to determine if the userID attempting to access the file has been permitted access to the file or if it is a member of a group of userIDs which has been permitted access to the file (e.g., all userIDs beginning with three specific characters, etc.). If the userID is a member of more than one such group, the access associated with the group providing the *longest* initial substring match is used.

If a userID from the list does provide an initial substring match with the userID attempting to access the file but access to the file is allowed only in conjunction with an associated program key, then the associated program key (maintained in the list) must also provide an initial substring match with the program key of the program or command processor attempting to access the file.

If two or more groups of userIDs in the access list provide the longest initial substring match with the userID attempting to access the file, then at least one of the userID groups must have been permitted in conjunction with an associated program key, and the program key that is the longest initial substring match determines the access to be used.

(2) If no initial substring match occurs between userIDs in the list and the userID attempting to access the file, the list is checked to determine if an entry contains a projectID (or initial substring of a projectID) that matches the projectID of the userID attempting to access the file. If a match occurs, access rights are determined in a similar manner as for userIDs.

(3) If the list contains no entries that provide an initial substring match for the userID or projectID attempting to access the file, the list is checked for an initial substring of the current program key as set by the program or command processor accessing the file. If more than one initial substring match is found, the program key with the longest match

determines the access to be used.

(4) Finally, the access specified for OTHERS is used if access rights were not determined by any of the above procedures.

## Protection of Programs with Nondefault Program Keys

When a user assigns a program key to an object file and then permits other data files to be accessed in a specified manner by programs having that particular program key, it is assumed that the intent is to provide a more controlled environment in which only certain programmer-defined file operations will be undertaken on the files in question. For this reason, MTS goes to considerable lengths to ensure that the running of a program which has a nondefault program key will not be compromised. If it is determined that an attempt to run a program with a nondefault program key may violate (as specified in the list given below) the intent of the program key, MTS will assign the default program key *EXEC as the program key to be used during the current execution of the program. When the system does this, a comment is printed and execution proceeds in an otherwise normal fashion. If, under these conditions, the nondefault program key is required to access certain data files, access will be denied. Furthermore, the program can be designed to terminate immediately on the first unsuccessful attempt to access the files.

The cases in which an attempt to run a program with a nondefault program key might compromise the intent of the owner, and thus the cases in which MTS will temporarily reassign the current program key during execution, are:

(1) if explicit concatenation is specified in the object program FDname on the RUN or RERUN command;

(2) if line-number ranges or modifiers are specified on the object program FDname on the RUN or RERUN command;

(3) if LOAD, DEBUG, or SDS has been specified instead of RUN or RERUN;

(4) if a loading error occurs which requires additional input;

(5) if a RESTART command is given to restart at a specific location;

(6) if an ALTER command is given after the program is loaded;

(7) if the shared-file separator character has been set to something other than a colon (:).

In addition, the effect of the LIBSRCH and DEBUG options of the SET command will be ignored while loading a program with a nondefault program key.

It should be noted that the above restrictions are required so that MTS will be able to prevent any unintended access by a program (with a nondefault program key) to data files. MTS accomplishes this in general by changing the program key to the default value so that program-key access to the data file will be denied on subsequent I/O operations.

In order to ease the burden of the user who has to debug and maintain programs with nondefault program keys, two courtesies are provided. First, the previously described reassignment (by the system) of program keys to ensure security is done only when running a program with a nondefault program key whose internal userID prefix is different from the userID currently signed on. That is, the owner of a program with a nondefault program key is not subjected to the above-mentioned checks while running under his or her own userID (unless the userID prefix of the program key differs from that userID). Second, a facility is provided whereby an alternate program key may be assigned to override temporarily the program key associated with the program. This is accomplished by means of the

```
        SET EXECPKEY=key
```

command or the EXECPKEY=key keyword parameter on the RUN command. The SET command is effective for all subsequent RUN commands until a

```
        SET EXECPKEY=OFF
```

command is entered. An alternate program key presented as a parameter on the RUN command overrides that provided via the SET command, if any, as well as the program key associated with the program. The temporary overriding is effective only if the userID prefix, if presented, is identical to the userID of the user attempting to override the program key.

## Protection of Run-Only Programs

It is assumed that when a user permits a program to be “run-only” (i.e., accessible in a READ fashion by only the RUN command processor), the intent is that the program as a single, self-contained file should be only RUN and that no information about the contents of the program should be released. Toward this end, MTS notes that a run-only program has been loaded, and disallows certain commands, i.e., ALTER, DISPLAY, DUMP, or MODIFY (locations, registers, etc.), and RESTART (at a specific location). In addition, MTS controls more carefully the loading process of run-only programs. In particular, the program key used by the system during the loading process of the RUN command, normally *MTS.RUN, is changed to a second value, *MTS.ETC.RUN, in the following cases:

(1) If explicit concatenation, line-number ranges, or I/O modifiers are specified on the object program FDname on the RUN or RERUN command, the program key is changed *before* attempting to access the first member of the FDname. In particular, this means that run-only programs cannot be members of an explicit concatenation.

(2) At the time an implicit concatenation is encountered during the loading process, the program key is changed and then an attempt is made to access the new FDname. In particular, this means that run-only programs cannot be “$CONTINUEd WITH”.

Furthermore, if the LIBSRCH or DEBUG option of the SET command has been specified, it is ignored during the loading of a run-only program. And finally, if an error occurs while loading a run-only program which requires additional input, the loading process is prematurely terminated.

As before, it should be noted that the above restrictions are required so that MTS will be able to prevent any unintended access by the user to the run-only program. In general, this is done by either changing the system-assigned program key before (or during) the loading process, and thus denying run-only program-key access during the loading process, or by terminating the loading process if any errors occur. Prohibiting certain commands while a run-only program is loaded is intended to protect the internal contents and structure of a run-only program.

Although the aforementioned changes are intended primarily to ensure that a run-only program is invoked as a complete, self-contained file, one of the potentially useful side effects of changing the program key to a second value during the loading process is that it provides a mechanism for run-only libraries. For example, if the file STAT:LIBRARY is permitted READ to program key *MTS.ETC.RUN, users may still explicitly or implicitly concatenate the statistical library package to their private programs while at the same time the library owner may guarantee that the library is used in a (somewhat less restrictive) "run-only" fashion.

## LOCKING AND UNLOCKING FILES

If a file has been permitted for sharing, certain rules govern the concurrent use of a file. Obviously, one user should not be allowed to destroy a file while, at the same time, another user is trying to read it. Under these simple rules, any number of jobs (users)[1] may simultaneously read a particular file, but only one job may modify (e.g., write, empty, truncate) a file at any one time, and then only if no other jobs are reading the file. Since access is granted to a file as a whole, these rules are necessary as well as sufficient to guarantee the integrity of the shared file.

MTS enforces these rules by implicitly and automatically "locking" a file for reading or modification whenever a request requiring such locking is made by the user. Thus, for example, when a user asks to LIST a file, MTS will lock the file for reading, list the file, and, when finished, unlock it. While the file is locked for reading, others permitted access to it may read the file, but may not modify it. Similarly, when a user wishes to COPY something to a file, MTS will lock the file for modification, copy the information to the file, and, when finished, unlock the file. If MTS attempts to lock a file and finds that, according to the rules, such locking is not possible at that time (for example, if another job has the file locked for modification), MTS automatically waits until such locking can be accomplished (in our example, until the first user finishes modifying the file and it is unlocked). An attention interrupt may be used to cancel the wait; a message is printed whenever the wait condition is interrupted.

It is anticipated that the implicit, automatic locking, unlocking, and waiting done by MTS will be sufficient to satisfy most shared use of files. For those users who want more control over the locking, unlocking, and waiting conditions that MTS provides implicitly, two commands, LOCK and UNLOCK, and two subroutines, LOCK and UNLK, are provided. These commands and subroutines allow the user to control explicitly the locking, unlocking, and waiting mechanisms. This explicit control is necessary, for example, if a user wants to empty a shared file and copy new information into it, all the while ensuring that no one can read the file after it is empty and before the new information has been entered. In this case, the user should explicitly lock the file for modification, empty it, copy the new information, and explicitly unlock the file. Additional information on using shared files is given in Appendix D.

The LOCKSTATUS command may be used to determine the current lock status of a file.

------------------

[1]In this discussion, a job (or user) means a terminal session or a batch run.

# APPENDIX A
# I/O MODIFIERS

This section lists all the I/O modifiers that may be used with FDnames or with calls to I/O subroutines.

The device types discussed below in the exceptions to the default modifier bit specifications are the device types as returned by the GDINFO subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003). Some of the device types discussed are given below; the remainder are given in the section "System Device List" in *MTS Volume 3*.

| | |
|---|---|
| FILE | Line files |
| SEQF | Sequential files |
| RMRD | Resource manager card input |
| RMPT | Resource manager printed output |
| RMPC | Resource manager punched output |
| 9TP | 9-track magnetic tape |
| MNET | UMnet/Michnet Computer Network |
| 3270 | IBM 3278 Display Station terminals |

The values indicated below with each bit specification are the decimal and hexadecimal values that the modifier word for a subroutine call would have if only that modifier option was specified.

<u>First Fullword of Modifier Bits</u>

| | | | | |
|---|---|---|---|---|
| Bit 31 | SEQUENTIAL, S | Value: | 1 (dec) | 00000001 |
| 30 | INDEXED, I | | 2 | 00000002 |

| | |
|---|---|
| Default: | SEQUENTIAL |
| Exceptions: | None |

The SEQUENTIAL modifier specifies that the input or output operation is to be done sequentially. The INDEXED modifier specifies that an indexed operation is to be performed.

In general, the INDEXED modifier is applied only to line files, while the SEQUENTIAL modifier is applied to line files, sequential files, and all types of devices. Note that the SEQUENTIAL modifier and the sequential file are not directly related.

I/O operations involving *line files* may be performed with either SEQUENTIAL or INDEXED specified. I/O operations involving *sequential files* must be done SEQUENTIALly. If the user specifies INDEXED on an I/O operation to a sequential file, an error message is generated unless the global option SEQFCHK is OFF, in which case the operation is performed as if SEQUENTIAL was specified. Attempting a sequential operation with a starting line number other than 1, e.g., COPY FYLE(2), also gives an error comment if SEQFCHK is ON.

I/O operations involving devices, such as card readers, printers, card punches, magnetic tape units, and terminals, are inherently sequential and are normally done SEQUENTIALly. If the SEQUENTIAL modifier is specified, the line number associated with the line is the value of the current line number plus (minus, if the backwards I/O modifier is given) the increment specified on the FDname. If the INDEXED modifier is specified, the line number associated with the line is the line number specified in the calling sequence. For devices, the INDEXED modifier is used primarily in conjunction with the PREFIX modifier. Note that the device treats the I/O operation as if SEQUENTIAL were specified.

For further details about indexed and sequential input/output operations, see the section "Input/Output Operations" in this volume.

| | | | | |
|---|---|---|---|---|
| Bit 29 | EBCD | Value: | 4 (dec) | 00000004 |
| 28 | BIN, BINARY | | 8 | 00000008 |

| | |
|---|---|
| Default: | EBCD |
| Exceptions: | None |

The action of the EBCD/BIN modifier pair depends on the device. For card readers and punches, the EBCD modifier specifies EBCDIC translation of the card image; this means that each card column represents one of the 256 8-bit EBCDIC character codes. The BIN modifier specifies that the card images are in column binary format; this means that each card column represents two 8-bit bytes of information. The top six and bottom six punch positions of each column correspond to the first and second bytes, respectively, with the high-order two bits of each byte taken as zero. Printers and files ignore the presence of this modifier pair.

Other device support routines that recognize this modifier pair are:

(1) The UMnet/Michnet Computer Network routines
(2) The IBM 3278 Display Station routines

For information on the use of this modifier pair in specifications involving the devices listed above, see the respective sections of *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004, and *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019. The list of device support routines recognizing this modifier is subject to change without notice. Users who wish to keep their programs device-independent should not specify this modifier.

| | | | | |
|---|---|---|---|---|
| Bit 27 | LC, LOWERCASE | Value: | 16 (dec) | 00000010 |
| 26 | UC, CASECONV | | 32 | 00000020 |

| | |
|---|---|
| Default: | LC |
| Exceptions: | None |

The LC/UC modifier pair is not device-dependent. If the LC modifier is specified, the characters are transmitted unchanged. If the UC modifier is specified, lowercase letters are changed to uppercase letters. This translation is performed *in the user's virtual memory region*. On input operations, the characters are read into the user's buffer area and then translated. On output operations, the characters are translated in the user's buffer area and then written out. Only the alphabetic characters (a-z) are affected by this modifier. Unlike IBM programming systems, MTS considers the characters ¢, ", and ! as special characters rather than "alphabetic extenders," and thus, the UC modifier does *not* convert ¢, ", and ! into @, #, and $, respectively. Note that the conversion to uppercase may also be performed by the terminal support routines (see *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004).

| | | | | |
|---|---|---|---|---|
| Bit 25 | NOCC | Value: | 64 (dec) | 00000040 |
| 24 | CC, CARCNTRL | | 128 | 00000080 |

| | |
|---|---|
| Default: | CC |
| Exceptions: | Line files (FILE), sequential files (SEQF), 9TP, and RMPC<br>Controlled by device commands for MNET |

The NOCC/CC modifier pair is device-dependent. This modifier pair controls whether logical carriage control on output records is enabled. For printers and terminals, the first character of each record is taken as logical carriage control if it is a valid carriage-control character and if the CC modifier is specified. If the first character is not valid as a carriage-control character, the record is written as if NOCC were specified. For further information on logical carriage control, see Appendix H to this section.

| | | | | |
|---|---|---|---|---|
| Bit 23 | –PFX | Value: | 256 (dec) | 00000100 |
| 22 | PFX, PREFIX | | 512 | 00000200 |

| | |
|---|---|
| Default: | –PFX |
| Exceptions: | None |

The PFX modifier pair controls the prefixing of the current input or output line with the current line number. On terminal input, the current input line number is printed before each input line is requested. The line number used is determined as specified in the description of the SEQUENTIAL and INDEXED modifiers. An example for terminal input is

```
COPY *SOURCE*(6,,2)@PFX A(6,,2)
     6_ first input line
     8_ second input line
          .
          .
end-of-file indicator
```

The current (prefix) line number is not necessarily equivalent to the file line number. In the example above, the prefix line and the file line numbers were explicitly made to correspond by also specifying a line number range on the output FDname (the file A). On input from card readers and files, the PFX modifier has no effect. On terminal output, the current line number is printed before each output line is written. The line number used is determined as specified in the section "Input/Output Operations" in this volume. An example for terminal output is

```
COPY A(1,10) *SINK*(100,,2)@PFX
    100_ first output line
    102_ second output line
          .
          .
```

Note again that the current line number is not equivalent to the file line number. On output to the printer or to a file, the PFX modifier has no effect.

If the INDEXED and PFX modifiers are given together for terminal output, the line numbers referenced by the INDEXED modifier are the same as those produced by the PFX modifier. As an example, consider the following FORTRAN program segment:

```
  INTEGER*2 LEN
  DATA MOD/Z00000202/   Enables INDEXED and PFX
1 CALL READ(REG,LEN,0,LNR,2,&2)
  CALL WRITE(REG,LEN,MOD,LNR,3)
  GO TO 1
2 STOP
```

This program performs a read SEQUENTIAL and a write INDEXED and PFX. The command (assuming compilation of the above into –LOAD)

```
RUN -LOAD 2=A 3=*SINK*
```

is equivalent to

```
COPY A *SINK*@I@PFX
```

which is also similar to

```
LIST A
```

with a slightly different formatting of the line numbers.

| | | | | |
|---|---|---|---|---|
| Bit 21 | –PEEL | Value: | 1024 (dec) | 00000400 |
| 20 | PEEL, GETLINE#, RETURNLINE# | | 2048 | 00000800 |

| | |
|---|---|
| Default: | –PEEL |
| Exceptions: | None |

The PEEL modifier pair has two functions, depending upon whether it is specified on input or on output. On input, if the PEEL (GETLINE#) modifier is specified, a line number is removed from the front of the current input line. The line number is converted to internal form (external value times 1000) and returned in the line number parameter during the read operation (see the subroutine descriptions of INPUT, GUSER, and READ). The complete input line including the line number is read into the user input region, PEEL processing is performed, the line number (if any) is removed, the remainder of the line is shifted left by the number of characters in the line number, and the length to be returned is decremented by the number of characters removed. Thus, the user input region must be large enough to accommodate both the line number and the line itself. The line number must begin in column 1 (leading zeros are permitted). The line-number separator character (which defaults to “,”) may be used to separate the line number from the line. As an example, consider the following FORTRAN program segment:

```
  INTEGER*2 LEN
  DATA MOD/2048/
1 CALL INPUT(REG,LEN,MOD,LNR,&2)   Read with PEEL
  CALL PRINT(REG,LEN,0,LNR)
```

```
      GO TO 1
    2 STOP
```

The program reads an input line, removes the line number, and writes out the line without its line number. Execution of the object module of the sample program is as follows:

```
RUN -OBJ INPUT=*SOURCE* PRINT=ABC
10AAA
12BBB
```

is equivalent to

```
COPY *SOURCE*@GETLINE# ABC
10AAA
12BBB
```

Listing the file ABC produces

```
LIST ABC
     1     AAA
     2     BBB
```

If the PEEL modifier is specified on input in conjunction with the INDEXED modifier on output, the line number of the input line can be used to control the destination of the line during output. For example:

```
      INTEGER*2 LEN
      DATA MOD1/2048/, MOD2/2/
    1 CALL INPUT(REG,LEN,MOD1,LNR,&2)   Read with PEEL
      CALL PRINT(REG,LEN,MOD2,LNR)      Write INDEXED
      GO TO 1
    2 STOP
```

This program reads an input line, removes the line number, and writes out the line with the extracted line number as the line number specification for an indexed write operation. The following sequence (assuming compilation of the above into –LOAD)

```
RUN -LOAD INPUT=*SOURCE* PRINT=ABC
10AAA
12BBB
```

is equivalent to

```
COPY *SOURCE*@GETLINE# ABC@I
10AAA
12BBB
```

Listing the file ABC produces

```
LIST ABC
    10     AAA
    12     BBB
```

On output, if the PEEL (RETURNLINE#) modifier is specified, the line number of the current output line is returned in the line number parameter of the subroutine call during the write operation (see the subroutine descriptions of PRINT (SPRINT), SPUNCH, SERCOM, and WRITE). The line itself is written out and is unaffected by the presence or absence of this modifier. The modifier is used on output to aid the programmer in recording the line number of the current line written out.

| | | | | |
|---|---|---|---|---|
| Bit 19 | –MCC | Value: | 4096 (dec) | 00001000 |
| 18 | MCC, MACHCARCNTRL | | 8192 | 00002000 |

| | |
|---|---|
| Default: | –MCC |
| Exceptions: | None |

The machine carriage-control modifier pair is device-dependent and in general its use is discouraged. The MCC modifier is used for printing output (via printers or terminals) from programs producing output in which the first byte of each line is to be used as a machine carriage-control command for output to an IBM 1403 (or 1443) printer. If the MCC modifier is specified and the first byte of the output line is a valid 1403 machine carriage-control command code, the line is spaced accordingly and printing starts with the next byte as column 1. If the first byte is not a valid 1403 machine carriage-control command code, the entire line is printed using single-spacing. Spacing operations performed by machine carriage control occur *after* the line is printed (as opposed to logical carriage control in which the spacing is performed *before* each line is printed). Most programs do not produce output using machine carriage control. The MCC modifier pair is ignored for files and devices other than printers, terminals connected through the UMnet/Michnet Computer Network, or IBM 3278 Display Station terminals. For further information on machine carriage control, see Appendix H to this section.

| | | | | |
|---|---|---|---|---|
| Bit 17 | –TRIM | Value: | 16384 (dec) | 00004000 |
| 16 | TRIM | | 32768 | 00008000 |

| | |
|---|---|
| Default: | –TRIM |
| Exceptions: | TRIM for 3270, RMPT, and 3066<br>Controlled by TRIM option of SET command for line files and sequential files |

The TRIM modifier pair is used to control the trimming of trailing blanks from input or output lines. If the TRIM modifier is specified, all trailing blanks *except one* are trimmed from the line. If –TRIM is specified, the line is not changed. For an input operation, trimming does *not* physically delete the trailing blanks from the line, but only changes the line length count. Note that the UMnet/Michnet Computer Network terminal routines unconditionally trim blanks from output lines.

| | | | | |
|---|---|---|---|---|
| Bit 15 | –SP | Value: | 65536 (dec) | 00010000 |
| 14 | SP, SPECIAL | | 131072 | 00020000 |

| | |
|---|---|
| Default: | –SP |
| Exceptions: | None |

The SP modifier pair is reserved for device-dependent uses. Its meaning depends upon the particular device type with which it is used. The device support routines recognizing this modifier pair are:

(1) The file routines
(2) The UMnet/Michnet Computer Network routines

(3) The IBM 3278 Display Station routines

The file routines use the SP modifier to mean skip (do not transmit data) on a read operation to a line or sequential file, and to mean replace on a write operation to a sequential file. For further details, see the section "Input/Ouput Operations" in this volume.

For information on the use of this modifier pair in specifications involving the devices listed above, see the corresponding sections of *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004, and *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019. The list of device support routines recognizing this modifier is subject to change without notice. Users who wish to keep their programs device-independent should not specify this modifier.

| | | | | |
|---|---|---|---|---|
| Bit 13 | –IC | Value: | 262144 (dec) | 00040000 |
| 12 | IC | | 524288 | 00080000 |

| | |
|---|---|
| Default: | The setting of the IC global option (initially ON) |
| Exceptions: | None |

The IC modifier pair controls implicit concatenation. If the IC modifier is specified, implicit concatenation occurs via the "$CONTINUE WITH" line. If –IC is specified, implicit concatenation does not occur. For example, LIST PROGRAM@–IC lists the file PROGRAM and prints "$CONTINUE WITH" lines instead of interpreting them as implicit concatenation. The use of the IC modifier in I/O subroutine calls or as applied to FDnames overrides the setting of the implicit concatenation global option (SET IC=ON or SET IC=OFF) for the I/O operations for which it is specified.

| | | | | |
|---|---|---|---|---|
| Bit 11 | FWD, FORWARDS | Value: | 1048576 (dec) | 00100000 |
| 10 | BKWD, BACKWARDS | | 2097152 | 00200000 |

| | |
|---|---|
| Default: | FWD |
| Exceptions: | None |

The forwards-backwards modifier pair control the direction of the next sequential read operation. On a read backwards operation, the information is placed in the designated region in a manner identical to a read forwards operation, i.e., the front of the logical record is placed at the beginning of the region. For further details on using this modifier, see the section "Input/Output Operations" in this volume.

| | | | | |
|---|---|---|---|---|
| Bit 9 | –ENDFILE | Value: | 4194304 (dec) | 00400000 |
| 8 | ENDFILE | | 8388608 | 00800000 |

| | |
|---|---|
| Default: | The setting of the ENDFILE global option (initially OFF) |
| Exceptions: | None |

The ENDFILE modifier pair controls the recognition of the $ENDFILE command delimiter in the input stream. If ENDFILE is specified, the $ENDFILE line is always recognized as a command delimiter. If –ENDFILE is specified, the $ENDFILE line is never recognized as a command delimiter (the line is taken as a data line). If neither is specified, the recognition of the $ENDFILE line is governed by the setting of the ENDFILE global option (initially OFF). See the SET command for further details.

| | | | | |
|---|---|---|---|---|
| Bit 7 | FDUBCONT | Value: | 16777216 (dec) | 01000000 |

Default: –FDUBCONT
Exceptions: None

The FDUBCONT modifier may be used to specify that another fullword of modifier bits follows the current fullword. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname.

Bit 6 Unused (should be set to zero).

| | | | | |
|---|---|---|---|---|
| Bit 5 | NOPROMPT | Value: | 67108864 (dec) | 04000000 |

Default: –NOPROMPT
Exceptions: None

The NOPROMPT modifier may be used to allow control to be returned to a program after certain errors occur that would otherwise result in a request for a replacement FDname in conversational mode or program termination in batch mode. If the NOPROMPT modifier is specified (bit 5 in the modifier word is 1) when an I/O subroutine call is made, GR0 will be set to a value (see the section "Special Returns" below) indicating that either the I/O operation terminated because of an error while attempting to open a new logical I/O unit or FDUB, or that the I/O operation was completed with its success or failure indicated by the return code in general register 15. This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

| | | | | |
|---|---|---|---|---|
| Bit 4 | MAXLEN | Value: | 134217728 (dec) | 08000000 |

Default: –MAXLEN
Exceptions: None

If the MAXLEN modifier is specified (bit 4 in the modifier word is 1) when an I/O *input* subroutine call is made, only a maximum specified number of bytes of an input record will be returned by the read operation. The second parameter of the input subroutine points to three halfwords instead of the normal single halfword. The first halfword is set to the length of the record returned by the read operation; the second halfword is preset by the caller to specify the maximum record length that is desired; and the third halfword is set to the actual (physical) length of the record. If the incoming record is longer than the maximum length as specified by the second halfword, the record returned will be truncated to the maximum specified length. If the DSR cannot determine the actual length of the record, the third halfword will be set to −1. If the incoming record is less than or equal to the maximum specified length, the first and third halfwords are not guaranteed to be identical values if the TRIM modifier is in effect. This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

| | | | | | |
|---|---|---|---|---|---|
| Bit 3 | NOEC | Value: | 268435456 (dec) | 10000000 |

Default: –NOEC
Exceptions: None

If the NOEC modifier is specified (bit 3 in the modifier word is 1) when an I/O subroutine call is made, explicit concatenation will be inhibited, i.e., if an end-of-file (return code 4) occurs, a return will be made to the calling program instead of proceeding with the next member of the concatenation (if any). This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

| | | | | |
|---|---|---|---|---|
| Bit 2 | NOATTN | Value: | 536870912 (dec) | 20000000 |

Default: –NOATTN
Exceptions: None

If the NOATTN modifier is specified (bit 2 in the modifier word is 1) when an I/O subroutine call is made, all pending attention and timer interrupts, and all attention and timer interrupts occurring during the call, are left pending. This modifier is useful only when used by systems programs (by systems programmers). It may be used only with an I/O subroutine call; it may not be used with an FDname.

| | | | | |
|---|---|---|---|---|
| Bit 1 | ERRRTN | Value: | 1073741824 (dec) | 40000000 |

Default: –ERRRTN
Exceptions: None

If the ERRRTN modifier is specified (bit 1 in the modifier word is 1) when an I/O call is made, and if an I/O error occurs, the error return code is passed back to the calling program instead of printing an error comment. The error return code is returned in general register 15 or, if the NOPROMPT modifier is also specified, in register 0 for some error conditions. The error comment may be retrieved by calling the subroutine GDINFO. This modifier may be used only with an I/O subroutine call; it may not be used with an FDname.

This modifier will cause any calls to the subroutines SETIOERR or SIOERR to be ignored.

| | | | | |
|---|---|---|---|---|
| Bit 0 | NOTIFY | Value: | –2147483648 (dec) | 80000000 |

Default: –NOTIFY
Exceptions: None

If the NOTIFY modifier is specified (bit 0 in the modifier word is 1) when an I/O subroutine call is made, GR0 will be set to a value (see the section “Special Returns” below) indicating that the I/O operation did or did not cause a new FDUB to be opened. A new FDUB is opened when

(1) implicit concatenation occurs,
(2) explicit concatenation occurs,
(3) a FDUB or logical I/O unit is used for the first time,
(4) a return is made from implicit concatenation, or

(5) the maximum line length increases.

This modifier may be used only with an I/O subroutine call; it may not be specified with an FDname.

Second Fullword of Modifier Bits

If any of the following bits are used, bit 7 (FDUBCONT) in the first word of I/O modifiers must also be set.

| | | | | |
|---|---|---|---|---|
| Bit 31 | –LOG | Value: | 1 (dec) | 00000001 |
| 30 | LOG | | 2 | 00000002 |

| | |
|---|---|
| Default: | LOG |
| Exceptions: | None |

If the LOG modifier is specified, the read or write operation will be logged in the log file, if logging is enabled by the LOG command. By specifying –LOG, the user may suppress information from being written into the log file.

| | | | | |
|---|---|---|---|---|
| Bit 29 | –MACRO | Value: | 4 (dec) | 00000004 |
| 28 | MACRO | | 8 | 00000008 |

| | |
|---|---|
| Default: | MACRO |
| Exceptions: | None |

If the MACRO modifier is specified and the input is being read from *SOURCE* (or equivalent), the MTS macro processor is called to interpret lines for macro commands or macro invocations. If the –MACRO is specified, the macro processor is not called. SET MACROS=ON (the default) must be specified for this modifier to be effective. The MACRO modifier pair has no effect on the generation of lines by a macro once it is invoked; these lines are always generated whether or not the MACRO or –MACRO modifier is subsequently specified.

| | | | | |
|---|---|---|---|---|
| Bit 27 | –MFR | Value: | 16 (dec) | 00000010 |
| 26 | MFR | | 32 | 00000020 |

| | |
|---|---|
| Default: | –MFR |
| Exceptions: | None |

If the MFR (macro flag required) modifier is specified and the input is being read from *SOURCE* (or equivalent), the “>” macro flag character must be given for lines that are macro *invocations*. If the –MFR modifier is specified, the “>” is not required. The MACRO modifier and SET MACROS=ON (the default) must also be specified for this modifier to be effective. The MFR modifier pair does not affect lines that are macro *commands*; these always require the flag character.

Bits 0-25 are reserved for future expansion and should be set to zero.

## Special Returns

If the NOPROMPT (bit 5 of the first word) or NOTIFY (bit 0 of the first word) modifiers are specified when an I/O subroutine call is made, the bits in general register 0 (GR0) will indicate the result of the subroutine call. If no bits are set (GR0 is zero), the I/O operation was completed and its success or failure is indicated by the return code in general register 15. If GR0 is nonzero, the I/O operation terminated without completion. The bit assignments are:

| | |
|---|---|
| Bit 31 | The NOTIFY modifier was enabled and a new FDUB was opened as the result of this call, or an old FDUB was used for the first time with the @NOTIFY modifier. |
| Bit 30 | The NOPROMPT modifier was enabled and an error occurred while opening a new logical I/O unit or FDUB. |

The values of bits 0-29 are unpredictable and are reserved for future expansion.

# APPENDIX B
# SEQUENTIAL FILES AND NOTE AND POINT

Associated with every sequential file are *at least* three logical pointers which determine where the next read or write operation will start. Every sequential file has one read pointer for each use of the file[1] as well as one write pointer and one last pointer. These logical pointers are automatically updated after every read or write operation by MTS as outlined below. In addition, there are two MTS subroutines, NOTE and POINT, whereby the user can remember the current values of these logical pointers, and, at some later time, alter the values of these pointers. In so doing, a user is able to start reading and/or writing a sequential file from points other than the beginning and/or end of the file.

These three logical pointers are manipulated by MTS as follows.

The read pointer is always initially set to point to the beginning of the file when the file is created or first referenced. The read pointer is updated *after* every read operation to point after the line read if it was read forwards and before the line if it was read backwards. A particular read pointer affected by a rewind operation (i.e., the read pointer associated with the FDUB on which the rewind was given) is reset to point to the beginning of the file. Finally, all read pointers are reset to point to the beginning of the file whenever the file is emptied.

The write pointer is initially set to point to the beginning of the file when the file is created, and is updated *after* every write operation to point to the *next* line to be written. The write pointer is reset to point to the beginning of the file when the file is emptied or rewound. In addition, if after any forward (backward) read operation, the read pointer is greater (less) than the write pointer, the write pointer is updated to coincide with the read pointer. This allows a user to rewind, read backwards, or skip backwards through a sequential file, begin reading forward, stop at some intermediate point and begin writing at that point. This is similar to what would happen if the same operations were performed on a magnetic tape. The difference is that, if after writing a few lines, the user again began to read the file, reading would begin from the intermediate point at which reading had previously stopped and writing had started (i.e., the read pointer is not updated after a write operation). Finally, whenever the file is first referenced, the write pointer is set equal to the last pointer.

The last pointer is initially set to point to the beginning of the file when the file is created, and is updated *after* every write operation to coincide with the *new updated* write pointer. The last pointer is also considered the logical end of file, so that writing a file beginning from some intermediate point implies that any information from that point on is to be discarded. However, if the write operation specifies the @SP modifier (i.e., a replace is requested), the last pointer is *not* set to the write pointer and the rest of the information is still there. In this case, the record being written into the file must have the same length as the one that was there before. If it is not the same length, it will be truncated or padded with blanks as necessary, written into the file, and then an error message (interceptable as

------------------

[1]When a user has more than one logical I/O unit attached to the same file, or has called the subroutine GETFD more than once for the same file, MTS creates a FDUB (File or Device Usage Block) for each of these uses. The ramifications of having more than one FDUB (and thus more than one read pointer) for the same file are discussed later.

usual with the @ERRRTN modifier) will be issued. Finally, the last pointer is reset to point to the beginning of the file whenever the file is emptied.

The NOTE subroutine can be called to obtain the values of these pointers when reading and/or writing a sequential file, and then the POINT subroutine can be called to “reposition” these pointers. (See the subroutine descriptions in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for calling sequences.) A call to NOTE will obtain four fullwords of information to be used later. These four fullwords are respectively, the read, write, and last pointers, as well as the last line number associated with the file corresponding to the FDUB-pointer given. These pointers always correspond to the *next line* about to be read or written. At some later time, the caller is able to indicate which of these pointers are to be altered for the next read or write operation and, using the information returned by NOTE, can call the POINT subroutine appropriately.

The logical pointers are valid from one copy of a sequential file to another as long as both copies are located on the same kind of storage. Thus, the pointers returned by NOTE for a file on the disk can be used to point into an identical copy of the file elsewhere on disk storage.

The user should be aware of the consequences of reading and/or writing multiple logical I/O units attached to the same file. Since there is one read pointer associated with each FDUB-pointer and with each logical I/O unit, the user is able to read alternately from a number of different points in the file, overlapping or not as the application dictates. However, since there is only one write pointer and one last pointer associated with the file, writing to multiple uses attached to the same file amounts to simply writing to the end of the file in the order that the write requests are received.

# APPENDIX C
# INTERNAL FILE STRUCTURE AND THE SIZE OF FILES

This appendix explains how big a file must be to hold a certain amount of data[1] and describes the internal structure of line files and sequential files.

The basic unit of space in a file is the page, a physical record that is 4096 bytes long.

## Line Files

A line file contains two logical components: the line directory and the data section.

The line directory consists of fixed-length entries, one for each line in the file, and ordered by line number. There are also entries for each available "hole" in the data section. The entries contain the line number and indicate where that line is, in terms of the logical page number within the file (an integer between 1 and 32767) and the displacement within the page. The line directory only points to the lines themselves, which are in the data section.

The data section contains the lengths of the lines, followed by the actual lines, unordered and unpacked (i.e., holes that result because of line replacements stay there until a piece that length or less is needed). Very long lines must be broken into pieces; each piece after the first requires 6 bytes to indicate its location and length.

Initially, the line directory and data section are contained in one page. If the file requires more than one page, the line directory and data section occupy separate pages.

Now to get some approximate numbers for sizes of line files.

If N is the number of lines in the file, and
L is the average length of a line,

then the file can be contained in one page if:

```
N*(L+2) < 3000 and N < 132
```

If the file requires more than one page, then it will use DP data pages and LP line directory pages, where:

------------------
[1]The FSIZE subroutine may be used to determine the size of a file needed to hold a given data set. See *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for further details.

$$DP = \left\lceil \frac{N*(L+2+6*L/4096)}{4096} \right\rceil, \text{ and}$$

$$LP = \left\lceil \frac{N+DP+2}{510} \right\rceil$$

The total number of pages required is thus:

$$P = DP + LP$$

It should be noted that the above figures are approximate, and closest to the actual number only when the file is first filled. In particular, the parameter N above really is the number of lines plus the number of holes, and, after the file has undergone much editing, it may be considerably greater than the number of lines. Duplicating a file into another file (and destroying the original and renaming the copy) will condense the file, and is recommended after a file has been extensively edited.

## Sequential Files

The organization of sequential files is quite simple compared to line files. In general, the first 16 bytes of the first page are used as a header in which pertinent information about the sequential file is retained.

Immediately following this header information are the lines of information stored in the sequence in which they were received. Since these lines may be up to 32767 bytes long, and since the pages on the disk are 4096 bytes, it is quite possible that a line will have to be broken up and stored on more than one page. This is quite likely, even if only “short” lines are written into a sequential file, since the lines are packed end-to-end using up all of one page before going on to the next page. Thus it turns out that even a short line may be broken up across a page boundary.

For this reason it is convenient to refer to a *segment* of a line as that part of the line which resides on a page. A line can therefore consist of one segment, two segments, or more, depending on the size of the line and how the line “fell” with respect to page boundaries. Each segment of a line in a sequential file has 6 bytes of overhead associated with it.

This gives the following lower bound for the number of pages:

$$P = \left\lceil \frac{16 + N*(L+6)}{4096} \right\rceil$$

This is a lower bound because it does not consider that when a line is spanned across page boundaries, each of the segments has the 6 bytes of overhead information attached to it, and almost every page boundary has a line spanned across it.

# APPENDIX D
# DETAILS ON USING SHARED FILES CONCURRENTLY

When discussing the use of shared files in MTS, two distinct processes are of concern. First, the process by which the owner of a file grants or denies others access to that file and subsequently the resolution involved in determining the allowable access to others; second, once access has been granted to a user, the process by which concurrent use of the shared file is controlled. The first process involves the use of the PERMIT command as well as the subsequent access resolution and is described elsewhere; only the second process is described here.

A file can be used concurrently among several tasks (jobs)[1] in MTS. Although the system has hardware available to allow separate tasks to address the same virtual storage (i.e., the same copy of a file), this facility is *not* used in the current mechanism for sharing files in MTS. Thus, two tasks using the same file concurrently actually address their own separate copies of the shared file.

Thus, the problem of controlling concurrent use of a shared file *among separate tasks* amounts to the bookkeeping necessary to guarantee that concurrent use will not endanger the integrity of the actual file on disk. Quite simply, this is accomplished, first, by maintaining a system-wide shared-file table of all files currently in use as well as how and by whom each file is being used and, second, by enforcing a set of rules describing exactly what types of concurrent use can be allowed at any time.

Before describing the rules of concurrent use, something should be said about opening and closing files in MTS as opposed to locking and unlocking files in MTS.

A file is opened the first time it is referenced (read, written, etc.). The opening process requires searching the appropriate file catalog, determining allowable access, allocating buffers in virtual memory for the file, and reading parts of the file from the secondary storage device (disk) into the buffers. A file is closed whenever the user or MTS indicates that the use of the file is complete. Closing a file requires that any modified buffers be rewritten back onto secondary storage, and the buffers be deallocated. Opening and closing a file is a rather time-consuming process and needs to be done only once for each use of a file.

On the other hand, a file must be locked (in the appropriate fashion) before any operation (reading, emptying, etc.) can be performed on the file. The locking process requires that MTS interrogate the system-wide shared-file table to determine if (concurrent) use is possible at this particular time. If so, MTS records in the table how the file has been locked and by whom. A file is unlocked whenever MTS indicates that the use of the file is complete or whenever the user indicates that concurrent use of the file need no longer be restricted. The unlocking process amounts to deleting the appropriate information from the system-wide shared-file table.[2] The MTS file-sharing facility was designed so that the opening and closing of files is independent of the locking and unlocking of files (i.e., a file can be opened without being locked or can be locked without being opened). This is desirable, since in many

------------------

[1]A task can be either a batch job, a terminal session, or a server.

[2]All files in MTS are potentially sharable since MTS allows concurrent signons of the same userID; thus all files must be locked and unlocked during the normal course of operation.

shared-file applications it is necessary to lock and unlock files many times during the single use of a file.

In MTS, three classes of locking are defined to maintain the integrity of a shared file. These three classes are: locking a file for reading, locking a file for modification (writing, emptying, truncating, renumbering, etc.), and locking a file for destroying (renaming, permitting). These three locking classes are inclusive in the sense that locking a file for modification also locks a file for reading, and locking a file for destroying also locks the file for modification and reading.

The rules for concurrent use of a file *among separate tasks* can now be stated in terms of the locking restrictions.

(1) Any number of tasks can have a file locked for reading at the same time as long as no other task has the file locked for modification or destroying.

(2) Only one task can have a file locked for modification at any given time, and then only if no other task has the file locked for reading or destroying.

(3) Only one task can have a file locked for destroying at any given time, and then only if no other task has the file open, or locked for reading or modification.

The process by which files are locked and unlocked falls into two categories. Files are *implicitly* (automatically) locked and unlocked by MTS whenever a user requests something of MTS which requires locking. Files can also be *explicitly* locked and unlocked by the user from either the command level or the subroutine level.

Thus, if a user asks MTS to LIST a file, MTS will implicitly and automatically lock the file for reading, list the file, and then unlock the file. Likewise, if the user asks MTS to COPY information *to* a file, MTS will implicitly and automatically lock the file for modification, copy information to the file, and then unlock the file. Similarly, if the user is operating at the subroutine level, on the first call to a subroutine to read (or write) a line from a file, MTS will implicitly and automatically lock the file for reading (or modification) and then leave the file locked in that manner until the use of that file is complete. In general, the locking is associated with the FDUB (File or Device Usage Block) associated with the file, and the implicit and automatic unlocking is done when the FDUB is released.[1] At the command level, a FDUB is obtained for each use of a file by that command and the FDUB is not released until the (normal) termination of that command. At the subroutine level, FDUBs are obtained via subroutines such as GETFD and released via subroutines such as FREEFD.

If, while attempting to lock a file, MTS determines that according to the concurrent-use rules, it cannot lock the file as requested (for example, if another task has the file already locked for modification), MTS will implicitly and automatically attempt to queue the task to wait until the file can be locked. Before doing so, however, MTS will check to ensure that queuing the task to wait for the file will not result in a situation wherein the current task, as well as others, will be deadlocked indefinitely on their respective wait queues. If such is the case, MTS will not allow the task to wait but instead will return an error indication. An example of deadlocking is as follows:

(1) Suppose task A has file X locked for reading, and

------------------

[1]When MTS *implicitly* locks a file through a particular FDUB, it may “raise” the locking class but will never “lower” it. Thus, if a user operating from the subroutine level first writes a line to a file and then reads a line from the same file, MTS will implicitly lock the file for modification before the first write and leave it locked for modification thereafter.

(2) Suppose task B has file Y locked for reading.

(3) Now suppose task A does something that requires file Y to be locked for modification. MTS, realizing that someone else (task B) has file Y locked for reading, will queue task A to wait for the use of file Y to complete.

(4) Finally, suppose task B does something that requires file X to be locked for modification. MTS realizes not only that someone else (task A) has file X locked for reading, but also that if task B is queued to wait for the use of file X to complete, neither task A nor task B will do another thing, since each would be waiting for the other (to unlock a file) before continuing. In this particular example, task B is not queued to wait, but instead an error indication is returned.

If the file is locked by another task, MTS normally will wait until the file is released by the other task. This waiting may be canceled by an attention interrupt, in which case the message

```
    "FDname": wait to lock cancelled.
```

is printed.

It is anticipated that most shared-file use of MTS could be handled by the implicit automatic locking, unlocking, and waiting done by MTS. To be sure, however, there also needs to be some means of *explicitly* locking and unlocking files, as well as some way of specifying a maximum time to wait on a file if such is desired. These facilities are necessary, for example, if a user operating at the command language level wished to empty a file and copy new information into it, all the while ensuring that after the file was empty and before the new information was copied in, no one would be able to read the (empty) file. Likewise, if a user is operating at the subroutine level, the facility would be necessary if one wished to read a line from a file, change it, and write the line back into the file, all the while ensuring that no other task could be attempting to read or modify that same line after it was read and before it was written back. Thus, MTS has available two commands (LOCK and UNLOCK) and two user-callable subroutines (LOCK and UNLK) to allow users to *explicitly* lock and unlock files as well as to (optionally) request that *no waiting* be done, or that the wait is not to last longer than a certain amount of time if the locking cannot be accomplished. In addition, (batch) tasks can request that their tasks be terminated if for any reason their explicit locking requests cannot be satisfied.

Since, *within any one task*, the locking and unlocking of files is controlled by each use of the file (i.e., each FDUB), the question arises as to how a file is actually locked and unlocked within a task when more than one use (and thus more than one implicit or explicit locking request) is associated with the same file. In general, it can be said that a file is always locked at the level of the highest locking request associated with the file, and when a file is unlocked for a particular use, it is left locked at the highest remaining locking request associated with the file. Thus, for example, if the user explicitly LOCKs a file for reading (one use of the file) and then EMPTYs the file (a second use), MTS will implicitly and automatically lock the file for modification (the second use requires a higher locking class), empty the file, and then unlock the file for modification but leave it locked for reading (the highest remaining locking request). Likewise, if a user, operating at the subroutine level, has called GETFD twice for the same file and has implicitly locked the file for modification by writing the file via FDUB1 and then explicitly locked the file for read via FDUB2, the file would remain locked for modification (the highest level locking request). If then the user explicitly unlocked FDUB1, the file would be unlocked for modification but would remain locked for reading (the highest remaining locking request).

When a file is locked explicitly at the command level, it will remain locked until it is explicitly unlocked at the command level (or until the task signs off). Likewise, if a file is locked explicitly (or implicitly) at the subroutine level via a FDUB it will remain locked until:

(1) the file is explicitly unlocked via the same FDUB,

(2) the FDUB is freed, or

(3) the program terminates execution normally (attention interrupts, program interrupts, calling the subroutines ERROR, MTS, or MTSCMD are *not* normal terminations) or is unloaded.

In addition, EDITing a file implicitly locks the file as needed and leaves the file locked until a new file is edited or until the File Editor returns.

Furthermore, when a file is locked *explicitly*, either from command or subroutine level (assuming no other FDUB has a locking request associated with the file within the same task) the file will actually be locked as requested, i.e., the locking class will be either “raised” or “lowered” or left the same in order to conform to the actual request.

Finally, it should be noted that locking a file is in effect locking the name of that file. Thus, one can, for example,

```
LOCK FILE1 RENAME
RENAME FILE1 AS FILE2
CREATE FILE1
```

all the while leaving FILE1 locked.

# APPENDIX E
# UPDATING FILES DEFENSIVELY IN MTS

One of the hazards of using a timesharing system like MTS is the unfortunate fact that when machine problems or power failures cause the system to go down abruptly, terminal users are not able to sign off gracefully and generally are left with no recourse for recovery of work currently in progress. This is particularly frustrating if the terminal user happens to be in the process of updating a file when the system goes down because, due to the structure of files in MTS, it is possible that a random portion of the most recent changes may not in fact be made. In addition, if the terminal user is particularly unfortunate, the partially updated file may possibly be left in an inconsistent, unusable state by the abrupt halting of the terminal session.

Two questions often asked are: Why do changes, apparently made, not actually get accomplished when the system crashes during the updating process? and, what can the terminal user (or even an interactive program) do to minimize the effort lost when the updating process is terminated prematurely due to machine or power failures?

To understand the answer to the first question, some background about the file system in MTS is necessary.

When a file is first referenced, the first few physical blocks (physical units of information, in general containing many lines) of the file are read from disk into buffers (blocks of storage in virtual memory). This is done because transferring information to and from the user via these buffer images of the physical blocks is much faster than actually reading and writing physical blocks of secondary storage at every request. This process of reading the first few physical blocks into buffers when the file is first referenced has historically been called “opening the file.” Thus, for example, when MTS is requested to read a line (logical record) from the file, if that line happens to be already in a buffer, the information is simply and immediately passed to the requester. If the line is not in a buffer, an existing buffer must be emptied and the physical block on secondary storage containing the requested line must then be read from disk into the now-empty virtual memory buffer and then, as before, the line requested is passed to the user.

Similarly, when MTS is requested to take a line from the user and write it into a file, if that line “happens to fit” (both physically and logically) into a buffer in virtual memory, it is put directly into the appropriate place in the buffer. If it does not happen to fit into any buffers, an existing buffer must be “emptied” and the physical block from secondary storage into which the line properly goes is read into the empty buffer and the line fitted into the appropriate place. In either case, once the line is written, the buffer into which it was placed is flagged as changed. This means that the buffer in virtual memory is no longer identical to the corresponding physical block on secondary storage and, more importantly, the buffer image in virtual memory is henceforth the one and only valid representation of that block of the file.

By now one is able to see the first indications of how problems can and do arise when the system goes down during the updating process. The problem, of course, is that while a file is open, its true and valid state is reflected not only in its physical blocks on secondary storage but also by its quite possibly changed buffer images in virtual memory. Suffice it to say further that when the system goes down abruptly, all buffer images in virtual memory are lost.

Thus, the big question appears to be: When are the buffer images in virtual memory written out in updated form on secondary storage? The answer is quite simply (and again for reasons of efficiency) that while the file is open the buffers are written out to secondary storage only when necessary. Whenever a new physical block is needed in virtual memory and a buffer is not available, an existing buffer must be emptied. Before that buffer is emptied and reused, however, a check is made to see if the buffer has been changed, and, if it has, the buffer is written back out on secondary storage in updated form. The randomness of changes made and changes actually accomplished should now be evident, since there is no direct relationship between when changes are made to a buffer and when the buffer is actually written out in updated form on secondary storage.

To decrease the probability that the secondary storage will be inconsistent, certain operations always cause the buffer images to be written. An example of such an operation is the expansion of a file. Also, for the same reason, whenever any buffer image is written to secondary storage, all changed buffer images are written.

When the user has finally finished updating the file, MTS goes through and writes out all virtual memory buffers that have changed in updated form onto secondary storage. Only at this point is the file safely on secondary storage in a consistent and up-to-date state. This process of writing out changed buffers onto secondary storage has historically been called "closing the file."

Turning to the second question, it becomes evident that the one thing a terminal user (or interactive program) can do to minimize losses which may occur when the system goes down during the updating process is to request MTS to close a file more often than it otherwise might. This of course brings up the obvious question. When does MTS normally close a file? Very simply (and specifically to minimize losses), MTS closes all open files before the execution of every command. This means that during an edit session using the EDIT command, the edit file will not be closed until the File Editor returns to MTS command mode and another MTS command is issued. To circumvent this problem, the Editor CLOSE command may be issued periodically to force the closing of the edit file. (Note: The Editor commands CHECKPOINT and RESTORE are of no help in the event of system crashes, since they are meant to be used only if the user wishes to backtrack.)

The system subroutines CLOSEFIL and FREEFD (the counterpart of GETFD) also close the indicated file. CLOSEFIL closes the file when it is called; FREEFD closes the file only when the last FDUB is released. Also, the system subroutine WRITEBUF will write the VM buffers to secondary storage without closing the file. Thus, interactive user programs which write large amounts of information into a file over long periods of time should occasionally call WRITEBUF or CLOSEFIL to ensure that changes made up to that point in time are actually accomplished. In particular, programs which complete the updating process but continue to execute for long periods of time should always call WRITEBUF, CLOSEFIL, or FREEFD as soon as updating is complete, since MTS will not close the file until execution has terminated.

It should be mentioned at this point that these procedures allow a user or program to force MTS to close a file more often than it otherwise might, and in that way do much to minimize the work lost if the system goes down before the updating process is completed. In general, there is not much that can be done to salvage the changes that were made between the time the file was last closed and the time the system went down. If one is lucky, and generally one is, the only loss is that of checking and redoing those changes (not necessarily the most recent) made but not accomplished during the last open period. If one is unlucky, the file (open when the system crashed) will show signs of inconsistency which are not correctable through normal means. These inconsistencies are evidenced by, for example, two lines with the same line number, or lines which refuse to be deleted or altered. Due to their much simpler internal structure, losses when writing sequential files are most often limited to the last few lines not being written on the disk. The public file *VALIDATEFILE, which checks for inconsistencies in files, should be run on any file that was open and being updated when the system went down. Most inconsistencies can be fixed by the user with the aid of *VALIDATEFILE. Some inconsistencies, however, make a file totally unreadable. In cases of this nature, the ITD staff should be consulted; if the damage is beyond human repair, the file will have to be restored from a save-tape to its status of the day before. The *RESTORE program may be run to restore a file from the save-tape (see *MTS Volume 2: Public File Descriptions*, Reference R1002, for details). In cases where many time-consuming updates are being made over long periods of the day and the restoring of a file to its status of the day before cannot be tolerated, the user should probably make changes to a copy of the file and periodically update the original by duplicating the copy.

Some general comments can be made in closing. First, upon reflection, it should be evident that if a file is only being read, no problems can result from a system crash since none of the virtual memory buffers are changed and thus the file on secondary storage is always valid. Second, due to their internal structure, line files are much more susceptible (than sequential files) to inconsistencies caused by system crashes during an updating process. Finally, update defensively; the work you save will be your own.

# APPENDIX F
# THE XEROX 9700 PAGE PRINTER

The Xerox 9700 Electronic Printing System is a high-speed page printer that produces high-quality output on one or two sides of standard 8.5 by 11-inch paper. The Xerox 9700 operates using a Xerographic process. In this respect, it is much like a large office copier, but instead of requiring an original document whose image is optically projected into the reproduction mechanism, the page printer uses a laser to create the image to be printed from data transmitted by MTS. The page printer prints at a rate of up to two images per second, which is more than ten times the speed of our fastest line printer.

## Pages, Images, and Sheets

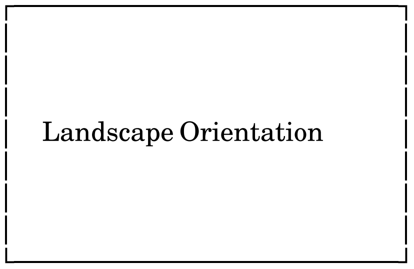
On line-printer output, there is no need to distinguish between pages, images, and sheets, since each sheet of paper contains one image of one page. However, on the page printer this relationship is more flexible.

A page is a logical quantity. A new page is started either when the appropriate carriage control appears (such as a "1" in column one, which causes a new logical page to be started), or when the number of lines per page is exceeded, so that text overflows onto a new page. An image is what gets printed on one side of a sheet of paper, and a sheet (of paper) is a physical quantity. An image may contain more than one page, for example when printing in two-up or four-up configuration.

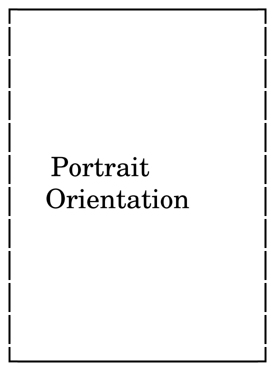
In the normal (default) manner of printing for the page printer, one page is placed on one image, and two images are placed on each sheet (one on each side). Pages also may be printed "two-up" and/or "two-sided". When printing "two-up", two pages are placed on each image, one above the other. When printing "two-sided", two images are printed on each sheet. Thus, when printing "two-up" and "two-sided", four pages can be printed on each physical sheet of paper.

## Page Orientation

The page printer can print text in many formats and fonts, but the major variation is between "landscape" and "portrait" orientation of the paper. These terms indicate the direction in which text lines lie on the paper. In landscape orientation (the way most landscapes are painted), the lines of text lie parallel to the long edge of the paper:



In portrait orientation (the way most portraits are painted), the lines of text are parallel to the short side of the paper:



Delivery of Page-Printer Output

Users can request delivery of page-printer output to the batch stations at NUBS, UNYN, Dearborn, and Flint by specifying the DELIVERY keyword on the SIGNON command for batch jobs, e.g.,

```
SIGNON userid DELIVERY=NUBS
```

or similarly on the CONTROL or SET commands for *PRINT* jobs submitted from a terminal session. DELIVERY=NONE will disable a previous delivery specification for future print jobs. The delivery schedule and stations serviced are given in the public file *DELIVERY.

Additional Features of the Xerox 9700

The Xerox 9700 has many features beyond those described above. It can print characters in different sizes and shapes. Boldface, italics, and script characters are all available. Special and foreign-language characters can be included. The fairly high resolution of the Xerox 9700 (300 dots per inch) makes some graphics support possible, although the 9700 is character addressable rather than dot addressable and so is not a true graphics device. Electronic forms can be used to simulate lined and shaded paper.

These additional features are described in *Using the Xerox 9700 Page Printer*, Reference R1038.

Programs Supporting Xerox 9700 Features

Only a few programs currently support the multiple-font and multiple-page capabilities of the Xerox 9700.

The program *PAGEFONTCONVERT, which is described in Reference R1038, may be used to reformat output intended for a line printer into output more appropriate for the Xerox 9700 page printer.

The program *SIDEBYSIDE, which is also described in Reference R1038, may be used to combine several pages onto one sheet of output.

The *TEX text-processing program supports the multiple-font capability of the page printer.

The program STAT:TEXTEDIT, a text-processor supported by the Statistical Research Laboratory, supports the multiple-font capability of the page printer (see the Stat Lab publication, TEXTEDIT).

## Rates

Printing charges for the page printer are based on the number of images printed and the number of sheets of paper consumed, not on the number of pages or number of lines printed. The specific charges are given in the public file *RATES.

## Operational Considerations

In spite of the high speed of the page printer, turnaround time for small jobs may be slower than when the job is printed on a line printer, because it is not possible to remove output from either of the two output bins without manual intervention (unless the bin fills up). Each output bin holds up to 1500 sheets, so it could take up to 25 minutes for a bin to fill. However, we expect turnaround for jobs to be faster than this, because the bins will be emptied manually on a periodic basis. Also, after a job has been sent to the page printer for printing, but before it has actually been printed and removed from the output bin, it will be shown as “done” on the job status screen and by the command

```
    SYS QUEUE
```

Because there is only one page printer at any site, a single large job could tie up the printer for fairly long periods and no shorter jobs would be able to print. If this proves to be a serious problem, some restrictions may be placed on the maximum size of jobs that can be printed on the page printer during certain hours of the day. Finally, if the page printer is out of service for repair or maintenance, no jobs can be printed and turnaround time will suffer.

# APPENDIX G
# LINE-PRINTER CHARACTER SETS

The standard character set assumed by system software for output directed to line printers is called the T3 character set. The T3 character set consists of all the upper/lowercase letters, the digits 0-9, plus many other special characters. This character set follows the ISO standard and produces as many of the TN characters that exist in the ISO standard. A sample of this character set is given in the file DOC:T3PRTTABLE.

An alternative character set called the TN character set may be specified for the line printer. The TN set has more characters than the T3 set and some characters have different hexadecimal representations. Output from text-processing programs such as FORMAT and TEXT360 should be used with this character set. A sample of this character set is given in the file DOC:TNPRTTABLE.

A second alternative character set called the PN character set is sometimes used with printers at other remote sites. The PN character set consists only of those characters necessary for PL/I plus the double-quote ("), a total of 60 characters plus the blank. This character set does not contain the lowercase letters. ITD no longer has a line printer that uses the PN character set.

For a batch job, selection of the character set is specified by the

```
PRINT={ANY | UC | MC | LC | T3 | TN | PN}
```

option of the SIGNON command. Terminal users may control the character set for output to a printer via *PRINT* by using the same PRINT option of the CONTROL command (or CONTROL subroutine) or the same PRINT option of the SET command.

For the line printer at a Campus Computing Site, the default is T3. ANY, UC (uppercase), LC (lowercase), MC (mixed case), and T3 are synonymous. TN is the only alternative choice. PN is not available. For line printers at remote sites, the default is ANY, i.e., output will be printed on any available printer. The specification of other choices is dependent on the type of line printers available at the remote sites.

The character set specification (via the PRINT options) is independent of print routing (via the ROUTE or PROUTE options). Currently, only the Computing Center (CNTR) has a line printer available. Hence, any job requiring the line printer must be either submitted at the Computing Center or must specify PROUTE=CNTR in the job if submitted at a remote batch station. Otherwise, a warning message will be issued and the PROUTE option will be ignored.

# APPENDIX H
# CARRIAGE CONTROL

The term *carriage control* refers to the user's ability to control the vertical spacing of output. Carriage control is used mainly for output to a terminal or a printer. If the user has specified carriage control, the first character of every record (if output to a printer or a terminal) is interpreted as a *carriage-control character*. The carriage-control character determines the vertical positioning of the output page and is not part of the printed text. The control character is stripped from the output record and printing begins with the second character, rather than replacing the first character with a blank and starting the printing with the blank. If the first character is not one of the legal codes for the particular device being used, a default option of single space is assumed, and the first character is printed as part of the output text. The character codes are independent of the source language used by the programmer.

MTS supports two types of carriage control: logical and machine. Both are used in the manner described above, differing only in the legal carriage-control characters and their effects. Logical carriage control is the more common, and, in general, the user need not be concerned with machine carriage control. MTS supports machine carriage control because a few old programs (notably *ASMG) produce it. The use of machine carriage control is discouraged. In most cases in which carriage control is desired (such as output to printers and terminals), logical carriage control is enabled by default. To select either machine carriage control or no carriage control, the appropriate modifier must be specified. For a description of the carriage-control modifiers, see Appendix A to this section.

## Logical Carriage Control

The following table describes the logical carriage-control characters and their effects.

| | | *Exceptions* | |
|---|---|---|---|
| *Character* | *Effect Before Printing* | *Printer* | *Terminals* |
| blank | Single space | | |
| 0 | Double space | | |
| – | Triple space | | |
| + | Overprint previous line (print without spacing first) | | ss[1] |
| & | Suppress carriage return after printing | undef[2] | |
| 9 | Single space and suppress overflow[3] | | ss |

| | | |
|---|---|---|
| 1 | Skip to top of next page[4] (physical line 4) | skip 3[5] |
| 2 | Skip to next 1/2 page[6] | skip 3 |
| 4 | Skip to next 1/4 page[6] | skip 3 |
| 6 | Skip to next 1/6 page[6] | skip 3 |
| 8 | Skip to logical bottom of page (physical line 63) | skip 3 |
| ; | Skip to top of next physical page (physical line 1) | undef |
| : | Skip to top of next sheet (physical line 1) | undef |
| < | Skip to bottom of physical page (physical line 66) | undef |

[1]ss = single space.
[2]undef = undefined, in which case spacing defaults to single space and the undefined character is printed as text.
[3]Normally, the printer automatically skips the first and last three lines of a page. A logical carriage-control character of "9" suppresses this skip, causing these top and bottom margins to be ignored.
[4]"Top" is physically three lines down from the perforation because of the automatic margin mentioned above.
[5]Skip 3 lines on the UMnet/Michnet Computer Network, and skip 3 lines on the IBM 3278 Display Station (may be modified by TOP device command, see *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004).
[6]The logical page is divided into two halves, four quarters, and six sixths. A logical carriage-control character of 4 will, for example, position the page at the next quarter block even if this may in fact be the top or the middle of a page.

## Machine Carriage Control

Machine carriage control acts at a lower level than logical control. In fact, logical control characters must be converted to machine control characters by system routines. It is very device- and installation-dependent and its use is not recommended. The first byte (i.e., character) of the output record is interpreted as the command in the channel command word (CCW) for an IBM 1403- or 1443-compatible line printer. The following table describes the codes and their functions.

| *Function* | *Byte Value (hex)* |
|---|---|
| Write and no space after printing | 01 |
| Write and space 1 line after printing | 09 |
| Write and space 2 lines after printing | 11 |
| Write and space 3 lines after printing | 19 |
| Write and skip to channel 1 after printing | 89 |
| Write and skip to channel 2 after printing | 91 |
| Write and skip to channel 3 after printing | 99 |
| Write and skip to channel 4 after printing | A1 |
| Write and skip to channel 5 after printing | A9 |
| Write and skip to channel 6 after printing | B1 |
| Write and skip to channel 7 after printing | B9 |
| Write and skip to channel 8 after printing | C1 |
| Write and skip to channel 10 after printing | D1 |
| Write and skip to channel 11 after printing | D9 |

To obtain the corresponding carriage-control operations (space or skip to Channel N) without printing, increase the value of the low-order digit by hexadecimal 2, i.e.,

| | |
|---|---|
| Space two lines | 13 |
| Skip to Channel 5 | AB |

Unlike logical carriage control, machine carriage control spaces *after* printing.

The printer has a 12-channel tape which moves synchronously with the paper. For each line on the page, there is a corresponding line on the tape which may have a hole punched in one of the 12 channels. Thus, a command such as “Skip to Channel 1” has the effect of moving the tape, and, consequently, the paper, until a hole in column 1 of the tape is located. For the tape used by MTS, the effect of this command is to position the paper at the logical top of the next page, i.e., 3 lines down from the physical top. With the present system, Channels 9 and 12 are not available. A complete description of the carriage-control tape for printers at the Computing Center, NUBS, and UNYN (but not necessarily for other remote batch stations) follows.

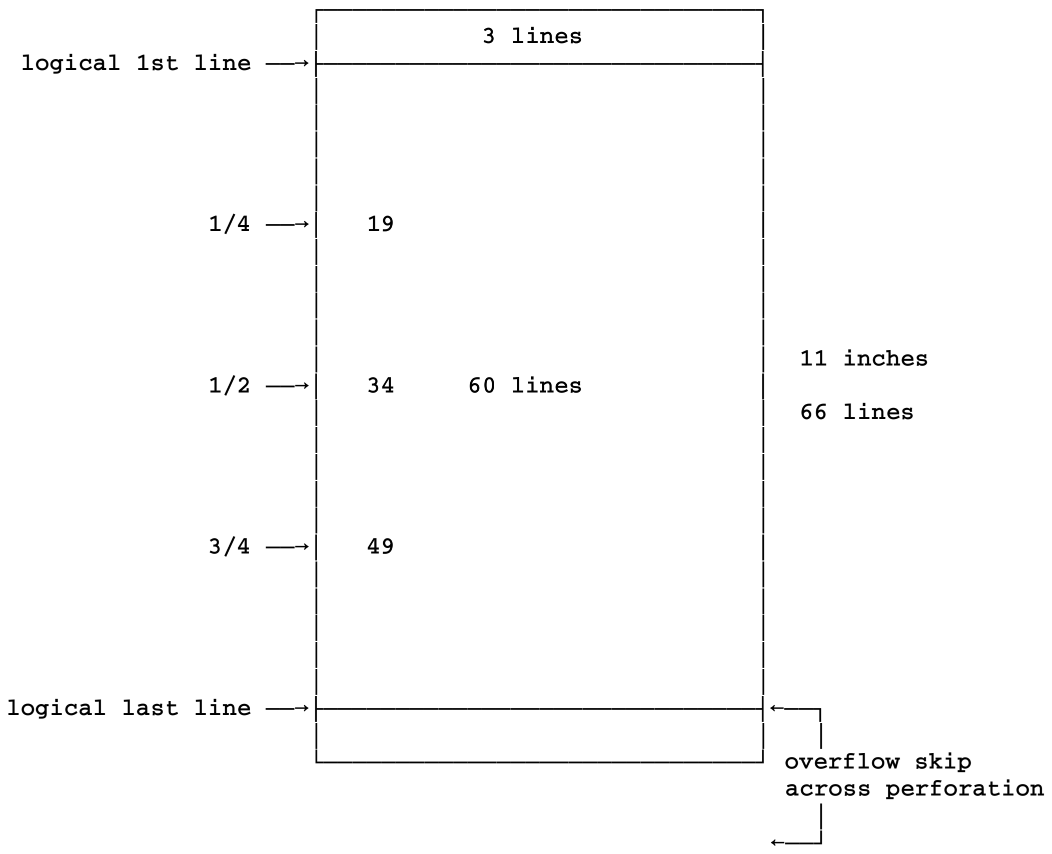
*Carriage-Control Tape*

| *Physical Line of Page* | *Channel Punched* |
|---|---|
| 1 | 11 |
| 4 | 1 |
| 5 | 5 |
| 9 | 10 |
| 14 | 6 |
| 19 | 4 |
| 24 | 7 |
| 34 | 2 |
| 44 | 6 |
| 49 | 4 |
| 54 | 7 |
| 63 | 8 |
| 66 | 3 |

## Devices

Line Printers

The line printer uses 11-inch paper printing 6 lines per inch, and thus 66 lines per page. Printing is usually done in a subset, called a logical page, of this total number of lines. Since the first and last 3 lines of the physical page are skipped, the logical page consists of 60 lines. Thus, in both machine and logical modes, a request to skip to the top of the next page, or equivalently to skip to Channel 1, positions the paper at the fourth physical line.



With logical carriage control, the printer automatically skips over the 3-line margin on both sides of the perforation (unless a “9” carriage control is used, in which case this automatic “overflow skip” is suppressed).

In the horizontal direction, the line printers at the Campus Computing Sites have 132 print positions per line. Other printers may vary, usually from 120 to 144 print positions per line.

Page Printers

The page printer uses only 8.5 by 11-inch paper. The images may be printed either in vertical (portrait) or horizontal (landscape) orientation. The number of lines and columns per page is dependent on the font package selected.

Only logical carriage control may be used with the page printer. The function of the carriage-control codes is the same as for the line printer except that the colon (:) control skips to the top of the next sheet, i.e., it skips to a new physical sheet of paper.

The page printer is described in further detail in Appendix F to this section.

Terminals

The terminals do not have the equivalent of a logical page. The paper is treated as a continuous sheet with no discrete physical pages. Hence, skipping to the top of a page results in merely skipping 3 lines.

# SYSTEM COMMAND LANGUAGE

The sequence of operations of a job processed by MTS is controlled by the commands of the system command language. A command is a request for the system to perform a particular function such as running a program, creating or destroying a file, or setting a global option.

To request that a particular command be executed, enter the command name and, after one or more blanks, the parameters that are required for the command. For example,

```
    SIGNON WXYZ
```

is a command that informs the system that the user whose userID is WXYZ wants to sign on to the system.

When a batch job or terminal session is initiated, the system is in a mode of operation called *MTS command mode*. After the SIGNON command is processed, the system is ready to accept another MTS command. Many commands provide simple services and require only a few parameters (or perhaps no parameters at all) to specify completely the function to be performed. Commands of this type, such as SIGNON, are processed in MTS command mode. Certain commands provide a wider range of services and may have extended facilities, such as a command language subsystem (CLS) associated with them. For example, when a user enters the command

```
    EDIT DATAFILE
```

the system is informed that the user wishes to edit the file named DATAFILE, but the details of which editing functions are to be performed are not specified. To process this command, the system leaves MTS command mode and enters another mode of operation called “edit mode.” While in edit mode, a part of the system called the MTS File Editor controls the behavior of the system. The File Editor provides a command language subsystem that operates only when the system is in edit mode. Edit mode commands allow users to inspect, alter, extend, etc., the contents of the file being edited. In addition, several edit mode commands are available that instruct the system to leave edit mode and return to MTS command mode.

The principal MTS commands that provide extended facilities and cause a transition from MTS command mode to a different mode of operation are ACCOUNTING, CALC, DEBUG, EDIT, FILEMENU, FSMESSAGE, FTP, MESSAGE, NET, and SYSTEMSTATUS. Each mode of operation provides one or more means of returning to MTS command mode. The various modes of operation are logically separate; that is, each mode may be entered and exited at any time without affecting the status of any other mode. With one exception the names of the various modes of operation are identical to the name of the MTS command that makes that mode available (e.g., CALC provides access to “calc” mode); the exception is *execution mode* which may be entered via the RUN command (and several other commands as described below).

The facilities of all modes of operation *except execution mode* are provided by system programs. MTS provides certain supporting services (such as loading a program) for execution mode but for the most part the behavior of the system in execution mode depends on the program being executed. The program to be executed may be one prepared by the user or a system program such as the FORTRAN or Pascal compiler. Many system utility programs are available to perform various services in execution mode; they are described in *MTS Volume 2: Public File Descriptions*, Reference R1002.

Besides the extended facilities provided for each mode of operation there are other characteristics of the modes of operations that are of considerable value. In particular, since the modes are logically separate, it is possible to retain certain information pertaining to a given mode of operation even though the system undergoes a transition to a different mode. For example, “calc” (desk-calculator) mode may be entered via the CALC command, certain variables may be defined and assigned values, and then a return to MTS command mode may be effected so that some other MTS commands can be executed. When a subsequent CALC command is entered, the system reenters calc mode and the variables previously established are still available. Thus, one may consider the use of calc mode to be comprised of several intervals during which calc mode is active with intervening periods during which calc mode is suspended and some other mode is active. The collection of intervals during which calc mode is active comprise a session during which all information pertinent to calc mode is retained.

The retention of information pertaining to a particular mode of operation can be very useful but occasionally the user may wish to “get a fresh start” for some mode of operation. The user controls whether information pertaining to a mode is retained or destroyed when the system leaves that mode of operation; he does so by presenting an appropriate command or signal to the system. For example, if the system entered calc mode from MTS command mode, entering the character string “RETURN” or “MTS” causes a return to MTS command mode with calc mode information retained but entering “STOP” causes a return to MTS command mode with calc mode information destroyed.

The “MCMD” command or an MTS command prefixed by the “$” command flag may be used to transfer from one mode of operation to another. For example, if the system is in edit mode and the user enters the edit command

```
MCMD CALC
```

or simply the MTS command

```
$CALC
```

the system leaves edit mode and enters calc mode. When a RETURN or STOP command is entered, the system leaves calc mode and reenters edit mode.

With the exceptions noted below (in the section on delimiters) the interpretation of an input line depends on the mode of operation.

## MODES OF OPERATION WITHIN MTS

### MTS Command Mode

MTS command mode is the default mode of system operation for all batch jobs and terminal sessions. MTS commands can be subdivided into several groups according to the services they provide. The major groups are:

(1) Global Control

The global-control commands include SIGNON and SIGNOFF to initiate and terminate jobs, and SET to set various system options.

(2) Program Control

Program-control commands include RUN, RESTART, and RERUN to initiate and/or resume program execution, and IF to test program execution results.

(3) Debugging

The debugging commands include DEBUG and SDS to enter the Symbolic Debugging System, DUMP, DISPLAY, and ALTER to inspect and/or modify registers and storage locations.

(4) File and Device Management

File and device management is provided by the CREATE, DESTROY, EMPTY, PERMIT, RENAME, RENUMBER, LOCK, UNLOCK, and CONTROL commands. Information about the status of files may be retrieved using FILESTATUS, FILEMENU, and LOCKSTATUS. The COPY, DUPLICATE, FTP, and LIST commands are useful in the transfer of data from one file or device to another. SOURCE and SINK control the current input and output streams. Magnetic tapes may be mounted by the MOUNT command.

(5) Help Information

Help information is provided by the HELP command.

The commands mentioned above are a subset of the system command language; a complete list of MTS commands, parameters that are allowed (or required) for their use, and a description of their use is given in the section “MTS Command Mode” in this volume.

## Execution Mode

Execution mode is used to execute programs and is entered via the RUN, RERUN, START, RESTART, and several other commands and from debug mode as described below. If the RUN command is given, any currently loaded program is unloaded and the object module specified by the first command parameter is loaded; when loading is complete, control is passed to the loaded program at its entry point and the system enters execution mode. The RERUN command causes the *previous* RUN or RERUN command to be processed as though it had just been reentered. If parameters such as logical I/O assignments are entered on a RERUN command, they override those specified on the previous RUN or RERUN. When a program is already loaded (e.g., via LOAD), the START or RESTART commands may be used to enter execution mode. Normally, the return to the mode from which execution mode was invoked occurs when the program proceeds to completion (or calls one of the system subroutines SYSTEM, MTS, MTSCMD, or ERROR).

The occurrence of certain abnormal conditions also causes a return to MTS command mode or debug mode. For example:

(1) For batch jobs only:

(a) a global time, page, or card limit is exceeded (does not return to debug mode).

(2) For batch jobs and terminal sessions:

  (a) a local time, page, or card limit is exceeded.
  (b) an abnormal condition (e.g., a program interrupt) occurs. The program may call the appropriate subroutine (e.g., PGNTTRP) to intercept the return to MTS command mode.

(3) For terminal sessions only:

  (a) an attention interrupt occurs. The program may call the appropriate subroutine (e.g., ATTNTRP) to intercept the return to MTS command mode.
  (b) the user's account has run out of funds and the user chooses to return to MTS command mode when prompted.

## Accounting Mode

Accounting mode may be used to obtain information about the resource status of a userID. It also may be used by project directors or instructors to manage the resources (money, disk space, etc.) available to their projects or classes. Accounting mode is entered via the ACCOUNTING command. Accounting mode is described in the section "Accounting" in *MTS Volume 5: System Services*, Reference R1005.

## Calc Mode

Calc (or desk-calculator) mode provides sophisticated desk-calculator capabilities. Calc mode is entered via the CALC command. While in calc mode, mathematical expressions may be evaluated, and symbolic variables may be defined and values may be assigned to them. Calc mode is described in the section "MTS Command Mode" in this volume. The facilities of calc mode are available to a user program by calling the CALC subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

## Debug Mode

Debug mode is entered via the DEBUG or SDS command. In debug mode, a part of the system, called the *Symbolic Debugging System* (or SDS), enables the user to monitor the execution of a program.

The debug command language subsystem provides convenient facilities for controlling the execution of the program and for displaying and modifying instructions and/or data at any point during execution of the program. When a DEBUG command is entered, the program to be debugged is loaded for execution and appropriate symbol table information (from SYM records in the object module) is made available to the symbolic debugging system. After loading is completed, the system enters debug mode at which time the user may initiate execution of the program. While the system is in debug mode, the program may be interrupted and various locations may be displayed or modified. After any desired modifications have been made, execution of the program may be resumed via the CONTINUE, GOTO, or STEP debug commands. The Symbolic Debugging System is described in *MTS Volume 13: The Symbolic Debugging System*, Reference R1013.

## Edit Mode

Edit mode is entered via the EDIT command. In edit mode, the user may use the MTS File Editor to display and selectively modify the contents of a file. The File Editor provides special support for display terminals such as the Ontel and the IBM 3270, and various microcomputers such as the IBM PC and the Apple Macintosh to visually display and modify a file. The File Editor is described in *MTS Volume 18: The MTS File Editor*, Reference R1018. The facilities of the File Editor are available to a user program by calling the EDIT subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

### File Transfer Mode

File-transfer mode is entered via the FTP command. In file-transfer mode, the user may transfer files to or from a remote host machine. The FTP command is described in the section "MTS Command Mode" in this volume.

### List Mode

List mode is entered via the LIST command. In list mode, the user may issue various commands to control the formatting of the listing. The LIST command is described in the section "MTS Command Mode" in this volume.

### Message-System Mode

Message-system mode is entered via the MESSAGE or FSMESSAGE command. In message-system mode, the user may send messages to other users either local or remote. The messages may be addressed by either userID or name. The MTS Message System is described in *MTS Volume 23: Messaging and Conferencing in MTS*, Reference R1023.

### Network Mode

Network mode is entered via the NET command. In network mode, a part of the system, called the *network interface*, provides access to computers located at other installations (hosts) of the UMnet/Michnet Computer Network. The NET command is described in the section "The NET Command" in *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004.

### Systemstatus Mode

Systemstatus mode is entered via the SYSTEMSTATUS command. In systemstatus mode, the user may inquire about the status of particular jobs being processed by the system, about the status of particular devices attached to the system, etc. The SYSTEMSTATUS command is described in the section "MTS Command Mode" in this volume.

### View Mode

View mode is entered via the VIEW command. In view mode, the user may look at the results of a print or batch job. The VIEW command is described in the section "MTS Command Mode" in this volume.

## PREFIX CHARACTERS

In order to assist users in interpreting system activity during a terminal session, a prefix character is printed when an input line is requested from a terminal and also at the beginning of each output line directed to a terminal. Since prefix characters are intended to assist terminal users in the interpretation of system activity, they are used only for terminal devices; they are never written to disk files, magnetic tapes or other devices. The prefix character depends on the mode of operation of the system and on the particular activity within that mode. Although certain prefix characters may be used in two or more modes of operation (e.g., the question mark), prefix characters are usually characteristic of only one type of system activity. The prefix characters most commonly used are tabulated below.

| *Mode* | *Prefix Character* |
|---|---|
| MTS Command Mode | # |
| Command Continuation | #– |
| Copying | > |
| Loading | . |
| Prompting | ? |
| Execution Mode | blank |
| Accounting Mode | $? |
| Calc Mode | |
| Input | ? |
| Output | = |
| Debug Mode | + |
| Command Insertion | ? |
| Edit Mode | : |
| Fast Insertion | ? |
| File-Transfer Mode | ftp> |
| List Mode | > |
| Message-System Mode | @ |
| Text insertion | ? |
| Network Mode | ) |
| Systemstatus Mode | – |
| View Mode | * |

The printing of prefix characters is controlled by an option that is normally on. However, prefix characters can be suppressed by entering the command

```
SET PFX=OFF
```

When the prefix option is off, prefix characters are suppressed globally; that is, they are suppressed for all modes of system operation.

The default prefix character for execution mode is the blank but an alternate can be assigned by calling the subroutines CUINFO (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for details) or by using the SET command option EXECPFX. Various system programs use alternate prefix characters in execution mode; for example, *FS assigns the equal sign (=) as its execution mode prefix.

## COMMANDS AND DELIMITERS

An input line having the form of an MTS command is interpreted as such by MTS only if the system is in MTS command mode. Thus, the input line

```
    COPY A B
```

is treated as an MTS copy command if the system is in MTS command mode, but is treated as an ordinary data line if the user's program reads it in execution mode. MTS provides two special constructs, the *end-of-file* delimiter and the *implicit concatenation* delimiter, that have the same general format as MTS commands. These constructs are recognized in a broader context than MTS commands.

The form of the end-of-file delimiter is

```
    $ENDFILE
```

The "$" is always required. This delimiter signals an end-of-file condition. The scope of this delimiter depends on the setting of the global ENDFILE option which can be set by the MTS command SET (e.g., SET ENDFILE=SOURCE), by the CUINFO subroutine (see the GUINFO, CUINFO subroutine description in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003), or by a pair of I/O modifier bits. If the ENDFILE option is set to SOURCE (the default case), a line consisting of "$ENDFILE" is recognized as the end-of-file delimiter only if it is read from *SOURCE* or *MSOURCE*. If the ENDFILE option is set to ALWAYS, a line consisting of "$ENDFILE" is *always* treated as an end-of-file delimiter. If the ENDFILE option is set to NEVER, a line consisting of "$ENDFILE" is *never* treated as a delimiter, but always as a data line.

If the ENDFILE option is set to NEVER for a batch job, the only other method available for generating an end-of-file condition involves the use of the implicit concatenation delimiter (see below). If the ENDFILE option is set to NEVER for a terminal session, there is always an alternate control function available to signal the end-of-file condition. The I/O FDname modifiers ENDFILE and –ENDFILE (see Appendix A of "Files and Devices") may be used to override the setting of the global ENDFILE option.

The form of the implicit concatenation delimiter is

```
    $CONTINUE•WITH FDname [RETURN]
```

where • means exactly one blank. With the exception noted below, this delimiter causes implicit concatenation to occur. For a complete description of implicit concatenation see the section on "Files and Devices" in this volume. Recognition of the implicit concatenation delimiter is controlled by both the global IC option and a pair of modifier bits. The IC option may be set ON or OFF by the SET command or by an appropriate call to the CUINFO subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003). The default is ON, which means that this construct is recognized as a delimiter. The I/O modifier bits are IC and –IC (see "Files and Devices") and when applied to an FDname, they override the setting of the global IC option for those uses of the FDname.

The implicit concatenation delimiter can be used to provide an end-of-file condition by specifying

```
$CONTINUE•WITH *DUMMY*
```

This may be useful in batch jobs when the ENDFILE option has been set to NEVER.

## MTS COMMAND MACRO PROCESSOR

The MTS command extension and macro processor is a special interface between the user and MTS that allows users to "custom" design their own commands. A series of MTS or CLS (command language subsystem) commands may be combined into a single command or into a macro (a program of commands) to perform a special task.

Extended commands can appear in two basic forms: as a simple command statement entered in "open code", usually signalled by a ">" in column one, or as a macro call of a previously defined macro. Any line entered from *SOURCE* is examined for a ">" in column one which indicates that the line is an open-code command that should be processed by the macro processor.

A conditional command is a common form of an open-code command. It is the form

```
>IF expression, command
```

where "expression" is a logical statement that evaluates to either true or false, and "command" is a statement that is acted upon if the expression is true. For example, the conditional command

```
>IF SIGNONID="WABC", SET ROUTE=UNYN
```

could be placed in a shared sigfile to request that output generated by the userID WABC be routed to the printer at the UNYN batch station.

A macro is a small program comprised of several commands. The definition of a macro is given in the form

```
>MACRO name [optional parameters]
command 1
...
command n
>ENDMACRO
```

The macro may be subsequently invoked by merely giving the name. For example, the following macro could be defined to save a file on magnetic tape TAPE99:

```
>MACRO TAPE99 FILE
MOUNT MYTAPE *T* WRITE=YES VOL=TAPE99
CONTROL *T* POSN=*EOT*
CONTROL *T* NAME={FILE}
CONTROL *T* VB(10000,100)
COPY {FILE} *T*
RELEASE *T*
>ENDMACRO
```

To invoke the macro to save the file DATA1 on the tape, the user would simply enter the command

```
    TAPE99 DATA1
```

Macro definitions may be saved in special macro libraries so that they need not be defined every time the user signs on. Complete details on using the MTS command extension and macro processor, including the defining of macro libraries, is given in *MTS Volume 21: MTS Command Extensions and Macros*, Reference R1021.

# MTS COMMAND MODE

MTS command mode is the default mode of system operation for all batch jobs and terminal sessions. The principal function of MTS command mode is the entry and processing of MTS commands.

## PROCESSING MTS COMMAND LINES

The following rules apply to the processing of MTS command lines:

(1) The MTS command verb must be the first word in the command line. Leading blanks are allowed both in batch and conversational mode. However, "$CONTINUE WITH" and "$ENDFILE" lines (which technically are not MTS commands) must begin in column one (the "$" is required). The command lines are not converted to uppercase.

(2) Null lines and blank lines are ignored.

(3) Lines beginning with an asterisk "*" or "$*" are treated as comment lines. They are not echoed.

(4) The dollar sign "$" command flag is optional in both batch and conversational mode. However, "$CONTINUE WITH" and "$ENDFILE" lines must begin with a dollar sign.

(5) If the leading characters of the command line constitute a valid MTS command name or command name abbreviation, the command is processed; if not, the error comment "Invalid MTS command" is produced. For terminal sessions, a new input line is requested if SET ERRORPROMPT=ON (the default); for batch jobs, subsequent input lines are discarded until an input line that begins *in column one with a dollar sign* and a valid command name is encountered.

The CMDSCAN option of the MTS SET command specifies the method used to recognize MTS command names. When the CMDSCAN option is set to UNAMBIGUOUS (the default), the recognition of MTS commands is limited in the following ways:

(1) When an MTS command is abbreviated, enough of the characters composing the command verb must be given to distinguish the command from all other MTS commands, or their abbreviations. If this is not the case, the error comment "Ambiguous MTS command" is produced.

(2) Characters other than the characters composing the full command name may not be given (e.g., CANCELLATION and CANCEL are not equivalent and will produce an error comment).

(3) Spelling correction of unrecognizable commands is attempted under control of the SPELLCOR option. If spelling correction determines that the command can be mistaken for no more than two other MTS commands, verification and/or correction of the command verb will result. Otherwise, the command will be in error.

(4) Certain short and ambiguous, but harmless abbreviations are recognized: C for COPY, D

for DISPLAY, L for LIST, R for RUN, and SIG for SIGNON and SIGNOFF.

When the CMDSCAN option is set to AMBIGUOUS, the command recognition procedure uses the following abbreviations for otherwise ambiguous commands:

| *Command* | *Abbreviation* |
|---|---|
| ALTER | A |
| CALC | CA |
| COPY | C |
| DESTROY | DE |
| DISPLAY | D |
| DUMP | DU |
| EMPTY | E |
| FILESTATUS | F |
| LOAD | LO |
| LIST | L |
| MODIFY | M |
| RENAME | REN |
| RESTART | RE |
| RUN | R |
| SET | S |
| SIGNON/SIGNOFF | SIG |
| SINK | SI |
| UNLOAD | UNL |

In MTS command mode, input lines are read from the file or device assigned or defaulted to the pseudodevice *SOURCE*. At the beginning of every batch job or terminal session, *SOURCE* is defaulted to the same device as *MSOURCE*. *MSOURCE* is always assigned to the keyboard of the user's terminal for terminal sessions and the card reader (or *BATCH* input) for batch jobs. The assignment for *SOURCE* may be changed using the SOURCE command, e.g.,

```
SOURCE CMDFILE
```

Also, at the beginning of every batch job or terminal session, *SINK* is defaulted to the same device as *MSINK*. *MSINK* is always assigned to the printer of the user's terminal for terminal sessions and the printer (or *BATCH* input) for batch jobs. The assignment for *SINK* may be changed using the SINK command, e.g.,

```
SINK OUTFILE
```

Normally, MTS commands from *SOURCE* are echo-printed (or simply "echoed") to *SINK*, if *SINK* differs from *SOURCE*. In addition, command lines are echoed to *MSINK*, if *MSINK* differs from both *SOURCE* and *SINK*. Thus, MTS commands read in via the batch card reader (*SOURCE*) appear in the output listing (*MSINK*) for the job. For a terminal job, *SOURCE*, *SINK*, and *MSINK* are normally identical (unless the SOURCE or SINK commands have been used to reassign *SOURCE* and *SINK*), thus echoing is not usually necessary. Echo-printing of command lines may be disabled by entering the command

```
SET ECHO=OFF
```

and may be reenabled by

```
SET ECHO=ON
```

Any MTS command may be executed from a program using the CMD, CMDNOE, or COMMAND subroutines (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003). This is simply a mechanism in which a transition is made to MTS command mode and the input lines are supplied by the subroutine rather than being read from *SOURCE*. Following execution of the MTS command, a transition back to execution mode is made automatically, unless the MTS command was one which caused modes to be switched.

## CONTINUATION LINES

Occasionally it is necessary to enter a command line that requires more characters than the maximum (e.g., a maximum of 80 characters per card read by a card reader) allowed by the input device assigned to *SOURCE*. In MTS command mode, input lines up to 255 characters can be accumulated according to the following conventions. If the last character of an input line is the *line-continuation character* (by default, a minus sign "–"), the line is assumed to be incomplete. The line-continuation character is deleted and MTS requests another input line (a continuation line) which is appended to the characters previously entered. As many continuation lines as desired may be entered, but the total accumulated length may not exceed 255 characters. In batch mode, the last character position is column 80 on cards read through a card reader or the last character in a line submitted via *BATCH*. For terminal input lines, the line-continuation character must be the last character entered before the line is terminated (e.g., by a carriage return). When reading a continuation line, MTS prints the prefix #–.

For certain applications, the use of the minus sign as the line-continuation character may be awkward; the CONTCHAR option of the SET command allows the user to assign a different character for this function. For example:

```
SET CONTCHAR=*
```

replaces the minus sign with an asterisk as the line-continuation character.

It should be noted that these rules for continuation lines apply particularly to MTS command mode input lines; continuation conventions are also available for some, but not all, other modes of operation and for some, but not all, system programs described in *MTS Volume 2: Public File Descriptions*, Reference R1002.

## FILE-NAME PATTERNS

Several MTS commands that take a file name (or list of file names) as a parameter allow the file name to be a file-name pattern. This pattern is used to represent a set of file names in the user's catalog of files. The commands that currently allow file-name patterns are

COPY
DESTROY
DUPLICATE
EMPTY
EDIT
FILESTATUS
FILEMENU

LIST
LOCKSTATUS
PERMIT
RENAME
RENUMBER
TRUNCATE

The presence of the "?" character in the file-name parameter is used to indicate a file-name pattern. A single "?" will match zero or more arbitrary characters in the file name. Thus,

| | |
|---|---|
| ?.S | represents all files whose names end with ".S", |
| A?B | represents all files whose names begin with "A" and end with "B", and |
| A?Q?B | represents all files whose names begin with "A", end with "B", and contain the letter "Q". |

"n" consecutive "?" characters will match "n−1" arbitrary characters in the file name. Thus,

| | |
|---|---|
| ???.s | represents all files whose names are four characters long and end with ".S", such as XX.S. |

The "?" cannot be used in the userID portion of a shared file name.

For example, the following COPY command will copy to *PRINT* all the user's files that begin with the characters "DATA":

```
COPY DATA? *PRINT*
```

## MTS COMMANDS

The following notation conventions are used in the prototypes of the commands:

| | |
|---|---|
| lowercase | represents a generic type which is to be replaced by an item supplied by the user. |
| uppercase | indicates material to be repeated verbatim in the command. |
| brackets [ ] | indicates that material within the brackets is optional. |
| braces { \| } | indicates that the material within the braces represents choices, from which exactly one must be selected. The choices are separated by vertical bars. |
| ellipsis **...** | indicates that the preceding syntactic unit may be repeated. |
| underlining | indicates the minimum unambiguous form of the command or parameter. Longer abbreviations are accepted. |

The following pages give a complete summary of the commands in the MTS command language.

## Summary of MTS Command Prototypes

```
ACCOUNTING [{statusopts | MANAGEMENT}]

ALTER location value ... ...

    location    GRx
                FRx
                [RF={hhhhhh | GRx}] xxxxxx
    value       {hhhh | X'hhhh'}
                C'xxxx'
                F'yyyy'
                H'yyyy'

CALC [expression]

CANCEL {*...* | *...* [JOB] nnnnnn | [JOB] nnnnnn} [ID=userid]

    *...*       {*PRINT* | *PUNCH* | *BATCH*}

COMMENT [text]

CONTROL FDname control-command

COPY [FROM] {FDlist1 | 'string'} [[TO] FDlist2]

CREATE {filename | *pdn*} [keywords]

    keywords    SIZE={n | nP}
                MAXSIZE={n | nP}
                TYPE={LINE | SEQ | IMPORT | EXPORT | DUMMY}

DEBUG [program] [I/Ounits] [options] [PAR=parameters]

    (see RUN command)

DESTROY {filelist | *pdn*} [{OK | ALLOK | PROMPT}]

DISPLAY [OUTPUT=FDname] [format] {location | item} ...

    format      {HEX | NOHEX}
                {MNEMONICS | NOMNEMONICS}
                {EBCDIC | NOEBCDIC}
                {SINGLESPACE | DOUBLESPACE}
                ORL={LONG | SHORT | n}

    location    {GRx | GRS}
                {FRx | FRS}
                [RF={hhhhhh | GRx}] xxxxxx[...xxxxxx]
                PSW
```

| | |
|---|---|
| item | ADDRESS |
| | AUTOHOLD |
| | CARDS |
| | CLASS |
| | COMMENT |
| | CONTCHAR |
| | COPIES |
| | COST |
| | CROUTE |
| | DATE |
| | DEBUG |
| | DELIVERY |
| | DESTINATION |
| | {EBM \| ETM} |
| | ERRORPROMPT |
| | EXECPFX |
| | FILE |
| | FORMAT |
| | GUINFO(name) |
| | HELPMODE |
| | HOSTNAME |
| | INITFILE(command) |
| | INSTALLATIONNAME |
| | JOBNAME |
| | {LASTRELOAD \| LRL} |
| | LIBSRCH |
| | LOGSTATUS |
| | MACHINE |
| | MAILCALL |
| | MAP |
| | MAPDOTS |
| | MARGIN |
| | MODEL |
| | NAME |
| | NAMELIB |
| | {NEWFILEACCESS \| NFA} |
| | NUMBER |
| | OVERLAY |
| | PAGES |
| | PAPER |
| | PASSWORD |
| | PDNS |
| | PRINT |
| | PRINTER |
| | PROJECT |
| | PROJECTPWCHANGE |
| | PROUTE |
| | RATES |
| | RCPRINT |
| | RERUN |
| | RF |
| | RUNS |

```
                    SEE_DISPATCHES
                    SENSE(devicename)
                    SEPCOPY
                    SHIFT
                    SHOWNAME
                    SIGFILE
                    SINK
                    SOURCE
                    SPELLCOR
                    SRVREPLY
                    SYMTAB
                    SYSTEM
                    TAILSHEET
                    TASKNUMBER
                    TDR
                    TERSE
                    TIME
                    TIMEDATE
                    TIMESPELLEDOUT
                    TWOSIDED
                    UPTIME
                    USERID
                    USMSG
                    UXREF
                    VERSION(command)
                    VMSIZE
                    XREF
                    *BATCH*
                    *PRINT*
                    *PUNCH*
                    *...*
                    *pdn*

DUMP [OUTPUT=FDname] [format] ...

    format          {HEX | NOHEX}
                    {MNEMONICS | NOMNEMONICS}
                    {EBCDIC | NOEBCDIC}
                    {SINGLESPACE | DOUBLESPACE}
                    ORL={LONG | SHORT}
                    {LIBRARY | NOLIBRARY}

DUPLICATE oldname [{AS | TO}] newname [keywords] [{OK | ALLOK | PROMPT}]

    keywords        EMPTYOK
                    {OPTIMIZE | NOOPTIMIZE}
                    {DATA | NODATA}

EDIT [filename] [:edit-command]

EMPTY filelist [{OK | ALLOK | PROMPT}]
```

```
{FILEMENU | FMENU} [name] [information]

FILESTATUS [name] [format] [information]

FSMESSAGE [FSMessage-command]

FTP [hostname]

{HELP | EXPLAIN} [topic]

IF RUNRC condition integer, MTS-command

     condition    =,~=,<,>,<=,>=
                  .EQ.,.NE.,.LT.,.GT.,.LE.,.GE.

LIST FDlist [[{ON | TO}] FDname] [[WITH] options]
LIST FDlist WITH options [{ON | TO} FDname]

LOAD [program] [I/Ounits] [options] [PAR=parameters]

     (see RUN command)

LOCATE {SYSTEM | LOCAL | FULL | SHORT | HELP}
LOCATE {jobnumber | jobname} [option ...]

     option       PRINT
                  EXECUTE
                  IMPORT
                  VIEW
                  PRINTQ
                  SUMMARY

LOCK filename [how] [option ...]

     how          READ
                  {WRITE | MOD | CHANGE | EMPTY | TRUNCATE | RENUMBER}
                  {DESTROY | RENAME | PERMIT}
                  NONE

     option       {WAIT | NOWAIT}
                  {QUIT | NOQUIT}

{LOCKSTATUS | LSTATUS} [item [options]]

     item         filelist
                  JOB {nnnnnn | ME}
                  USER=userid

     option       LOCK
                  LOCKDESTROY
                  LOCKMOD
                  LOCKREAD
```

```
                 WAIT
                 WAITDESTROY
                 WAITMOD
                 WAITREAD
                 WAITOPEN
                 INVALID
                 OPEN
                 NOTONOTL
                 ANY
                 REPEAT=seconds
                 OUTPUT=FDname

LOG [FDname1] {[ON] FDname2 [format] [options] | OFF}

MAKE program [WITH] options

MESSAGE [message-command]

MODIFY location value ... ...

     (see ALTER command)

MOUNT [request [;request] ...]

MTS

NET [{hostID | *pdn*}] [.network-command]

PERMIT filelist [access [accessor]]
PERMIT filelist LIKE filelist2 [EXCEPT access [accessor]]

     access      READ
                 {WRITEXP | WE | APPEND}
                 {WRITECHG | WC | EMPTY}
                 TRUNCATE
                 {DESTROY | RENAME}
                 PERMIT
                 RCWHG
                 RWEXP
                 RW
                 FULL
                 UNLIM
                 DEFAULT
                 NONE
                 RUN
                 EDIT

     accessor    ALL
                 ME
                 OTHERS
                 OWNER
                 [ID=]userid
```

```
                    PROJECT=project
                    PKEY=key
                    [ID=]userid&PKEY=key
                    PROJECT=project&PKEY=key

RELEASE {*PRINT* | *PUNCH* | *BATCH* | *pdn*}

RENAME oldname [AS] newname [{OK | ALLOK | PROMPT}]

RENUMBER filelist [first [last [beg [inc]]]] [{ALLOK | PROMPT}]

RERUN [{ECHO | NOECHO}] [I/Ounits] [options] [PAR=parameters]

    (see RUN command)

RESTART [[AT] location] [I/Ounits] [options]

    (see RUN command)

RUN [program] [I/Ounits] [options] [PAR=parameters]

    options     MAP[=mapFDname]
                NOMAP
                XREF
                UXREF
                {EXECPKEY | PKEY}={key | OFF}
                TIME={t | tS | tM}
                PAGES=p
                CARDS=c
                {PLOTTIME | PT}={t | tM}

    I/Ounits    INPUT=FDname          (defaults to *SOURCE*)
                PRINT=FDname          (defaults to *SINK*)
                SPUNCH=FDname         (defaults to *PUNCH* in batch)
                GUSER=FDname          (defaults to *MSOURCE*)
                SERCOM=FDname         (defaults to *MSINK*)
                {0 | 1 | ... | 99}=FDname

SDS [debug-command]

SET option ...

    option      ADDRESS="line1;line2;..."
                AUTOHOLD={ON | OFF}
                CLASS=char
                CMDSCAN={AMBIGUOUS | UNAMBIGUOUS}
                CMDSKIP={ON | OFF}
                COMMENT="text"
                CONTCHAR=character
                COPIES=n
                COST={ON | OFF}
                CROUTE=station
```

```
DEBUG={ON | OFF}
DELIVERY={station | MAIL | NONE}
DESTINATION=userid@node
DISPATCHES[(filter)]={ON | OFF}
EBM=characters
ECHO={ON | OFF}
ENDFILE={ALWAYS | SOURCE | NEVER}
ERRMAP={ON | OFF}
ERRORDUMP={NOLIBRARY | OFF | LIBRARY}
ERRORPROMPT={ON | OFF}
ETM=characters
{EXECPFX | EXECPREFIX}=character
EXECPKEY={key | OFF}
FILE={filename | "file name"}
FORMAT={LANDSCAPE | PORTRAIT | TWOUP | format-name}
HELPMODE={LINE | DEFAULT | SCREEN}
IC={ON | OFF}
INITFILE(command)={FDname | OFF}
JOBNAME={jobname | DEFAULT}
LIBR={ON | OFF}
LIBSRCH={OFF | FDname}
MACROS={ON | OFF}
MAILCALL={ON | OFF}
MAPDOTS={ON | OFF}
MARGIN={n.nn | NO}
NAME={name | 'name' | NONE}
NAMELIB={FDname | OFF}
{NEWFILEACCESS | NFA}={'string' | "string" | OFF}
NUMBER={(b,l,c) | NO}
OVERLAY={NONE | SHADED | LINED}
PAGES=n
PAPER={PLAIN | 3HOLE | LABEL24 | LABEL33}
PARFIELDCASE={UC | MC}
PDMAP={ON | OFF}
{PFX | PREFIX}={ON | OFF}
PRINT={T3 | TN}
PRINTER={PAGE | LINE}
PRMAP={ON | OFF}
PROJECTPWCHANGE={ON | OFF}
PROUTE=station
{PW | PASSWORD}
RCPRINT={NEVER | OFF | POSITIVE | NONZERO |
      NONNEGATIVE | ALWAYS | ON}
RF={hhhhhh | GRx}
ROUTE=station
SEPCOPY={YES | NO}
SEQFCHK={ON | OFF}
SHIFT={YES | NO}
SHOWNAME={ON | OFF}
SIGFILE={OFF | FDname}
SIGFILEATTN={ON | OFF}
SPELLCOR={OFF | PROMPT | ON}
```

```
                    SRVREPLY={ON | OFF}
                    SYMTAB={ON | OFF}
                    TDR={ON | OFF}
                    TERSE={ON | OFF}
                    {T | TIME}={t | tS | tM | OFF}
                    TRIM={ON | OFF}
                    TWOSIDED={YES | NO}
                    USMSG={ON | OFF}
                    UXREF={ON | OFF}
                    VERSION(command)={NEW | OLD | CURRENT}
                    XREF={ON | OFF}
                    *LIBRARY={ON | OFF}

SIGNOFF [{SHORT | $ | LONG}]

SIGNON {userid | *} [option ...] ['comment']

          option    ADDRESS="line1;line2;..."
                    CARDS=c
                    COPIES=n
                    CROUTE=station
                    DELIVERY={station | MAIL | NONE}
                    FORMAT={LANDSCAPE | PORTRAIT | TWOUP | format-name}
                    JOBNAME={jobname | DEFAULT}
                    {LANDSCAPE | PORTRAIT | TWOUP}
                    MARGIN={n.nn | NO}
                    {NOMESSAGES | NOMSGS}
                    {ONESIDED | TWOSIDED}
                    OVERLAY={NONE | SHADED | LINED}
                    PAGES=p
                    PAPER={PLAIN | 3HOLE | LABEL24 | LABEL33}
                    {PLOTTIME | PT}={t | tM}
                    PRINT={T3 | TN}
                    PRINTER={PAGE | LINE}
                    PROUTE=station
                    QUICK
                    ROUTE=station
                    RATEGROUP={NORMAL | LOW | DEFERRED | MINIMUM}
                    RERUN={YES | NO}
                    SEPCOPY={YES | NO}
                    SHIFT={YES | NO}
                    {SHORT | LONG}
                    SIGFILE={ON | OFF}
                    TAPES=n
                    TIME={t | tS | tM}
                    WAITUNTIL='time and/or date'

SINK {FDname | PREVIOUS}

SOURCE {FDname | PREVIOUS}
```

```
START [[AT] [RF={hhhhhh | GRx}] location] [I/Ounits] [options]

          (see RUN command)

SYSTEMSTATUS [option]

          option      DISPLAY disp[+disp[+...]] [count]
                      LOAD [job#]
                      MCMD MTS-command
                      MTS
                      QUEUE [{receipt | USER | ALL | ROUTE=station | *} ...]
                      REPEAT [TIME=n]
                      RETURN
                      STOP
                      TAPEQUEUE [LIST]
                      TASKS [descriptor]

                      descriptor   job#
                                   B
                                   F
                                   M
                                   N [job-name]
                                   D device-name
                                   T device-type
                                   U userid
                                   P projectid

                      USERS
                      $MTS-command

TRUNCATE filelist [{ALLOK | PROMPT}]

UNLOAD [CLS=clsname] [EVERYTHING]

UNLOCK filename

VIEW [jobnumber [;view-command]]
```

## ACCOUNTING

MTS Command Description

Purpose: To invoke the accounting system.

Prototype: ACCOUNTING [{statusopts | MANAGEMENT}]

statusopts

"statusopts" specify one or more options that may be used to selectively filter the status information given about the current userID. The options are as follows:

FULL

Print all information. In addition to the items listed under NOFULL below, the following quantities are printed:

amount of temporary file space
cumulative figures for file storage
CPU and wait-memory used
CPU time used
number of tape mounts
tape-drive time used
lines, images, sheets, and pages printed
phototypesetter units and media used
cards read and punched
plotter paper used
UMnet/Michnet network charges
surcharges and royalties
number of batch and terminal sessions
expiration date and time

FULL is the default in batch mode if no options other than HEADING or NOHEADING are specified.

{NOFULL | –FULL | ˜FULL}

Print the maximum, used, and remaining figures for the following items:

charge
current file space
concurrent signons
terminal time
plotter time
UMnet/Michnet Computer Network originate time

NOFULL is the default in conversational mode if no options other than HEADING or NOHEADING are specified.

HEADING

Print a heading before the next line that contains a used amount. This is the default for the first line printed.

{NOHEADING | –HEADING | ¬HEADING}

Do not print a heading. If this option is specified, it should be first.

{CHARGE | DOLLARS | FUNDS | $}

Print the remaining amount of funds.

{DISK | FILE}

Print the remaining amount of file space.

EXPIRE

Print the expiration date and time of the account.

NETWORK

Print the remaining amount of outbound network connect time available.

PLOTTER

Print the remaining amount of plotter time available.

SIGNONS

Print the remaining number of concurrent signons permitted.

{TERMINAL | CONNECT}

Print the remaining amount of terminal connect time available.

One of the following modifiers may be appended to the CHARGE, DISK, SIGNONS, TERMINAL, PLOTTER, or NETWORK parameters or their synonyms. If a modifier is to apply to more than one parameter, the parameters may be separated by commas and grouped within parentheses, e.g., ($,DISK)@D.

{@DETAILED | @FULL | @NOREMAINING | @–REMAINING | @¬REMAINING}

Print the maximum, used, and remaining figures for the modified quantities rather than only the remaining amounts.

{@REMAINING | @NODETAILED | @–DETAILED | @¬DETAILED | @NOFULL | @–FULL | @¬FULL}

Print only the remaining amounts for the modified quantities. This is the default if a modifier is not specified.

MANAGEMENT

If the MANAGEMENT option is specified, accounting management mode is entered.

If no options are specified on the command, status information is printed about the current userID. This is equivalent to specifying the FULL option in batch mode or the NOFULL option in conversational mode.

Program Key: *ACCOUNTING

Description: The ACCOUNTING command may be used to print information regarding the userIDs charge; current and cumulative file space; signons; terminal, plotting, and network time; CPU and wait-memory use; CPU time; I/O; and expiration time.

If the command is given in conversational mode and no options are specified (other than HEADING or NOHEADING), the items listed for the parameter NOFULL are printed. If the command is given in batch mode and no options are specified (other than HEADING or NOHEADING), the items listed for the parameter FULL are printed in addition to those listed with NOFULL. If all information about an item is zero, no information normally is printed unless the item is specified as a “statusopt”. The information is current at the time the command is given with the exception that tape drive time and paper tape punched as well as the associated charges for these are not included for tapes currently mounted, nor are charges included for a concurrent signon using the same userID.

It must be emphasized that the information printed is only approximate. A user’s true position is indicated only by the monthly billing.

Accounting management mode allows project directors and instructors to distribute money, permanent disk space, and terminal and plotting time to various userIDs belonging to their project or class. Also, the expiration time and maximum number of concurrent signons for individual userIDs may be changed. Before a project can use management mode, one of the userIDs belonging to the project must receive authorization by contacting the ITD Accounts Office. For the details of accounting management mode, see the section “Accounting” in *MTS Volume 5: System Services*, Reference R1005.

Examples:

```
ACCOUNTING CHARGE
```

In the above example, the amount of funds remaining in the user’s account is printed.

```
ACCOUNTING NOFULL EXPIRE
```

In the above example, the information that is normally listed when the command is given in conversational mode plus the expiration date and time is printed.

```
ACCOUNTING ($,DISK,SIGNONS)@D PLOTTER
```

In the above example, the maximum, used, and remaining funds, file space, and concurrent signons, as well as the remaining plotter time, are printed.

## ALTER

MTS Command Description

Purpose: To alter the contents of a general register, floating-point register, or specified virtual memory location(s).

Prototype: ALTER location value ... ...

Each alteration requires a pair of the following parameters, the first specifying what is to be altered and the second specifying the new contents. Any number of items may be altered with a single ALTER command.

location

“location” is the general register, floating-point register, or virtual memory location(s) that is to be altered. “location” may be given in one of the following forms:

GRx

GRx specifies the general register “x”, where “x” is a decimal integer from 0 to 15 or a hexadecimal integer from 0 to 9, A to F.

FRx

FRx specifies the floating-point register “x”, where “x” is one of the integers 0,2,4, or 6.

[RF={hhhhhh | GRx}] xxxxxx

This specifies a virtual memory location given by an *optional* local relocation factor and a displacement. “hhhhhh” is the hexadecimal value of a local relocation factor; GRx indicates the general register whose contents are to be used as a local relocation factor. “xxxxxx” is the hexadecimal value of a displacement. The displacement is added to the current value of the relocation factor to provide an absolute 24-bit virtual memory address. If a local relocation factor is not specified, the global relocation factor is used. The global relocation factor is initially zero, but may be changed by the RF option of the SET command. When a local relocation factor is specified in the command, it remains in effect for the remainder of the command unless subsequently overridden by a second local relocation factor specification.

value

The new contents are specified by any one of the following constant expressions:

hhhh or X'hhhh'

> Any hexadecimal constant expression of any length may be given.

C'xxxx'

> Any EBCDIC character expression of any length may be given between the delimiting primes; a prime in the character string must be represented by two consecutive primes.

F'yyyy' or H'yyyy'

> A fullword (F) or halfword (H) decimal constant expression may be given consisting of a sign followed by decimal digits all enclosed in primes. The plus sign (+) is optional; the minus sign (−) is required. Decimal constants may not be specified for floating-point registers.

Program Key: *MTS.ALTER

Description: The new constant given by the parameter "value" replaces the contents of the register or virtual memory location specified by "location". Register numbers and virtual memory addresses are checked for validity and an error comment is produced if an illegal value is specified. If a virtual memory address is given, it must not be less than 600000. The parameter pairs in the command are processed from left to right.

If the currently loaded program has a nondefault program key that is not prefixed with the current userID, the program key will be set to the default (for this invocation of the program only).

This command is invalid if the currently loaded program is a "run-only" program.

General registers are altered as follows:

> A character constant is truncated or padded with trailing blanks to four bytes (characters) and placed, left-justified, into the register.
>
> The integer value of a hexadecimal constant (consisting of one to eight hexadecimal digits including leading zeros) is placed, right-justified, into the register.
>
> The integer value of a decimal constant is placed into the register.

Floating-point registers are altered as follows:

> A character constant is truncated or padded with trailing blanks to eight bytes (characters) and placed, left-justified, into the register.
>
> A hexadecimal constant is truncated or padded with trailing zeros to eight bytes and placed, left-justified with leading zeros retained, into the register.

Virtual memory is altered as follows:

A character constant is placed, one character per byte, into consecutive virtual memory locations.

A hexadecimal constant is placed, two hexadecimal digits per byte with leading zeros retained, in consecutive memory locations. If an odd number of hexadecimal digits is given, the last byte of memory altered has bits 4-7 set to zero.

The integer value of a decimal constant is placed, *without* regard to boundary alignment, into either four or two bytes (for fullword or halfword constants) starting with the byte specified by “location”.

Examples:

```
ALTER GR3 1A3E0 FR6 X'41104'
```

The hexadecimal constant 0001A3E0 is placed in GR3 and the hexadecimal constant 4110400000000000 is placed in FR6.

```
ALTER RF=6188E2 200 X'D502CC7E6000' 400 X'05EF' GRA 0
```

The hexadecimal constant D502CC7E6000 is placed in virtual memory location 618AE2, the hexadecimal constant 05EF is placed in location 618CE2, and the constant zero is placed in GR10.

## CALC

MTS Command Description

Purpose: To enter calc mode for performing desk-calculator operations.

Prototype: CALC [expression]

expression

"expression" is an optional arithmetic expression to be evaluated.

Program Key: *CALC

Description: Calc mode provides a desk-calculator facility in MTS. This facility evaluates mathematical expressions using double-precision, floating-point arithmetic. The result may be presented in several formats.

The CALC command may be used in two ways. If the optional parameter, a single arithmetic expression, is given with the command, the expression is evaluated, the result is printed, and control is returned to MTS command mode. If the expression is omitted, calc mode is entered. In calc mode, several expressions may be evaluated.

Once in calc mode, the user is prompted for expressions to be evaluated. The expressions are given in the form:

[variable=]expression[@format]

where

| | |
|---|---|
| "variable" | is a 1- to 8-character (alphanumeric) variable name (the first character must be a letter), |
| "expression" | is an arithmetic expression consisting of numeric constants, variable names, functions, and the arithmetic operators +, –, *, /, and ** (exponentiation). |

If the input line is in the form of an assignment statement, the value of "expression" is assigned to "variable". There is no limit to the number of variables which may have values assigned to them, but a variable must be assigned a value before it is used in an expression. All variables contain double-precision, floating-point, scalar values.

The user may return to the caller (normally MTS command mode) via the MTS, RETURN, or STOP commands, or an end-of-file.

Output

If the input line is an expression which does not assign a value to a variable, the result of the evaluated expression is printed. An expression may consist of a single element or a number of elements. The manner in which the result is printed is controlled by a format specification. The format may be defaulted, set to a particular specification for a single expression, or set to a global specification for all expressions.

When calc mode is entered, the default format specification is in effect. With this format, the result is printed as a decimal number (up to 16 significant digits) if the expression contains decimal constants or variables; the result is printed as a hexadecimal number if the expression contains only hexadecimal constants or variables. If the expression consists of only a single number, the result is printed in decimal if the number is entered in hexadecimal format, and in hexadecimal if the number is entered in decimal format. If the result is to be printed in hexadecimal format, but it is not an integer, it is printed in floating-point hexadecimal format (excess-40 notation). If the expression consists of only a single variable name, the value of the variable is printed in decimal. Finally, if the expression consists of only the symbol "$", the result of the preceding expression is printed in the format opposite to the previous printing.

In addition to arithmetic expressions as input, a global format specification may be specified by the SET calc command. All subsequent results are printed according to this format specification. This format is given in the form:

SET $\underline{\text{F}}$ORMAT=fmt

where "fmt" may be any FORTRAN-type numeric field specification (I,F,E,G,Z), X (for hexadecimal), RX (for floating-point hexadecimal), and GREG (for Gregorian date conversion). For example,

```
SET FORMAT=F10.2
```

sets the format to the FORTRAN floating-point 10.2 format. If "fmt" is omitted, the format specification is reset to the default.

A format specification may also be appended to an expression (see expression prototype above). This format overrides the global format and affects only that expression. Thus, a format may be set for a single expression. When appended to an expression, the "FORMAT=" may be omitted. Thus,

```
1.23@F10.2
```

is equivalent to

```
1.23@FORMAT=F10.2
```

## Expressions

Expressions are the basic input to calc mode. The operands of an expression may be constants, variables, or function invocations. The operators may be the arithmetic operators +, −, *, /, and ** (exponentiation). Expressions may be parenthesized; all blanks are ignored. Note that operators of equal precedence are evaluated from left to right (e.g., 2**2**3 is equivalent to (2**2)**3 and gives the result 64); the user may use parentheses to override this.

Constants may be entered in any of four forms: decimal, hexadecimal, floating-point hexadecimal, and Gregorian date. All constants are converted to double-precision, floating-point numbers before the expression is evaluated.

(1) Decimal constants may be entered with or without a decimal point and with an optional exponent in the form E±nn.

(2) A hexadecimal constant is of the form X'hhhhhhhh' or 'hhhhhhhh'. One to eight hexadecimal digits may be given; if fewer than eight are given, the constant is padded on the left with zeros to eight digits.

(3) A floating-point hexadecimal constant is of the form R'ccmmmmmm'. The constant consists of a two-digit characteristic in excess-40 notation and a mantissa of one to fourteen digits.

(4) A Gregorian date is of the form GR'mm/dd/yy' or G'mm/dd/yy'; the digits “mm”, “dd”, and “yy” specify the month, the day, and the year, respectively. Gregorian dates are converted to their equivalent Julian dates before they are evaluated in an expression. The Julian date is computed as the number of days since March 1, 1900.

Variables may have names consisting of one to eight alphanumeric characters, the first of which must be a letter. A variable is created when it is first used on the left side of an assignment expression; it remains defined until calc mode is terminated by a STOP command or an end-of-file. In addition, the special variable “$” is always defined and contains the result of the last valid expression.

Functions may be used as operands in expressions. A function is given in the form

    name(expression,**...**)

where “name” is the function name and “expression” is a calc mode expression which may be passed as an argument to the function. Any of the functions in the elementary function library may be used (see *MTS Volume 3* for a description of these functions). Among the functions included are SQRT, EXP, LOG, LOG10, SIN, COS, TAN, COTAN, ARSIN, ARCOS, ATAN, SINH, COSH, TANH, ERFC, ERF, DLGAMA, and GAMMA. All functions use the double-precision form in the elementary function library.

Commands: Any of the calc commands described below may be used in calc mode.

CLEAR

The CLEAR command undefines all currently defined variables except for the "$" variable.

LIST

The LIST command lists all currently defined variables and their values.

MCMD MTS-command

The MCMD command executes an MTS command while in calc mode.

MTS [MTS-command]

The MTS command returns control to the caller (normally MTS command mode). If an MTS command is specified, that command is immediately executed before control returns to the caller. All current variable definitions and format specifications are retained.

RETURN

The RETURN command returns control to the caller (normally MTS command mode). All current variable definitions and format specifications are retained.

SET keyword **...**

The SET command controls several calc mode options. The options which may be set are:

| | |
|---|---|
| DIGITS=[n] | "n" specifies the number of digits that are printed to the right of the decimal point for a decimal number. If more digits are available than are to be printed, the number is rounded to the nearest digit. If "n" is omitted, it is reset to the default (all digits are printed). |
| [FORMAT=][fmt] | "fmt" specifies the global format specification. FORMAT may be omitted; if "fmt" is omitted, the format is reset to default format processing. |
| OUTPUT=[FDname] | "FDname" specifies the file or device to which expression results are written. If "FDname" is omitted, the output device is reset to *SINK*. |
| SIGDIGITS=[n] | "n" specifies the number of significant digits that are printed for a decimal number. If "n" is omitted, it is reset to the default of 16. |

STOP

The STOP command (or an end-of-file) terminates calc mode and returns control to the caller (normally MTS command mode). All current variable definitions and format specifications are released.

Examples: The following two terminal session examples illustrate the use of the CALC command. User input to calc mode is prefixed by "#" or "?"; output is prefixed by "=".

```
#CALC SQRT(2)
=1.414213562237309492


#CALC
?X=SIN(2*3)
?Y=COS(3*2)
?X**2+Y**2
=1
?X@FORMAT=F10.5
=  -0.27942
?SET FORMAT=E20.14
?Y
=0.96017028665037E+00
?SET FORMAT=
?Y
=.960170286650365995
?Y@RX
=R'40F5CDB84BC117AA'
?X**2+Y**2@X
=X'00000001'
?STOP
```

## CANCEL

MTS Command Description

Purpose: To cancel *PRINT*, *PUNCH*, or *BATCH* jobs submitted from a terminal, or regular batch jobs that the user has previously submitted.

Prototype: CANCEL {*...* | *...* [JOB] nnnnnn | [JOB] nnnnnn} [ID=userid]

where the parameters may be given in any order.

*...* [JOB] nnnnnn

"*...*" is either *PRINT*, *PUNCH*, or *BATCH*, and "nnnnnn" is the 6-digit job number or the job name. If the appropriate "*...*" has not been released, the associated job is canceled; in this case, the job number or name is not required. If the "*...*" job has been released to the system, the job number or name must also be specified.

If both the "*...*" name and the job number (or job name) are given, and the number does not match the number of the currently open "*...*" job, it is assumed that the user is referring to an earlier "*...*" job of the same type and the cancel information is passed to the system. If both the "*...*" name and the job number are given, but the job number is that belonging to a different type of "*...*" name, an error comment is printed.

{ID | USER}=userid

"userid" is the ITD userID associated with the job to be canceled. If the userID given with the command is different from the current userID, the terminal user is prompted for the password.

Program Key: *MTS.CANCEL

Restrictions: An "*...*" job cannot be canceled once processing has begun by the system. Thus, *PRINT* can be canceled only if it is awaiting print, *PUNCH* if it is awaiting punch, and *BATCH* if it is awaiting execution. To locate a job in the system, the SYSTEMSTATUS command may be used.

Canceling an *PRINT* or *PUNCH* job while it is still open causes the page-line or punch charges for the job to be rebated automatically. Once the job has been released to the system, it still can be canceled in some cases (subject to the constraints mentioned above). Charges are rebated automatically only if the job is canceled during the same session in which it was released. In all other cases, the user must explicitly apply for a rebate.

Examples:

```
CANCEL *PRINT*
```

The current *PRINT* job is canceled.

```
CANCEL 610399
```

The job with job number 610399 is canceled.

```
CANCEL *PUNCH* 603321
```

The *PUNCH* job with job number 603321 is canceled.

## COMMENT

MTS Command Description

| | |
|---|---|
| Purpose: | To allow insertion of comments on output to the terminal or the printer. |
| Prototype: | COMMENT [text] |
| Program Key: | *MTS.COMMENT |
| Description: | This command is ignored by the system. As with all commands, it is echoed on *SINK* and *MSINK*, unless the command SET ECHO=OFF has been given. |
| | Comments also may be inserted into the output by prefixing them with an asterisk (*) or dollar sign ($). This type of comment is not echoed on *SINK* or *MSINK*. |

Examples:

```
COMMENT This is a comment.

* This is also a comment.
```

## CONTROL

MTS Command Description

| | |
|---|---|
| Purpose: | To control the operation of certain types of files and devices. |
| Prototype: | CONTROL FDname control-command |
| Program Key: | *MTS.CONTROL |

Description: The "FDname" parameter specifies the file or device on which the control operation specified by the "control-command" parameter is to be performed. The "control-command" parameter includes all characters beginning with the first nonblank character after the "FDname" parameter. The control command is converted to uppercase before it is processed by the system.

Control commands may also be processed from user programs via the CONTROL subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Only certain types of files and devices allow control operations to be performed. A list of the acceptable devices and their control commands follows. The codes in parentheses refer to the device type returned by the GDINFO subroutine (see *MTS Volume 3*).

Magnetic Tapes (9TP):

| *Control Command* | *Function* |
|---|---|
| Positioning: | |
| REW | Rewind |
| FSR [n] | Forward space "n" records |
| BSR [n] | Backspace "n" records |
| FSF [n] | Forward space "n" files |
| BSF [n] | Backspace "n" files |
| POSN={* \| *n* \| *EOT* \| name} | Position to beginning of current file, nth file, end-of-tape, or file name |
| Blocking: | |
| {FORMAT \| FMT \| RECFM}=fmt[([size][,lrecl])] where "fmt" is {U \| F \| FB \| FBS \| V \| VB \| VS \| VBS \| D \| DB} | Specify blocking format and, optionally, block size and/or logical record length |
| {SIZE \| BLKSIZE}=n | Specify blocksize (18 ≤ n ≤ 32767) |
| LRECL=n | Specify logical record length (1 ≤ n ≤ 32767) |
| BLK={ON \| OFF} | Enable or disable blocking |

Label Processing:

| | |
|---|---|
| {DSN \| NAME}[=name] | Specify name for next new file written |
| DTCHK={ON \| OFF} | Enable or disable expiration date checking |
| EXPDT=[{mm-dd-yy \| mm/dd/yy}] | Specify file expiration date for new files |
| INIT | Initialize (empty) a labeled tape |
| LP={ON \| OFF} | Enable or disable label processing |
| EOV | Terminate tape with end-of-volume labels |
| BLKPFX=n | Specify block-prefix length for ANSI labeled tapes (0 ≤ n ≤ 99) |
| CC={A \| M} | Specify control character for next new file written |

Error Recovery:

| | |
|---|---|
| RETRY=n | Specify read error retry count (0 ≤ n ≤ 15) |
| SNS | Return sense data |

Miscellaneous:

| | |
|---|---|
| {WTM \| EOF} [n] | Write "n" tape-marks |
| MODE=mode | Specify tape mode |
| PUSH | Push current tape parameters onto stack |
| POP | Pop tape parameters from stack |
| MINSIZE=n | Specify minimum block size (1 ≤ n ≤ 100) |
| TIMER={ON \| OFF \| n} | Enable, disable, or specify elapsed-time interval (in minutes) for inactive tape warning message; defaults to ON, n=15 minutes |
| TRANSLATE={MTS(parity) \| MTS \| IBM \| NONE} | Specify translation scheme for ANSI-labeled or ASCII unlabeled tapes; parity is EVEN, ODD, ZERO, or ONE; defaults to MTS(ZERO) |

Note that in the FSR, BSR, FSF, BSF, and WTM commands, "n" must be in the range from 0 to 32767. If omitted, "n" defaults to 1. For a complete description of these commands, see *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019.

UMnet/Michnet Computer Network (MNET):

Any of the UMnet/Michnet Computer Network device commands as normally entered after a percent sign (%) may be specified. The percent sign should *not* be given as part of the device control information. For a complete description of the device commands, see the section "The UMnet/Michnet Computer Network" in *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004.

IBM 3270 Display Device Commands (3270):

Any of the IBM 3278 device commands as normally entered after a percent sign (%) may be specified. The percent sign should *not* be given as part of the device control information.

Files (SEQF, FILE):

| *Control Command* | *Function* |
|---|---|
| EMPTY | Empty the file |
| TRUNCATE | Truncate the file |
| RENUMBER par | Renumber the file |
| {PKEY \| PGMKEY}=key | Set the program key |
| SIZE={n \| nP} | Change the size of the file |
| MAXSIZE={n \| nP} | Change the maximum size |
| SIZEINC=[±]{n \| nP} | Increment the size |
| MAXSIZEINC=[±]{n \| nP} | Increment the maximum size |
| {BUFFERS \| MAXBUFS}=n | Set the number of file buffers |
| EXPFAC={nP \| n% \| DEFAULT} | Set the expansion factor for a file |
| SAVE | Save the file on the system file-save tapes |
| NOSAVE | Do not save the file on the system file-save tapes |
| TOUCH | Update the last data change time to the current time |

For a complete description of these commands, see the section “Control Options for Files” in the section “Files and Devices” in this volume.

*PRINT*, *PUNCH*, *BATCH* (RMPT, RMPC, RMBA)

| *Control Command* | *Function* |
|---|---|
| CANCEL | Hold job |
| COPIES=n | Specify number of copies of job |
| DELIVERY={station \| MAIL \| NONE} | Specify output delivery station |
| HOLD | Hold job |
| JOBNAME={jobname \| DEFAULT} | Specify job name |
| RELEASE | Release job |
| ROUTE=station | Route all output of job |

*PRINT* or *BATCH* only:

| | |
|---|---|
| COMMENT="text" | Specify head sheet text |
| FORMAT={LANDSCAPE \| PORTRAIT \| TWOUP \| format-name} | Specify output format |
| {LANDSCAPE \| PORTRAIT \| TWOUP} | Specify output orientation |

| | |
|---|---|
| MARGIN={n.nn \| NO} | Specify output margin |
| NUMBER={(b,l,c) \| NO} | Specify output page numbering |
| {ONESIDED \| TWOSIDED} | Specify number of printed sides |
| OVERLAY={NONE \| SHADED \| LINED} | Specify output overlay |
| PAPER={PLAIN \| 3HOLE \| LABEL24 \| LABEL33} | Specify output form |
| PRINT={T3 \| TN} | Specify line-printer character set |
| PRINTER={PAGE \| LINE} | Specify printer type |
| SHIFT={YES \| NO} | Specify output shifting |
| TWOSIDED={YES \| NO} | Specify number of printed sides |

*PRINT* only:

| | |
|---|---|
| ADDRESS="line1;line2;..." | Campus mail delivery address |
| PAGES=n | Specify number of pages |
| PROUTE=station | Route print output of job |
| SEPCOPY={YES \| NO} | Specify separate jobs for copies |

*PUNCH* only:

| | |
|---|---|
| CARDS=n | Specify number of cards |
| CROUTE=station | Route card output of job |

*IMPORT* and *EXPORT* (RMIM, RMEX)

| *Control Command* | *Function* |
|---|---|
| CLASS=char | Specify file class |
| FILE={filename \| "file name"} | Specify file name |

For *IMPORT* only:

| | |
|---|---|
| CANCEL | Cancel incoming job |
| HOLD | Hold incoming job in open state |
| JOBNAME=jobname | Specify job name |
| JOB#=jobnumber | Specify job number |
| KEEP | Requeue job |
| RELEASE | Release held job |

For *EXPORT* only:

| | |
|---|---|
| COMMENT="text" | Specify header comment |
| COPIES=n | Specify number of copies of job |
| DESTINATION=userid@node | Specify BITNET destination |
| ENCODING={NONE \| NETDATA \| DISKDUMP} | Specify data encoding |

TYPE={PUNCH | PRINT} Specify job type

For information on using the CONTROL command with with BITNET connections, see *BITNET in MTS*, Reference R1039.

Server Commands:

For information on using the CONTROL command with servers, see *Using and Creating MTS Network Servers*, Reference R1073.

Examples:

```
CONTROL *TAPE* REW
```

A REW control command is given to the pseudodevice *TAPE* which could be a 9-track magnetic tape.

```
COPY *SOURCE* CNTRLFILE
CON *MSINK* OUTLEN=100
RUN PROGRAM INPUT=INPUTDATA PRINT=*MSINK*
CON *MSINK* RESET
$ENDFILE
```

The file CNTRLFILE is established which can be used as a source file of commands for running a program that prints output lines up to 100 characters long on a terminal via the UMnet/Michnet Computer Network. The CONTROL commands set the terminal output length to 100 characters for the duration of the program run, and then reset the output length to the default value. This sequence of commands can be started with the command SOURCE CNTRLFILE.

## COPY

MTS Command Description

Purpose: To copy from a file or device (or set of files) to another file or device (or set of files).

Prototype: COPY [FROM] {FDlist1 | 'string'} [[TO] FDlist2]

Two FDname lists may be given as parameters:

FDlist1

"FDlist1" specifies the file or device (or set of files) that contain the lines to be copied (the input). "FDlist1" may be a single file or device name or a file-name pattern. An explicit concatenation of file or device names plus *one* file-name pattern may be specified. For example,

```
A?+B
```

is valid, but

```
A?+B?
```

is not valid. Line-number ranges and FDname modifiers may be included.

'string' or "string"

A single arbitrary character string (enclosed in single or double quotes) may be copied to the output set of files.

FDlist2

"FDlist2" specifies the file or device (or set of files) that is to receive the copied lines (the output). "FDlist2" may be a single file or device name or a file-name pattern. An explicit concatenation of file or device names plus *one* file-name pattern may be specified. "FDlist2" may contain a pattern only if "FDlist1" contains a pattern. Line-number ranges and FDname modifiers may be included. If "FDlist2" is omitted, the output lines are written on *SINK*.

If "FDlist1" contains a pattern, a set of files are copied. If "FDlist2" contains a pattern, the output files are generated by replacing each question mark in "FDlist2" with the string matched by the corresponding question mark in "FDlist1". "FDlist1" must have at least as many question marks as "FDlist2".

"FDlist2" does not have to contain a pattern if "FDlist1" contains a pattern. For example, the command

```
COPY -A? TO -B?
```

is equivalent to the command sequence

```
COPY -A1 TO -B1
COPY -A2 TO -B2
```

and the command

```
COPY -A? TO -B
```

is equivalent to the command sequence

```
COPY -A1 TO -B
COPY -A2 TO -B(*L+1)
```

assuming the existence of the files –A1 and –A2. The first command copies two files into two other files with similar names, while the second command copies two files into a single file back to back.

Program Key: *MTS.COPY

Description: The COPY command causes a series of read and write operations to be performed. Lines are read sequentially from "FDlist1" and written to "FDlist2" until an end-of-file condition is encountered on "FDlist1".

For line files, the read operation uses the line numbers of "FDlist1". For sequential files and devices, the read operation simulates line numbers, although a beginning line number and increment may be specified for a device. For line files and devices, the write operation generates line numbers starting at 1 with an increment of 1 unless a beginning line number and increment is specified on "FDlist2". For sequential files, the write operation ignores line numbers and writes the lines at the end of the file. If the @I FDname modifier is used on "FDlist2", the write operation uses the line numbers from the read operation for generating line numbers for the write operation.

For line files, lines may be written on "FDlist2" sequentially or indexed. If a line file is written sequentially, renumbering of lines occurs; however, the user can specify the beginning line number and increment for "FDlist2". For sequential files and devices, lines are written on "FDlist2" sequentially. If a line file is copied to a sequential file, the line numbers are lost. See Appendix A to the section "Files and Devices" for a further description of the use of modifiers with read and write operations.

An exact copy of a line file can be made by issuing the command

```
COPY file1(*F)@–TRIM@–IC@–ENDFILE file2@–TRIM@I
```

An exact copy of a sequential file can be made by issuing the command

```
COPY file1@–TRIM@–IC@–ENDFILE file2@–TRIM
```

The –TRIM FDname modifier disables trimming, the –IC modifier disables implicit concatenation, and the –ENDFILE disables the recognition of the $ENDFILE delimiter. The above commands are both equivalent to issuing the command

DUPLICATE file1 AS file2

The user must have READ access to “FDlist1” and WRITE access to “FDlist2”.

A quoted string can be used in place of the file being copied from. This makes it possible to insert lines with carriage controls and other formatting codes directly into the print stream to the Xerox 9700 page printer, rather than having to insert them in the file and then copy the file to the page printer. For example, the following commands force the page printer to begin every copy of “file1” on a new sheet:

```
$CONTROL *PRINT*
$COPY ":" *PRINT*
$COPY file1 *PRINT*
$COPY ":" *PRINT*
$COPY file2 *PRINT*
$RELEASE *PRINT*
```

It is also possible to use the COPY command to switch formats in a single print job. The following example prints “file1” in portrait mode, and “file2” in landscape mode:

```
$CONTROL *PRINT* PAPER=3HOLE
$COPY '$9700 PORTRAIT' *PRINT*
$COPY file1 *PRINT*
$COPY '$9700 LANDSCAPE' *PRINT*
$COPY file2 *PRINT*
$RELEASE *PRINT*
```

You can also use $COPY to send arbitrary strings such as headers to the page printer:

```
$CONTROL *PRINT*
$COPY " Output from Job 1" *PRINT*
$COPY file1 *PRINT*
$RELEASE *PRINT*
```

This will print “Output from Job 1” (without the quotes) at the top of the first page of file1. Note that the first character in the string must be a carriage-control character, in this case, a blank.

The quoted string must stand alone and cannot be part of a concatenation. Thus, these commands work properly:

```
$CONTROL *PRINT*
$COPY ":" *PRINT*
$COPY file1 *PRINT*
$RELEASE *PRINT*
```

But this is illegal:

```
$COPY ":"+file1 *PRINT*
```

Examples: `COPY DATA1 TO DATA2`

The file DATA1 (beginning with line 1) is copied to the file DATA2. If DATA2 is a line file, new line numbers are generated for DATA2 starting at 1 and incremented by 1. The line numbers from DATA1 are not carried over to DATA2.

```
COPY DATA1 DATA2@I
```

The file DATA1 (beginning with line 1) is copied to the file DATA2. The line numbers from DATA1 are retained in DATA2. If DATA1 is a sequential file, these line numbers start with 1 and are incremented by 1. DATA2 must be a line file.

```
COPY DATA1+DATA2(5,20) DATA3(10,,10)
```

The file DATA1 (beginning with line 1) and all lines between 5 and 20, inclusive, of the file DATA2 are copied to the file DATA3. The line numbers of DATA3 start at 10 and are incremented by 10.

```
COPY DATA1
```

The file DATA1 (beginning with line 1) is copied to *SINK* (default).

```
COPY DATA? *PRINT*
```

All files beginning with the string DATA are copied to *PRINT*.

```
COPY -?  TEMPFILES
```

All temporary files are copied into the file TEMPFILES.

```
COPY WABC:DATA1 DATA1
```

The file DATA1 belonging to userID WABC is copied to the file DATA1. The current userID must have read access to WABC:DATA1 (see the PERMIT command).

## CREATE

MTS Command Description

| | |
|---|---|
| Purpose: | To create either a permanent or temporary file, or a pseudodevice name. |
| Prototype: | CREATE {filename \| *pdn*} [keywords] |

When creating files, only the "filename" parameter giving the name of the file to be created is required. See the section "Files and Devices" in this volume for restrictions on the length of file names and legal characters. The other legal keyword parameters that may be given are:

SIZE={n | nP}
MAXSIZE={n | nP}

The SIZE parameter specifies the *estimated* size of the file to be created. The MAXSIZE parameter specifies the maximum size to which the file may be expanded. The size may be given in one of two forms:

| | |
|---|---|
| n | the number of 50 byte lines |
| nP | the number of 4096 byte pages |

If the SIZE parameter is omitted, the default size is 1 page. If this size is exceeded when the file is used, the system attempts to extend the file. If the MAXSIZE parameter is omitted, the default maximum size is 32767 pages. The maximum value that may be specified is 2686975 lines or 32767 pages. However, the size of the largest file that may be created within these limits depends on the total amount of file space available in the system and the user's maximum file space allotment.

The formulas for calculating the size of a file are given in Appendix C to "Files and Devices."

TYPE={LINE | SEQ}

The TYPE parameter specifies the type of file to be created where

| | |
|---|---|
| LINE | a line file |
| SEQ | a sequential file |

The default type is LINE.

When creating pseudodevice names, the TYPE parameter must be given specifying the type of pseudodevice name being created.

TYPE={IMPORT | EXPORT | DUMMY}

The TYPE parameter specifies the type of pseudodevice name to be created where

| | |
|---|---|
| IMPORT | is a pseudodevice that is used for importing data using BITNET connections. |
| EXPORT | is a pseudodevice that is used for exporting data using BITNET connections. |
| DUMMY | is a pseudodevice that always returns an end-of-file on a read operation and discards output on a write operation. |

Program Key: *MTS.CREATE

Description: The CREATE command can be used to create either a permanent or temporary file. The parameter "filename" gives the name of the file to be created.

When the command is entered, MTS checks to ensure that a file of the given name does not already exist; an error comment is produced if the file already exists. Then MTS checks the user's file space allocation to determine if there is enough space remaining to allow creation of the file. Finally, MTS attempts to acquire the space. If all three steps are successful, MTS informs the user of the successful creation of the file. The file is initially empty when created.

The SIZE parameter gives an *approximation* of the number of bytes that can actually be stored in the file. The actual capacity of the file is affected by the type of the file, the location of the file, the length of the lines stored in the file, and the order in which the lines are written (for line files).

Since temporary files are created automatically when first referenced in a command or subroutine call, their explicit creation is necessary only when nondefault specifications are required.

A file may be created from a program by calling the CREATE subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Examples:

```
CREATE A
```

The file A is created with the default size of 1 page.

```
CREATE BIGFILE SIZE=100P
```

The file BIGFILE is created with a size of 100 pages.

```
CREATE *JUNK* TYPE=DUMMY
```

The pseudodevice name *JUNK* is created. All output written to *JUNK* will be discarded.

## DEBUG

MTS Command Description

Purpose: To load a program and enter into debug command mode.

Prototype:

```
DEBUG [program] [I/Ounits] [limits] [mapoptions]
        [{EXECPKEY | PKEY}={key | OFF}] [PAR=parameters]
```

The following parameters may be given:

program

"program" specifies the file(s) or device(s) containing the program to be loaded and debugged. If omitted, debug mode is entered without loading a program.

I/Ounits

The keyword parameters "I/Ounits" are the assignments of logical I/O units to files or devices for use by the loaded program during execution. The logical I/O unit assignments are used to select appropriate I/O subroutines to be used by the loaded program for input and output of data. Where no specifications are stated, the following default assignments occur:

```
SCARDS=*SOURCE*
GUSER=*MSOURCE*
SERCOM=*MSINK*
SPRINT=*SINK*
SPUNCH=*PUNCH*        (batch mode if global card est. > 0)
```

Synonyms may be used for the following logical I/O unit names:

INPUT for SCARDS
PRINT for SPRINT
OBJECT for SPUNCH

The logical I/O units 0 through 99 have no default specifications. See the section "Files and Devices" in this volume and the subroutine descriptions for INPUT (SCARDS), PRINT (SPRINT), OBJECT (SPUNCH), SERCOM, GUSER, READ, and WRITE in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for further details on the use of these subroutines.

The logical I/O unit assignments may be initially assigned or reassigned in debug mode via the SET debug command.

FORTRAN users are reminded that MTS logical I/O units 0 through 99 are not necessarily the same as the FORTRAN logical I/O units 0 through 99.

limits

> The keyword parameters "limits" specify local limits for time, pages printed, and cards punched. For a complete description of the use of limits, see the RUN command description and the section "UserIDs, Limits, and Sigfiles" in this volume.

MAP[=mapFDname] [NOMAP] [XREF] [UXREF]

> The MAP, XREF, and UXREF parameters are used to obtain a loader map and cross-reference listings from the dynamic loader. These are not normally needed when in debug mode. See the RUN command description for further details.

{EXECPKEY | PKEY}={key | OFF}

> The EXECPKEY parameter "key" specifies an override program key to be used instead of the program key associated with "program". If OFF is specified, the override specified by the EXECPKEY option of the SET command is disabled. For further details on the use of program keys, see the section "Files and Devices" in this volume.

PAR=parameters

> The PAR field specifies an arbitrary string of characters to be passed to the loaded program on initiation of execution. This is usually a parameter list for the program and its interpretation depends on the loaded program. The PAR field must be the last parameter specified in the command. The parameter list is terminated by the end of the command line. Note that the parameter field always has a blank added after the last character and the length count is incremented by one.
>
> The parameter list may be initially assigned or reassigned in debug mode via the SET debug command.

Program Key: *SDS

Description: The DEBUG command invokes the dynamic loader to load the object in "program". For a complete description of the loading process, see the description of the RUN command in this section and the section "The Dynamic Loader" in *MTS Volume 5: System Services*, Reference R1005.

If the program is loaded successfully, control is transferred to debug command mode. In debug mode, the user may use the facilities of the symbolic debugging system (SDS) to display or modify parts of the loaded program and to initiate execution. SDS monitors the execution of the program. See *MTS Volume 13: The Symbolic Debugging System*, Reference R1013, for further details.

If no parameters are given on the DEBUG command, debug mode is entered without loading (or unloading) any program.

If “program” has a nondefault program key that is not prefixed by the current userID, the program key will be set to the default value (for this invocation of the program only).

Example:

```
DEBUG OBJPROG 5=INPUT 6=OUTPUT
```

This loads the program OBJPROG and transfers control to debug mode. Logical I/O units 5 and 6 are assigned to the files INPUT and OUTPUT, respectively.

## DESTROY

MTS Command Description

Purpose: To destroy a file or a set of files, or a pseudodevice name.

Prototype: DESTROY {filelist | *pdn*} [{OK | ALLOK | PROMPT}]

Program Key: *MTS.DESTROY

Description: The “filelist” parameter specifies the file or the set of files to be destroyed and may be a single file name, a file-name pattern, or a parenthesized list of either.

If a single file name is specified, confirmation is requested in conversational mode unless the file is a temporary file. If more than one file is specified, a single summary confirmation is requested. If the reply is OK, then all the files specified are destroyed; otherwise, none are destroyed. Confirmation is not requested in batch mode.

When only a single file name is specified, the OK option may be used to bypass the confirmation request; the OK option is ignored if more than one file name is specified. The ALLOK option may be used to bypass the confirmation request when several files have been specified. The PROMPT option causes prompting for confirmation for each individual file, including temporary files.

The response to a prompt for confirmation may be OK to destroy the file, NO to skip the file but continue with the next file in the list, or CANCEL to terminate the command.

If only “?” is specified for “filelist”, a special confirmation request

```
Do you really want to destroy ALL your files?
```

is printed. If the reply is OK, then the command continues with a summary confirmation request or with confirmation requests for each individual file.

The user must have DESTROY access to the files.

The destroyed files are deleted from the user’s file catalog. The space occupied by the files is released and the user is no longer charged for it.

If a file is currently in use by this job and is locked in some manner, it remains in use and locked. This allows the file to be locked, destroyed, recreated, and written into without any other job erroneously using it while it does not exist.

A file or a set of files may be destroyed from a program by calling the DESTROY subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

The “*pdn*” parameter specifies a pseudodevice name to be destroyed.

Examples:

```
DESTROY DATA1
```

This command destroys the file DATA1. The terminal user is prompted for confirmation.

```
DESTROY DATA2 OK
```

This command destroys the file DATA2. Confirmation is given on the command.

```
DESTROY (DATA1,DATA2,DATA3) ALLOK
```

This command destroys the files DATA1, DATA2, and DATA3. Confirmation is given on the command.

```
DESTROY DATA?  PROMPT
```

This command destroys all files beginning with the string DATA. The terminal user is prompted for confirmation for each file to be destroyed.

```
DESTROY *JUNK*
```

This command destroys the pseudodevice name *JUNK*.

## DISPLAY

MTS Command Description

Purpose: To display the contents of general registers, floating-point registers, specified virtual memory locations, the program status word, the accumulated cost of the job, and/or other system information for the user's job.

Prototype: DISPLAY [OUTPUT=FDname] [format] {location | item} ...

"location | item" is the only required parameter. As many "location | item" and "format" parameters may be given as desired.

OUTPUT=FDname

"FDname" is the file or device to which the output from the DISPLAY command is written. If "FDname" is omitted, the output is written on *SINK*. An error comment is produced if "FDname" specifies a nonexistent or unavailable file or device.

format

The format of the display may be specified by any combination of the following options:

| | |
|---|---|
| HEX | Hexadecimal conversion |
| NOHEX | Hexadecimal conversion off |
| MNEMONIC | Mnemonic and hex conversion |
| NOMNEMONIC | Mnemonic and hex conversion off |
| EBCDIC | EBCDIC conversion |
| NOEBCDIC | EBCDIC conversion off |
| SINGLESPACE | Single-space output |
| DOUBLESPACE | Double-space output |
| ORL=SHORT | Short output record (70 char) |
| ORL=LONG | Long output record (130 char) |
| ORL=n | Output record is "n" characters |

If not specified, the following default option settings apply:

NOHEX
NOMNEMONIC
NOEBCDIC
SINGLESPACE
ORL=LONG for printers
ORL=SHORT for terminals

If the NOHEX, NOMNEMONIC, and NOEBCDIC are all specified simultaneously (explicitly or by default), the output is displayed with hexadecimal conversion.

A "format" parameter affects only those "location" parameters that are blocks of virtual memory and which follow it. A "format" parameter does not affect single memory locations or other items.

location

"location" specifies what is to be displayed from the loaded program. If the loaded program is a "run-only" program, these options are not legal. This may be any of the following:

GRx

GRx specifies the general register "x", where "x" is a decimal integer from 0 to 15 or a hexadecimal integer from 0 to 9, A to F, or "S", if all general registers are to be displayed.

FRx

FRx specifies the floating-point register "x", where "x" is one of the integers 0,2,4, or 6, or "S", if all floating-point registers are to be displayed.

[RF={hhhhhh | GRx}] xxxxxx[**...**xxxxxx]

This specifies a virtual memory location or range of locations given by an *optional* local relocation factor and a displacement or range of displacements. "hhhhhh" is the hexadecimal value of a local relocation factor; GRx indicates the general register whose contents are to be used as a local relocation factor. "xxxxxx" is the hexadecimal value of a displacement. A range of displacements can be given by "xxxxxx**...**xxxxxx". The displacement is added to the current value of the relocation factor to provide an absolute 24-bit virtual memory address. If a local relocation factor is not specified, the global relocation factor is used. The global relocation factor is initially zero, but may be changed by the RF option of the SET command or by calling the CUINFO subroutine. When a relocation factor is specified in the command, it remains in effect for the remainder of the command unless subsequently overridden by a second local relocation factor specification.

PSW

PSW specifies the program status word at the time the last loaded program terminated.

item

"item" specifies informational items to be displayed that are not concerned with a loaded program. This may be any of the following:

ADDRESS

ADDRESS displays the current setting of the SET ADDRESS option.

AUTOHOLD

AUTOHOLD displays the current setting of the SET AUTOHOLD option.

CARDS

CARDS displays the current output card limit (SET CARDS option).

CLASS

CLASS displays the current job file class (SET CLASS option).

COMMENT

COMMENT displays the current head sheet comment text (SET COMMENT option).

CONTCHAR

CONTCHAR displays the current line-continuation character (SET CONTCHAR option).

COPIES

COPIES displays the current setting of the SET COPIES option.

COST

COST displays the accumulated cost of the current job. This includes all charges up to the current time *except* charges for permanent file storage, mounted tapes, and open UMnet/Michnet Computer Network connections.

CROUTE

CROUTE displays the current setting of the SET CROUTE option.

DATE

DATE displays the current date.

DEBUG

DEBUG displays the current setting of the SET DEBUG option.

<u>DELIV</u>ERY

DELIVERY displays the current delivery setting (SET DELIVERY option).

DESTINATION

DESTINATION displays the current BITNET destination (SET DESTINATION option).

{EBM | ETM}

EBM and ETM display the current settings of the SET EBM and SET ETM options.

ERRORPROMPT

ERRORPROMPT displays the current setting of the SET ERRORPROMPT option.

EXECPFX

EXECPFX displays the current execution prefix character (SET EXECPFX option).

FILE

FILE displays the current job file name (SET FILE option).

FORMAT

FORMAT displays the current page-printer output format (SET FORMAT option).

GUINFO(name)

GUINFO(name) displays the value of the GUINFO item specified by "name". Alternatively, any GUINFO item may be displayed directly by giving the command

```
DISPLAY name
```

provided that "name" does not conflict with any existing option for the DISPLAY command. The list of GUINFO items is given in the description of the GUINFO/CUINFO subroutine in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

HELPMODE

HELPMODE displays the current helpmode setting; 0=line, 1=default, 2=screen (SET HELPMODE option).

HOSTNAME

HOSTNAME displays the current host name (normally UM).

INITFILE(command)

INITFILE displays the name of the current initialization file for an MTS command (SET INITFILE option).

INSTALLATIONNAME

INSTALLATIONNAME displays the current installation name ("MTS Ann Arbor" for the University of Michigan).

JOBNAME

JOBNAME displays the current job name (SET JOBNAME option).

{LASTRELOAD | LRL}

LASTRELOAD displays the time of the last reload of the system.

LIBSRCH

LIBSRCH displays the current setting of the SET LIBSRCH option.

LOGSTATUS

LOGSTATUS displays information about the files and devices currently being logged by the LOG command.

MACHINE

MACHINE displays the type, model, and serial number of the computer that the system is currently running on.

MAILCALL

MAILCALL displays the current setting of the SET MAILCALL option.

MAP

MAP displays the loader map. The format and contents of the loader map are controlled by the SET PRMAP, PDMAP, and MAPDOTS options; the SET SYMTAB option must be ON.

MAPDOTS

MAPDOTS displays the current setting of the SET MAPDOTS option.

MARGIN

MARGIN displays the current page-printer output margin (SET MARGIN option).

NAME

NAME displays the current name (SET NAME option).

NAMELIB

NAMELIB displays the current name-library (SET NAMELIB option).

{NEWFILEACCESS | NFA}

NEWFILEACCESS displays the setting of the default file access used when new files are created (SET NEWFILEACCESS option).

NUMBER

NUMBER displays the current output page-number setting (SET NUMBER option).

OVERLAY

OVERLAY displays the current page-printer output overlay (SET OVERLAY option).

PAGES

PAGES displays the current output page limit (SET PAGES option).

PAPER

PAPER displays the current setting of the SET PAPER option.

PASSWORD

PASSWORD displays pertinent information about the user's password.

<u>PDN</u>S

PDNS displays the user-mounted pseudodevice names that are active. These are the pseudodevice names for magnetic tapes and UMnet/Michnet Computer Network connections. The information displayed includes the pseudodevice name, tape name, device type, and device name.

PRINT

PRINT displays the current setting of the SET PRINT option.

PRINTER

PRINTER displays the current setting of the SET PRINTER option.

PROJECT

PROJECT displays the current projectID.

PROJECTPWCHANGE

PROJECTPWCHANGE displays the current setting of the SET PROJECTPWCHANGE option.

PROUTE

PROUTE displays the current setting of the SET PROUTE option.

RATES

RATES displays the current rates in effect for the session.

RCPRINT

RCPRINT displays the current setting of the SET RCPRINT option.

RERUN

RERUN displays the text of the last RUN or RERUN command. This is the text that will be used if another RERUN command is issued.

RF

RF displays the current relocation factor (SET RF command).

RUNS

RUNS displays the command text of all RUN commands issued during the session.

SEE_DISPATCHES

SEE_DISPATCHES displays the current setting of the SET DISPATCHES option.

SENSE(device-name)

SENSE displays the sense information associated with the specified device name.

SEPCOPY

SEPCOPY displays the current setting of the SET SEPCOPY option.

SHIFT

SHIFT displays the current page-printer output shift value (SET SHIFT option).

SHOWNAME

SHOWNAME displays the current setting on the SET SHOWNAME option.

SIGFILE

SIGFILE displays the current and new (if any) signon and project signon files (sigfiles) (SET SIGFILE option).

SINK

SINK displays the current *SINK* and the previous sink file or device (if any).

SOURCE

SOURCE displays the current *SOURCE* and the previous source file or device (if any).

SPELLCOR

SPELLCOR displays the current setting of the SET SPELLCOR option.

SYMTAB

SYMTAB displays the current setting of the SET SYMTAB option.

{SYSTEM | MODEL}

SYSTEM or MODEL displays information about the currently executing version of MTS.

TAILSHEET

TAILSHEET displays an itemized list of the accumulated costs for the current session. This includes all charges up to the current time *except* charges for permanent file storage, mounted tapes, and open UMnet/Michnet Computer Network connections.

TASKNUMBER

TASKNUMBER displays the current task number.

TDR

TDR displays the current setting of the SET TDR option.

TERSE

TERSE displays the current setting of the SET TERSE option.

{TIME | TIMESPELLEDOUT}

TIME and TIMESPELLEDOUT display the current time.

TIMEDATE

TIMEDATE displays the current time and date.

TWOSIDED

TWOSIDED displays the current setting of the SET TWOSIDED option.

UPTIME

UPTIME displays the amount of time elapsed since the last system reload.

USMSG

USMSG displays the current setting of the SET USMSG option.

USERID

USERID displays the current userID.

UXREF

UXREF displays the current setting of the SET UXREF option.

VERSION(command)

VERSION displays the current version in use for the specified MTS command (SET VERSION option).

VMSIZE

VMSIZE displays the current size of the user's virtual memory in a decimal number of pages.

XREF

XREF displays the current setting of the SET XREF option.

*BATCH*

*BATCH* displays the active *BATCH* job (if there is one) giving the receipt number and the number of lines read for the job.

*PRINT*

*PRINT* displays the active *PRINT* job (if there is one) giving the receipt number, the number of lines, pages, images, and sheets, the number of copies (if greater than 1), the route, the printer character set (if specified), the printer type, the output form, and the delivery station (if specified).

*PUNCH*

*PUNCH* displays the active *PUNCH* job (if there is one) giving the receipt number, the current number of cards, and the punch route.

*...*

*...* displays all active *PRINT*, *PUNCH*, and *BATCH* jobs giving the items listed above for each type of job.

*pdn*

"*pdn*" displays information about pseudodevices mounted by the MOUNT command or created by the CREATE command. For magnetic tapes, this includes information such as the tape name, volume name, file number, block number, record number, format, density, file name, and the status of various user options.

Program Key: *MTS.DISPLAY

Description: The DISPLAY command displays general registers, floating-point registers, specified virtual memory locations, the program status word, the accumulated cost of the current job, and other system information for the user's job.

The general registers, floating-point registers, and the PSW are displayed in labeled hexadecimal format.

Blocks of virtual memory are displayed in hexadecimal, mnemonic and hexadecimal, and/or EBCDIC format.

Examples:

```
DISPLAY GR3 FRS EBCDIC 818E08...818FA6
```

This displays GR3 and all the floating-point registers on *SINK* in single-spaced hexadecimal format, and displays virtual memory locations 818E08 through 818FA6 (assuming a global relocation factor of zero) in single-spaced EBCDIC format.

```
DISPLAY ON DISPLAYFILE ORL=LONG GRS PSW VMSIZE
```

This displays on the file DISPLAYFILE all the general registers, the program status word, and the size of the user's virtual memory in long record hexadecimal format.

```
DISPLAY RF=818000 200...480 MNEMONIC EBCDIC 800...A80
```

This displays virtual memory locations 818200 through 818480 on *SINK* in hexadecimal format (the default), and displays locations 818800 through 818A80 in hexadecimal, mnemonic, and EBCDIC format.

```
DISPLAY COST
```

This displays the accumulated cost of the current job.

```
DISPLAY *PRINT*
```

This displays information about the active *PRINT* job.

## DUMP

MTS Command Description

Purpose: To display the contents of general registers, floating-point registers, the program status word, and the virtual memory locations associated with the user's current loaded program.

Prototype: DUMP [OUTPUT=FDname] [format] ...

As many "format" parameters may be given as desired. The only restriction on the order of parameters in the command line is that "ON FDname" must appear first if it appears at all.

OUTPUT=FDname

"FDname" is the file or device to which the output from the DUMP command is written. If "FDname" is omitted, the output is written on *SINK*. An error comment is produced if "FDname" specifies a nonexistent or unavailable file or device.

format

The format of the display may be specified by any combination of the following options:

| | |
|---|---|
| HEX | Hexadecimal conversion |
| NOHEX | Hexadecimal conversion off |
| MNEMONIC | Mnemonic and hex conversion |
| NOMNEMONIC | Mnemonic and hex conversion off |
| EBCDIC | EBCDIC conversion |
| NOEBCDIC | EBCDIC conversion off |
| SINGLESPACE | Single-space output |
| DOUBLESPACE | Double-space output |
| ORL=SHORT | Short output record (70 char) |
| ORL=LONG | Long output record (130 char) |
| LIBRARY | Include library space in dump |
| NOLIBRARY | Exclude library space from dump |

If not specified, the following default option settings apply:

NOHEX
NOMNEMONIC
NOEBCDIC
SINGLESPACE
ORL=LONG for printers

ORL=SHORT for terminals
LIBRARY

If the NOHEX, NOMNEMONIC, and NOEBCDIC are all specified simultaneously (explicitly or by default), the output is dumped with hexadecimal conversion. If ORL=L is specified or default, then both hexadecimal and EBCD conversion is given side by side.

Program Key: *MTS.DUMP

Description: The DUMP command displays the general registers, floating-point registers, the program status word, and the virtual memory locations associated with the user's current loaded program.

The general registers are displayed in hexadecimal, fixed-point decimal, and symbolic address format. The floating-point registers are displayed in hexadecimal and floating-point decimal format.

Blocks of virtual memory are displayed in hexadecimal, mnemonic and hexadecimal, and/or EBCDIC format.

A dump may be produced from a program by calling the STDDMP or SDUMP subroutines (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

This command is invalid if the currently loaded program is a "run-only" program.

Examples:

```
DUMP
```

The general registers, floating-point registers, program status word, and virtual memory locations are displayed in single-spaced hexadecimal format.

```
DUMP HEX EBCDIC DOUBLESPACE SP2
```

The general registers, floating-point registers, program status word, and virtual memory locations are displayed in double-spaced hexadecimal and EBCDIC format.

## DUPLICATE

MTS Command Description

Purpose: To duplicate a file or a set of files.

Prototype: DUPLICATE oldname [AS] newname [keywords] [{OK | ALLOK | PROMPT}]

The legal keywords are:

{OPTIMIZE | NOOPTIMIZE}

The OPTIMIZE keyword specifies that the file is to be optimized during the duplication process; that is, all unused space in each page of the file is eliminated. NOOPTIMIZE duplicates the file without optimization. The default is OPTIMIZE.

{DATA | NODATA}

The DATA keyword specifies that both the data lines in the file and the file attributes are copied to the duplicate file. The NODATA keyword specifies that only the file attributes are copied; the duplicate file remains empty. The default is DATA.

EMPTYOK

The EMPTYOK keyword specifies that confirmation is not required to copy an empty file into a preexisting nonempty file.

Program Key: *MTS.DUPLICA

Description: The “oldname” parameter specifies the file or the set of files to be duplicated and may be a single file name or a file-name pattern. The “newname” parameter specifies the new names for the files being duplicated.

If “oldname” is a pattern, a set of files are duplicated. If “newname” is a pattern, the new names are generated by replacing each question mark in “newname” with the string matched by the corresponding question mark in “oldname”. “oldname” must have at least as many question marks as “newname”. “newname” must be a pattern if “oldname” matches more than one file. For example,

```
DUPLICATE ?.S ?.OLD
```

is equivalent to

```
DUPLICATE A.S A.OLD
DUPLICATE B.S B.OLD
```

if ?.S matches the files A.S and B.S.

If the files specified by “newname” do not exist, they are created at the minimum size (MINSIZE) for the corresponding “oldname”. The file type (LINE or SEQ) is the same. The data from “oldname” is copied exactly to “newname”. The expansion-factor, file-save, and program-key attributes (MAXSIZE, NOSAVE, and PKEY) are retained. If the user has PERMIT access to “oldname”, then “newname” is permitted like “oldname”; otherwise, the access to “newname” is DEFAULT.

If the files specified by “newname” already exist, “newname” is emptied and the data is copied from “oldname” to “newname”. The other attributes and access information are not changed.

If a single file name is specified as “oldname”, confirmation is not requested in conversational mode unless “newname” already exists. If more than one file is specified, a single summary confirmation is requested. If the reply is OK, then all the files specified are duplicated; otherwise, none are duplicated. Confirmation is not requested in batch mode.

When only a single file name is specified, the OK option may be used to bypass the confirmation request; the OK option is ignored if more than one file name is specified. The ALLOK option may be used to bypass the confirmation request when several files have been specified. The PROMPT option causes prompting for confirmation for each individual file.

The response to a prompt for confirmation may be OK to duplicate the file, NO to skip the file but continue with the next file in the list, or CANCEL to terminate the command.

The user must have READ access to “oldname”. “newname” must specify a file belonging to the current userID, if it does not exist. The user must have EMPTY and WRITE access to “newname”, if it already does exist.

Permanent files may be duplicated as temporary files, and temporary files may be duplicated as permanent files if the user’s file space allocation allows it.

Example:

```
DUPLICATE DATA1 AS NEWDATA1
```

The file DATA1 is duplicated as NEWDATA1. The terminal user is prompted for confirmation.

```
DUPLICATE DATA2 NEWDATA2 OK
```

The file DATA2 is duplicated as NEWDATA2. The user is not prompted for confirmation.

```
DUPLICATE DATA?  NEWDATA?  ALLOK
```

All files beginning with the string DATA are duplicated as corresponding files beginning with the string NEWDATA. The user is not prompted for confirmation.

```
DUPLICATE WABC:DATA1 DATA1
```

The file DATA1 belonging to userID WABC is duplicated to the file DATA1. The current userID must have read access to WABC:DATA1 (see the PERMIT command).

## EDIT

MTS Command Description

Purpose: To invoke the MTS File Editor for making changes to a file.

Prototype: EDIT [filename] [:edit-command]

The legal parameters are:

filename

“filename” is the name of the file to be edited.

edit-command

“edit-command” is an optional edit command. The “:” is optional if “filename” is specified. The edit command is executed as a single command and an immediate return is made to the caller.

Program Key: *EDIT

Description: “filename” is the name of the file to be edited. This may be either a line file or a sequential file. If “filename” is omitted, the currently active file becomes the edit file; if there is no currently active file, edit mode is entered without an edit file.

For further details about using the File Editor, see *MTS Volume 18: The MTS File Editor*, Reference R1018.

Example:

```
EDIT DATAFILE
```

The File Editor is invoked to edit the line file DATAFILE.

```
EDIT :CHANGE 10 'A'B'
```

This command changes the first occurrence of the character A in line 10 of the edit file to the character B and then returns to MTS command mode.

## EMPTY

MTS Command Description

| | |
|---|---|
| Purpose: | To empty a file or a set of files. |
| Prototype: | EMPTY filelist [{OK ǀ ALLOK ǀ PROMPT}] |
| Program Key: | *MTS.EMPTY |

Description: The "filelist" parameter specifies the file or the set of files to be emptied and may be a single file name, a file-name pattern, or a parenthesized list of either.

If a single file name is specified, confirmation is requested in conversational mode unless the file is a temporary file. If more than one file is specified, a single summary confirmation is requested. If the reply is OK, then all the files specified are emptied; otherwise, none are emptied. Confirmation is not requested in batch mode.

When only a single file name is specified, the OK option may be used to bypass the confirmation request; the OK option is ignored if more than one file name is specified. The ALLOK option may be used to bypass the confirmation request when several files have been specified. The PROMPT option causes prompting for confirmation for each individual file, including temporary files.

The response to a prompt for confirmation may be OK to empty the file, NO to skip the file but continue with the next file in the list, or CANCEL to terminate the command.

If only "?" is specified for "filelist", a special confirmation request

```
Do you really want to empty ALL your files?
```

is printed. If the reply is OK, then the command continues with a summary confirmation request or with confirmation requests for each individual file.

The user must have EMPTY (or WRITE-CHANGE) access to the files.

The current contents of the files are discarded. The space occupied by the files is not released and the user is still charged for it.

A file or a set of files may be emptied from a program by calling the EMPTY or EMPTYF subroutines (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

When a file is emptied, the *entire* contents are discarded. The EMPTY command cannot be used to empty only a portion of a file. The DELETE edit command may be used to delete a portion of a file (see *MTS Volume 18: The MTS File Editor*, Reference R1018).

Examples:

```
EMPTY DATA1
```

This command empties the file DATA1. The terminal user is prompted for confirmation.

```
EMPTY DATA2 OK
```

This command empties the file DATA2. Confirmation is given on the command.

```
EMPTY (DATA1,DATA2,DATA3) ALLOK
```

This command empties the files DATA1, DATA2, and DATA3. Confirmation is given on the command.

```
EMPTY DATA?  PROMPT
```

This command empties all files beginning with the string DATA. The terminal user is prompted for confirmation for each file to be emptied.

## FILEMENU

MTS Command Description

Purpose: To obtain file information in full-screen format from which MTS commands can be issued.

Prototype: FILEMENU [name] [information]
FMENU [name] [information]

name

The "name" parameter is specified as

{filename | ? | –? | userid:?}

"filename" is the name of the file for which information is to be displayed. "?" specifies that information is to be displayed for all permanent files belonging to the userID in use. "–?" requests that information is to be obtained for all of the temporary files belonging to the userID in use. "userid:?" specifies that information is to be obtained for all files for the specified userID for which the user has the appropriate access. The "name" parameter may also be given as a parenthesized list of items, i.e.,

(name1,name2,name3,...)

The default is "?". The display for "?", "–?", and "userid:?" is sorted by file name, initially.

The presence of the "?" character in "filename" indicates that this is a partially specified file name. A single "?" will match zero or more arbitrary characters in the file name. "n" consecutive "?" characters will match "n–1" arbitrary characters in the file name. The "?" character cannot be used in the userID portion of a shared file name.

information

The "information" parameters are specified as

item1 item2 item3 ...

The items that may be displayed is the same as for the FILESTATUS command (see the description of the FILESTATUS command for details). The order in which the items are displayed is controlled by the order in which they are specified, with the restriction that the MYACCESS and ACCESS items are always displayed last.

By default, only the LASTCHG, RPM, and MYACCESS information is displayed; this default can be reset by the user by using the /DISPLAY command (see below).

FILEMENU retains all file information internally when a file is displayed for later use in sorting (by the /SORT command) and filtering (by the /REMOVE command). However, information classed as FILE information can be quite expensive to obtain, especially when a large number of files are being listed. Therefore, unless file information is requested on the initial FILEMENU command (or on a subsequent /ENTER or /ADD command), this information is not acquired; each numeric item is assigned a value of −1, and the EMPTY item is assigned a value "???", rather than YES or NO.

Selective Filtering

Any information item may be specified as a comparison that must be satisfied before a file may be displayed. For a complete description, see the FILESTATUS command. There is one extension: FILEMENU accepts a filter based upon the ACCESS keyword, so that it is possible, for example, to specify a filter like "ACCESS=UNLIM" or "ACCESS>READ". (MTS uses a numeric code for each type of file access; when a filter such as ACCESS>READ or ACCESS<UNLIM is used, the numeric access codes are compared. See the description of the PERMIT subroutine in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for a description of the numeric-access codes.)

Program Key: *FILEMENU

Description: FILEMENU is a command language subsystem for use on full-screen terminals which allows MTS commands to be issued from a list of files that is displayed on the screen. The user can window over this file list, sort it into order by any file item, and add or delete files from the list. Several independent lists of files may be displayed simultaneously.

It should be noted that this command is *only* supported on full-screen terminals, that is, terminals with a display screen that can be controlled by MTS such as the IBM 3278 Display Terminal, the Ontel Terminal, and several microcomputers.

Screen Format:

FILEMENU divides each line of the screen into three parts: the NAME area, on the left, where the name and owner of a file are displayed; the INPUT area, in the middle, from which commands may be issued, and the DATA area, on the right of the screen, where file information is displayed. If more information is requested than will fit in the data area, the data area may extend to more than one screen line. If the input area is not wide enough to accommodate a particular command, it may be extended, first to encompass the data area, then to succeeding lines, so that a command of up to 255 characters (the maximum accepted by MTS) may be issued.

Issuing Commands:

FILEMENU recognizes two type of commands: MTS commands and FILEMENU commands. Both may be issued from the input areas of the screen or by program-function (PF) keys. FILEMENU commands, which are distinguished by an initial slash (/), request some action from FILEMENU; for example, windowing over the file list, sorting the list into order, or adding or deleting files. MTS commands are passed to MTS for execution. If an MTS command line begins with a slash (i.e., if the name of a command macro begins with a slash), the slash should be doubled.

Substitution:

When an MTS command is issued, the name of the file that is displayed upon the line where the command was issued is appended to the command. Alternatively, if the command line contains question marks, the file name is substituted for each question mark in the line. If a question mark is to be passed unchanged to MTS, it should be doubled. For example, suppose a line displays the file name WABC:TEST. Entering "EDIT" on that line would issue the MTS command "EDIT WABC:TEST"; entering "COPY ? TO *PRINT*" would issue the command "COPY WABC:TEST TO *PRINT*"; and entering "PERMIT ? R P=W??" would issue the command "PERMIT WABC:TEST R P=W?".

Duplicating Commands:

If an equals sign (=) or quotation mark (") is entered as a command, then the last MTS command, before substitution, is executed for that file. The duplicated command line, rather than the equals sign, is displayed for the file after the command is executed.

Errors:

When a FILEMENU command is executed successfully, it is removed from the screen. When an MTS command is issued, the line from which it was entered is marked by a one-character indicator before the start of the name area. An asterisk (*) indicates an MTS command that executed successfully and a not sign (~ or ¬) denotes a command not recognized as a valid MTS command or MTS macro command. If a command line begins with a slash but does not contain a valid FILEMENU command, then the line will be marked by a question mark (?). In addition, error messages are displayed at the bottom of the part of the screen used for each file list.

Multiple File Lists:

A single file list is displayed when FILEMENU is first invoked. Additional lists may be added to the screen by using the /ENTER command. Each list is completely independent of all of the others; sorting or windowing over any list will not affect other lists displayed.

Issuing Program-Function Commands:

Any MTS or FILEMENU command may be assigned to a program-function (PF) key. When a PF key is pressed, the command for that key is executed for the file list or file to which the cursor points (depending upon the type of command). If the cursor line does not contain a file, the file name substituted into an MTS command is left blank. If a FILEMENU command which affects a file is assigned to a PF key and the cursor is not on a file line, the command is ignored.

Commands: The following commands are recognized and processed by FILEMENU. It should be noted that if a command is entered from the screen, it will not be executed until an /EXECUTE command is issued by a PF key.

/ADD [name] [information]

This adds files to the file list upon which the command is issued. The files are added after the file line where the command is issued. The "name" and "information" parameters are identical to those permitted on the FILEMENU command.

/DISPLAY [information]

This resets the specification of the file information items to be displayed for the list where the command is issued. The "information" parameter consists of one or more file item names; if it is omitted, the default display is restored (i.e., LASTCHG, RPM, and ACCESS).

/EDIT [command]

This calls the MTS File Editor to edit the file on the line from which the command is issued. If "command" is specified, the command is passed to the File Editor for processing; if "command" is omitted, the user is placed in edit command mode. FILEMENU remembers the user's position in each file /EDITed; when a file is /EDITed for a second or subsequent time, the EDIT display begins at the final position during the previous /EDIT session rather than at the beginning of the file.

/ENTER [name] [information]

This adds a new list of files to the screen. The "name" and "information" parameters are identical to those permitted on the FILEMENU and /ADD commands. If there is not enough space on the screen to add a new list, an error message is displayed. The new list is added below the list where the command is issued.

/EXECUTE [SINGLE]

This executes commands entered in the input areas of one or more files. If the SINGLE option is specified, only the command for the file where the cursor is located is executed; otherwise, commands are executed for each file in the list, starting at the top of the list and proceeding down until the end of the list is reached or an error is detected. This command is only valid if issued from a PF key; it is ignored otherwise.

/EXTEND

This extends the file line from where it was issued, first to include the data area, then to succeeding lines on the screen. The maximum number of lines to which a line may expand will vary with screen size and with the position of the line being expanded on the screen, but will never be greater than the space required to permit a 255-character command to be issued. This command is meaningful only if assigned to a PF key.

/FORGET

This removes the file list from where the command was issued from the screen. If there is only one list on the screen, an error message is displayed.

/GET

This windows a file list forward so that the first displayed file on the list is the file on the line from where the command was issued.

/GOTO {n | name}

This moves a file list so that a specific file is the first displayed. If "n" is specified, the nth file is displayed; if "name", either a filename or a file-name pattern, is specified, the first file after the file on which the command is issued that matches the pattern is displayed.

/HELP [item]

This invokes the full-screen help facility. "item" may be any FILEMENU command; if specified, information about that command will be displayed.

/INSERT text

This inserts "text" into the command line immediately before the cursor location. This command is valid only if assigned to a PF key. If it is ignored if it is issued directly from the screen or if the cursor is not on a line containing a file.

/MCMD command

This passes "command" to MTS for execution *without* doing any substitution. This is useful for issuing commands that do not take filename parameters or for issuing commands for files not on the current list. This command is identical to the /$ command.

/MTS [command]

This returns control to MTS. Reissuing the FILEMENU command will restart FILEMENU at the point where the MTS command was issued. "command" may be any MTS command to be executed before the return to MTS command mode is made.

/PF {n [command] | ?}

If "/PF n command" is specified, the given command is assigned to PF key "n". If the command line is omitted, the PF key becomes a no-op key. If a question mark is specified instead of "n command", the current settings of the PF keys are displayed. The number of PF keys available depends on the type of terminal used; "n" may be any number, even though it may not be possible to actually use some PF keys from a given terminal.

"Normal" PF keys are numbered from 1 to the maximum available on the terminal: PF1 is 1, PF2 is 2, and so on. In addition, for the IBM 3278, the ENTER key is treated as PF0, the PA1 key (ATTN on Ontels) is PF-1, the PA2 key (EOF on Ontels) is PF-2, and the SYS REQ key is PF-3.

/REDISPLAY

This displays the last MTS command issued for the file list.

/REMOVE [name] [filters]

This removes files from the file list where the command is issued. The "name" and "filters" parameters are identical to those specified for the FILEMENU, /ADD, and /ENTER commands. If both are omitted, only the file upon the line where the command was issued is removed; otherwise, all files that match the name or filter patterns are removed. When a file is removed, it is replaced by a blank line; the blank lines are removed when the next /SORT command is issued.

/RETURN

This terminates FILEMENU and returns control to the caller (normally MTS command mode).

/SET

This sets various options. The following options may be set.

DISPLAY={item | (item,item,...)}

This specifies the file information items displayed by default. The DISPLAY setting for a file list determines the items displayed for all lists created by /ENTER commands issued from the original list; the DISPLAY option is overridden by an explicit "info" parameter on the /ENTER command.

ECHO={ON | OFF}

This specifies whether MTS commands are placed in the conversation buffer as they are executed.

HEADING={ON | OFF}

This specifies whether a column heading for the file list should be displayed.

VISUAL={ON | OFF}

This specifies whether /EDIT automatically enters visual edit mode.

The /SET options are specific to each file list; that is, they may have different settings for each list on the screen. When a new list is created, its /SET values default to those of the list from which it was created. For the initial list created by the FILEMENU command, the ECHO, HEADING, and VISUAL options default off and the DISPLAY option defaults to (LASTCHG,RPM,MYACCESS), unless /SET commands are included in an initfile.

/SORT item [{ASCENDING | DESCENDING}]

This sorts a file list into an order based upon the specified file item. The order is specified by the ASCENDING or DESCENDING parameter; the default is ASCENDING. If two files have identical "item" fields, a secondary comparison by file name is performed.

/STOP

This terminates FILEMENU and returns control to the caller (normally MTS command mode).

/WB [n] [SCREENS]

This moves the file display backward. If simply "/WB" is specified, the display moves to the top of the file list. If "/WB n" is specified, the display moves backward "n" lines. If "/WB n SCREENS" is specified, the display moves back "n" screens.

/WF [n] [SCREENS]

This moves the file display forward. If simply "/WF" is specified, the display moves to the end of the file list. If "/WF n" is specified, the display moves forward "n" lines. If "/WB n SCREENS" is specified, the display moves forward "n" screens.

/$command

This passes “command” to MTS for execution *without* doing any substitution. This command is identical to the /MCMD command.

Using an Initialization File:

An initialization file (init file) may be used to redefine the PF keys with the MTS command

```
SET INITFILE(FILEMENU)=FDname
```

Only the /MCMD, /PF, /SET, and /$ commands are valid in an init file; in addition, lines beginning with “/*” are ignored, so that comment lines may be included. Any other lines will be flagged as errors.

Default PF Assignments:

The following PF-key assignments are designed to make FILEMENU PF keys as identical as possible to the default MTS File Editor and IBM 3278 PF-key assignments:

| | | |
|---|---|---|
| 1/13: /WB 1 S | 2/14: /GET | 3/15: /WB |
| 4/16: /WF 1 S | 5/17: /EDIT | 6/18: /WF |
| 7/19: /WB 1 | 8/20: /EXECUTE | 9/21: /EXEC SIN |
| 10/22: /WF 1 | 11/23: /EXTEND | 12/24: /STOP |

The PA1 and PA2 keys (on the IBM 3278), ATTN and EOF keys (on the Ontel), and Control-C and Control-E keys (on MCP WINDOW terminals) are set to /MTS. The IBM 3278 ENTER (PF0) and SYS REQ (PF-3) keys are left unassigned. The IBM 3278 CLEAR key removes all screen changes since the last PF key was pressed; it cannot be redefined by the user.

## FILESTATUS

MTS Command Description

Purpose: To obtain file information, access information, and/or catalog information for a file, or information about pseudodevice names.

Prototype: FILESTATUS [name] [format] [information]

The parameters to this command may be given in two formats: "keyword=value" or "value".

name

The "name" parameter is specified as

NAME={filelist | ? | –? | userid:?}

or

{filelist | ? | –? | userid:?}

"filelist" may be a single file name or pseudodevice name, a file-name or pseudodevice name pattern, or a parenthesized list of either, or "?". "?" specifies that information is to be obtained for all the permanent files belonging to the current userID. "–?" specifies that information is to be obtained for all of the temporary files belonging to the current job. "userid:?" specifies that information is to be obtained for all files of the specified userID for which the user has the appropriate access. The default is "?". The output for "?", "–?", and "userid:?" is sorted alphabetically by file name.

If no NAME keyword parameter is specified on the command, the first nonkeyword parameter is treated as the "name" parameter, unless it is a legal parameter to this command and the user does not have a file by that name. Thus,

FILESTATUS TYPE

specifies that the type of all the user's files is to be displayed, unless the user has a file of the name TYPE, in which case catalog information (the default) for that file is displayed. If the user has a file by the name of TYPE and desires to display the catalog information for all files, he must specify

FILESTATUS ? TYPE

If no name parameter is given, "?" is assumed as the default "name" parameter.

A list of current pseudodevice names may be obtained by specifying "*?*".

format

The "format" parameter is specified as

{COLUMNS | KEYWORD | LABELED | PACKED}

where the format parameters produce output in the following forms:

(1) COLUMNS — fixed-width columns with headings.

(2) KEYWORD or LABELED — free-form output, presented as "label=value" (e.g., TYPE=LINE). Items are separated by commas and blanks.

(3) PACKED — same as KEYWORD, except the "label=" portion is deleted (e.g., LINE).

If no format parameter is specified, the default is KEYWORD for output to a terminal, and COLUMNS for output to any other file or device. Space restrictions dictate that if both the COLUMNS and VERBOSE parameters are specified, the format is changed to PACKED. Normally, the information for each file is printed on one line. However, if only the name of the file is requested (the default case), then several file names are printed on each line for noncolumnar output. Users should note that this differs from PACKED format.

Headings may be specified for columnar output. This parameter is ignored for keyword, labeled, or packed output formats. The heading parameter is given as

{HEADING | NOHEADING}

The default is to produce headings for columnar output.

For output for multiple files in any format, the separation between file entries can be specified by

SPACING={1 | 2 | 3}

where 1 means single-spacing, 2 means double-spacing, and 3 means triple-spacing; the default is 1. This is the spacing of the first line of information for a file entry; if the file entry requires more than one line of output, succeeding lines are always single-spaced. The succeeding lines may be indented by specifying

INDENT=n

where "n" is the number of columns to indent. "n" must be in the range of 0 to 20; the default is 1.

The output file or device may be specified by the parameter

OUTPUT=FDname

The default is *SINK*.

The access information may be abbreviated by the parameter

{TERSE | VERBOSE}

If the output format is COLUMNS (the default for nonterminal output), the default form for access printing is TERSE, since the access information will not fit into any practical fixed-width column size. However, if the output format is *not* COLUMNS (the default for terminal output), the default form for access printing will be the same as the MTS TERSE option (set by the SET command). Since the default setting for this is TERSE=OFF, the default access information output format in this case will be VERBOSE.

Explicitly specifying TERSE or VERBOSE will force the output format as specified.

information

The "information" parameters are specified as

item1[,item2,...]

The order in which the items are presented is controlled by the order in which they are specified.

The information printed by the FILESTATUS command is available to a program via the GFINFO subroutine (see *MTS Volume 3*).

Use of the command does not affect the use count, last change date, last reference date, or any other items associated with the file.

The information items are divided into five groups: catalog information, access information, file information, name, and summary information. Each group has a different cost associated with obtaining the information; a different permit access is required for each group. (The owner always has permit access, so all the information is accessible.) Below are listed the separate items contained in each group.

Catalog information

The relative cost of obtaining this information is inexpensive. Any access is sufficient.

| | |
|---|---|
| OWNER | Owner userID. |
| VOLUME | Name of direct-access volume on which file is located. |

| | |
|---|---|
| USECNT | Number of references since file was created. |
| CREATE | Date when file was created. |
| LASTREF | Date when file was last referenced. |
| LASTCHANGE | Date when the file (data or catalog information) was last changed. |
| LASTCATALOGCHANGE | Date when file catalog information was last changed. |
| LASTDATACHANGE | Time and date when file data information was last changed. |
| LASTDATACHANGEDATE | Date when file data information was last changed. |
| TYPE | Type of file (LINE, SEQ, or SEQWL). |
| LOC | Type of storage device on which file is located (DISK). |
| RPM | References per month since creation date. If this is more than 999, the number of references per day is given (with D suffix); if this is more than 999, the number of references per hour is given (with H suffix); if this is more than 999, a * suffix is printed to indicate a truncated result. |
| IDLEDAYS | The number of days since the last reference. |
| PKEY | Program key associated with the file. |
| NOSAVE | If ON, this indicates that the user has specified that the file is not to be file-saved. |

Access information

These items provide information regarding the permit status of files. The relative cost of obtaining this information is moderately inexpensive.

| | |
|---|---|
| MYACCESS | Access to the file by the userID currently in use. |
| ACCESS | If this userID does not have permit access to the file, this item provides the same information as MYACCESS; otherwise, it provides information regarding the access for the owner, the access for others, and any specific access information that may exist. |

The abbreviations used in the output are:

| | |
|---|---|
| N | none |
| R | read |
| WE | write-expand |
| WC | write-change |
| W | write |
| RWE | read and write-expand |
| RWC | read and write-change |
| RW | read and write |
| RT | read and truncate |
| T | truncate |
| D | destroy |

| | |
|---|---|
| F | full |
| P | permit |
| U | unlimited |

| | |
|---|---|
| SA | (Short Access) The access information is presented in a more abbreviated form. This is valid only with COLUMNS output. |

File information

The cost of obtaining this information is relatively expensive. Any access (except none) is sufficient. In the following list, the first group of items only require opening the file and are moderately expensive to obtain. The second group of items also require reading the entire file and can be very expensive to obtain.

| | |
|---|---|
| SIZE | Current file size. |
| TRUNC | Truncated file size. |
| MAXSIZE | Maximum file size. |
| MAXLEN | Maximum length of a line. |
| EMPTY | Specifies if the file is empty. |
| EXPFAC | Expansion factor of file (this is printed as “DEF” if it is the default setting, “nn%” if it is a percentage expansion factor, or “nnP” if it is an absolute expansion factor). |
| TRUNCSAVES | Number of disk pages that would be freed if file were truncated (SIZE–TRUNC). |

| | |
|---|---|
| LINES | Number of lines. |
| AVLEN | Average length of a line. |
| MINSIZE | Minimum file size. |
| HOLES | Number of holes in file. |
| MAXHOLE | Size of maximum hole. |
| AVAILSPACE | Number of bytes of unused space (amount of space available before file expansion occurs). |
| MINSAVES | Number of disk pages that would be freed if file were reduced to minimum size (SIZE–MINSIZE). |

Name Information

The relative cost of obtaining this information is very low. Any access (except none) is valid.

| | |
|---|---|
| NAME | Name of the file. |

Summary Information

If SUMMARY is specified, a single summary line is produced that contains a summary for each item requested. The summary includes only the files for which items are printed. The items that may be summarized are:

| | |
|---|---|
| NAME | Number of files. |
| SIZE | Sum of file sizes. |
| MAXSIZE | Largest MAXSIZE value. |
| RPM | References/month for all files. |
| USECNT | Sum of use counts. |
| IDLEDAYS | Smallest IDLEDAYS value. |
| TYPE | Number of line files and the number of sequential files, respectively. |
| LASTREF | Most recent last reference date. |
| LASTDATACHANGE | Most recent last data change date. |
| LASTCATALOGCHANGE | Most recent last catalog change date. |
| CREATE | Most recent creation date. |
| MAXHOLE | Sum of maximum holes. |
| MINSIZE | Sum of all minimum file sizes. |
| MAXLEN | Largest MAXLEN value. |
| LINES | Sum of all lines. |
| AVLEN | Sum of line lengths divided by the number of lines in the file. |
| HOLES | Sum of holes. |
| AVAILSPACE | Amount of available space. |
| TRUNC | Sum of all truncated file sizes. |

Groups of Items

| | |
|---|---|
| CATALOG | The following items are provided: TYPE, IDLEDAYS, RPM, USECNT, OWNER, CREATE, LASTREF, LASTCATALOG, LASTDATA, VOLUME, PKEY, NOSAVE. |
| FILE | The following items are provided: SIZE, TRUNCSAVES, MAXSIZE, TYPE, MAXLEN, EMPTY. |
| FULLFILE | The following items are provided: SIZE, MINSAVES, TRUNCSAVES, MAXSIZE, EXPFAC, TYPE, LINES, AVLEN, MAXLEN, MAXHOLE, EMPTY. |
| ALL | The following items are provided: SIZE, TRUNCSAVES, TYPE, LINES, MAXLEN, RPM, USECNT, CREATE, LASTREF, LASTCATALOG, LASTDATA, ACCESS. |
| TOTAL | All information items. |

In the above groups, the implicit use of NOSAVE and EMPTY causes the value to be printed only if they are not the normal setting. If one of these three is explicitly requested, its value is always printed.

For ALL, access information is printed if it will fit on the remainder of the line; otherwise, only partial access information (with the asterisk to indicate there is unprinted specific sharing information) is printed. If the output form is not columnar, as for example will normally be the case on a terminal, access information is always printed, since ALL will not fit on a single line.

Defaults

For a file specification of "?" or "userid:?" with no parameters, only the names are printed. For a specific "filelist" with no parameters, CATALOG (except for PKEY and NOSAVE) followed by ACCESS is defaulted.

Selective Filtering

All information items except ACCESS and MYACCESS can also be given as a comparison that must be satisfied before the items are printed. Thus,

```
FILESTATUS TYPE
```

prints the type of the user's files, whereas

```
FILESTATUS TYPE=SEQ
```

prints only the sequential files. Most items are specified as numeric quantities; SIZE, MAXSIZE, MINSIZE, and TRUNC may be specified either with or without the P suffix, but the value is always given in pages; RPM may be given with the D or H suffixes; TYPE may be LINE or SEQ; VOLUME must be a 6-character volume name; OWNER must be a 1- to 4-character userID substring; LASTREF, LASTCHANGE, and CREATE must be an 8-character date either in the form "MMMDD/YY", where "MMM" are the first three letters of the month, "DD" is the 2-character numeric day of the month, and "YY" is the last two digits of the year, (e.g., "MAR24/77"), or in the form "MMxDDxYY", where "x" is either "/" or "–" separating the month, day, and year (e.g., "03–24–77"); PKEY must be 1 to 13 characters; NOSAVE is ON or OFF; EMPTY is YES or NO.

For TYPE, VOLUME, OWNER, PKEY, NOSAVE, and EMPTY, only the comparisons "=" or "~=" are legal. Thus, for example, TYPE=SEQ and TYPE~=SEQ are the two legal forms for the item TYPE and the value SEQ. For the other items, the comparisons "=", "~=", ">", "<", ">=", and "<=" are legal. Thus, for example, SIZE<2 or CREDATE>MAR01/87. If multiple filters are specified, all conditions specified must be met before the item is printed. Thus, for example,

```
FILES SIZE>100 TYPE=SEQ
```

will find all sequential files of more than 100 pages in size. Ranges may be obtained by specifying the same item in different comparisons. Thus, for example,

```
FILES CREDATE>=MAR01/87 CREDATE<APR01/87
```

will find all files created in March 1987.

Note that specifying a filter causes all files to be investigated to determine if they fulfill the filter specifications. Thus, the filter SIZE=10 opens all the user's files and extracts the size; the user is correspondingly charged for opening and reading all of his files, not just for opening and reading the files that were actually printed.

Program Key: *MTS.FILESTA

Examples:

```
FILES
```

The above example prints the names of the files belonging to this userID in the following format:

```
DATAFILE1 DATAFILE2 DATAFILE3 DTEST     FORMAT
PROG      PROGRAM1  PROGRAM2  PROGRAM3  RESULTS1
RESULTS2  RESULTS3  SPROG     SPROGRAM1 SPROGRAM2
SPROGRAM3
```

```
FILE DATAFILE1
```

The above example prints the catalog and access information for the file DATAFILE1 in keyword format:

```
DATAFILE1     Type=Line, Idledays=38, RPM=2,
 Usecnt=12, Owner=50AX, Create=Apr18/87,
 LastRef=Jun04/87, LastCatalog=May13/87,
 LastData=18:33:15 May14/87,
 Access=Unlim Owner, None Others
```

```
FILE DATAFILE1 ALL
```

The above example prints the information for the file DATAFILE1 in keyword format:

```
DATAFILE1     Size=1P, Truncsaves=0P, Minsaves=0P,
 Type=Line, Lines=14, Maxlen=80, RPM=2, Usecnt=12,
 Create=Apr18/87, LastRef=Jun04/87,
 LastCatalog=May13/87, LastData=18:33:15 May14/87,
 Access=Unlim Owner, None Others
```

```
FILE DATAFILE1 TOTAL
```

The above example prints the information for the file DATAFILE1 in keyword format:

```
DATAFILE      Size=1P, Minsaves=0P, Truncsaves=0P,
 Maxsize=32767P, Expfac=Def, Type=Line, RPM=2,
 Idledays=38, Lines=14, Holes=1, Avlen=19,
 Maxlen=80, Availspace=2729, Maxhole=255,
 Create=Apr18/87, LastRef=Jun04/87,
 LastCatalog=May13/87, LastData=18:33:15 May14/87,
 Volume=MTS012, Owner=50AX, Loc=3350, Usecnt=12,
          Pkey=*EXEC, Access=Unlim Owner, None Others
```

```
FILE DATAFILE2 ACCESS
```

The above example gives the full access information in the following format:

```
DATAFILE2      ACCESS=Unlim Owner, Read (P=W?,2G85),
 None WRST
```

```
FILES (DATAFILE1,DATAFILE2) ACCESS TERSE
```

The above example gives the access information for the files DATAFILE1 and DATAFILE2 in keyword format:

```
DATAFILE1      Access=U Owner, N Others
DATAFILE2      Access=U Owner, R (P=W?,2G85), N WRST
```

```
FILES (F1,F2,F3) IDLEDAYS TYPE SIZE
```

The above example prints the number of idle days, the type, and the size for the files F1, F2, and F3 in keyword format:

```
F1             Idledays=41, Type=Line, Size=12P
F2             Idledays=20, Type=Seq, Size=3P
F3             Idledays=2, Type=Line, Size=18P
```

```
FILES (F1,F2,F3) IDLEDAYS TYPE SIZE COLUMNS
```

The above example is identical to the previous example except that the information is printed in column format.

```
File name      Idle Type   Size
               Days        Pgs.

F1               41 Line     12
F2               20 Seq       3
F3                2 Line     18
```

```
FILES OUTPUT=-FILES COLUMNS
```

The above example writes a list of the user’s files into the file –FILES in column format.

## FSMESSAGE

MTS Command Description

| | |
|---|---|
| Purpose: | To invoke the full-screen interface to the MTS Message System. |
| Prototype: | FSMESSAGE [FSMessage-command] |
| | "FSMessage-command" is an optional FSMessage-system command. The command is executed as a single command and an immediate return is made to the caller. |
| Program Key: | *FSMESSAGE |
| Description: | The FSMESSAGE command provides a full-screen interface to the MTS Message System. For further details, see *MTS Volume 23: Messaging and Conferencing in MTS*, Reference R1023. |

## FTP

MTS Command Description

| | |
|---|---|
| Purpose: | To transfer files between MTS and another Internet site. |
| Prototype: | FTP [hostname] |
| Program Key: | *FTP |

Description:

FTP, the Department of Defense/Internet standard File Transfer Protocol, allows users to transfer files between MTS and any site on the Internet—the collection of hosts and networks that includes Arpanet, NSFNET, regional networks such as UMnet/Michnet, and many local networks.

In addition to the transfer of files between accounts on Internet hosts, FTP enables you to transfer files to your MTS account from guest accounts on many remote hosts (see "Anonymous FTP" below). Public MTS files are also available by using Anonymous FTP from other hosts to MTS.

While using FTP, you can enter MTS commands by prefixing the commands with a dollar sign ($).

This description contains instructions for using FTP for file transfers to and from MTS, sample terminal sessions, and detailed descriptions of FTP commands. It was adapted from a document entitled *Internet Services (TCP/IP) on MTS—MTS FTP*, published by Information Technology Services at Rensselaer Polytechnic Institute in Troy, New York.

For information on how to use the FTP services available for other computers and operating systems, please refer to their documentation. Since most of the command-driven implementations use similar commands, however, this description may be of assistance.

### Getting Started

To use FTP on MTS, sign on to MTS and enter the following command at the pound sign (#) prompt:

```
FTP
```

To establish a connection to a host computer, type the following at the FTP prompt (ftp>):

```
OPEN hostname
```

where "hostname" is the symbolic name, or Internet address, of the host with which you would like to communicate, e.g., um.cc.umich.edu. You will then be prompted to enter your login and password for the remote host.

To transfer a file from a host computer to MTS, enter the command:

GET remotefilename newfilename

where "remotefilename" is the name of the file to be transferred and "newfilename" is the name to be given to the new file. Note that the operating systems on some hosts, such as UNIX, may be case sensitive. See "Command Descriptions" below, for explanations of commands for listing files, changing directories, etc.

To transfer a file from MTS to the host computer, enter the command:

PUT localfilename newfilename

When you are finished with transfers to or from a host, you can close the connection to that host by typing:

CLOSE

You can then begin a session to another host with the OPEN command. To quit FTP, type:

BYE

An alternative way to open a connection to a host is to specify the hostname when you begin using FTP, as follows:

FTP hostname

This will immediately establish a connection with the host.

The following sample terminal session illustrates the use of FTP:

```
#ftp
um.cc.umich.edu FTP client (Version of Feb ...
ftp>  open sample-sun.stanford.edu
220 sample-sun FTP server (SunOS 4.0/3) ready.
Name (36.44.0.6.ABCD):  joew
331 Password required.
Password:  blanked
230 Login successful.
ftp>  ls
200 Port command successful
150 Opening data connection...
EXAMPLE.FILE1
EXAMPLE.FILE2
ftp>  get EXAMPLE.FILE1 NEW.FILE
ftp>  $edit NEW.FILE
:  visual
:  stop
ftp>  put NEW.FILE REMOTE.FILE
ftp>  bye
221 Goodbye.
```

### Anonymous FTP

Internet hosts often allow remote users access to public information. These public files are available through what is called “Anonymous FTP”. With Anonymous FTP, you can use “anonymous” as your user name and some identification for the password. (People often enter their Internet mailname as a password, for example, joew@um.cc.umich.edu). If you do not have an Internet mailname, you can enter your own MTS userID or other text.) You can then access files that have been made public on the remote host. In this way, you can transfer files without having an account on the remote system.

Using Anonymous FTP, a logon sequence would look something like:

```
ftp>  open sumex-aim.stanford.edu
220 sumex-aim FTP server (SunOS 4.0/3) ready.
Name (36.44.0.6.ABCD):  anonymous
331 Guest login ok, send ident as password.
Password:  blanked
230 Guest login ok, access restrictions apply.
ftp>
```

### Remote FTP to MTS

In addition to using FTP to transfer files while you are signed on to MTS, you can transfer files to and from MTS while logged on to a remote system. The registered hostnames for the MTS systems are um.cc.umich.edu and ub.cc.umich.edu. Some systems may require that you use the numerical IP addresses of the MTS hosts rather than the hostnames. For UM-MTS, the numerical address is 35.1.1.43 and for UB-MTS, it is 35.1.1.47.

You can use these names and numerical addresses to access files on your own MTS account and files that are available for Anonymous FTP, e.g., those that are permitted READ PKEY=*FTP. The following sample FTP session from a remote Sun computer shows how you can access files from UM-MTS.

```
ftp>  open um.cc.umich.edu
um.cc.umich.edu FTP Server (Version of Mar ...
Name (35.1.1.43:joew):  ABCD
331 Password required.
Password (35.1.1.43:joew):  blanked
230 Login successful.
ftp>
```

Note: The question mark (?) is used as a wildcard character on MTS rather than the asterisk (*), which is used as a wildcard on UNIX systems.

### File Types Used in Transfers

Use the following file types when transferring files ending in these suffixes:

| Suffix | File Type |
|---|---|
| .arc | BINARY |
| .tar | BINARY |

| | |
|---|---|
| .shar | ASCII |
| .tar.z | BINARY |
| .shar.z | BINARY |
| .z | BINARY |

If the remote host is a TOPS-20 or DEC-10 system, use TENEX instead of BINARY.

The default file type is ASCII, which is used for all text files.

To change the file type before transferring a binary file, for example, use the command

```
TYPE BINARY
```

For additional information about file types, see the MTS file RFC:RFC959.TXT.

## Command Descriptions

APPEND localfile [remotefile]

> Appends a local file to a file on the remote machine. If you do not specify "remotefile", the local filename will be used for the new file. The current settings for type, format, mode, and structure are used for the file transfer.

ASCII

> Sets the file-transfer type to network ASCII. This is the default type.

BINARY

> Sets the file-transfer type to BINARY.

BYE

> Terminates the FTP session with the remote server and exits FTP. QUIT and STOP are synonyms for BYE.

CD remotedirectory

> Changes the working directory on the remote machine to "remotedirectory". For example, a CD command to a Sun workstation might look like:

```
cd /usr/bin.
```

CLOSE

> Terminates the FTP session with the remote server and returns to local FTP command mode.

DELETE remotefile

Deletes the file called “remotefile” on the remote machine.

DIR [remotedirectory] [localfile]

Lists directory information for selected files on the remote host. If you specify “localfile”, the directory listing will be written to a file with that name. If no directory is specified, the current working directory of the remote machine is used. If no local file is specified, the contents will be displayed on your screen. For example:

```
dir /usr/bin mtsfile
```

EBCDIC

Sets the file-transfer type to EBCDIC.

FILESTATUS

Lists the files available in the working directory on the remote account. This is a synonym for LS. Note: This command only works from MTS.

GET remotefile [localfile]

Retrieves “remotefile” and stores it on the local machine. If you do not specify “localfile”, the file on the local machine is given the name it has on the remote machine. The name will be converted to uppercase on MTS and will be truncated if longer than 12 characters. The current settings for type, mode, and structure are used while transferring the file.

HASH

Turns on and off the hash-sign printing, which displays a pound sign (#) as each data block is transferred, thereby showing the progression of the transfer. The size of a data block is approximately 1024 bytes. By default, HASH is off.

HELP [command]

Displays an explanation of a command. If you do not specify a command, FTP displays a list of the known commands. The question mark (?) is a synonym for HELP.

IMAGE

Sets the file-transfer type to IMAGE, which is a synonym for BINARY.

LCD directory

Changes the default directory on the local host, which has the effect of changing the MTS account from which you are transferring files, as in

LCD memo. Note that for accounts on MTS, you must include a colon (:) directly following the userID.

LPWD

Displays the name of current working directory of the local machine. On MTS, this is the account in which you are using FTP, unless you have specified another account with the LCD command. Note: This command only works from MTS.

LS [remotedirectory] [localfile]

Lists the files in a directory on the remote machine. If you do not specify "remotedirectory", the current working directory is used. If you do not specify "localfile", the files are displayed on the screen. LS is a synonym for FILESTATUS.

MDELETE remotefiles

Deletes "remotefiles" on the remote machine. This command accepts a filename pattern using a wildcard character, such as an asterisk (*). For example, the following command would delete all files ending in .tar on a remote UNIX system:

```
mdelete *.tar
```

If you specify a pattern in this way and if interactive prompting is on, you will be prompted for each file you want to delete. (See the explanation of the PROMPT command.)

MDIR remotefiles [localfile]

This command is like DIR, except that you can specify a filename pattern using a wildcard character (for example, an (*) in "remotefiles". If interactive prompting is in effect, FTP will prompt you for the file you want. (See the description of the PROMPT command.)

MGET remotefiles

This command works like GET, except that you can specify a filename pattern by using a wildcard character, such as an asterisk (*), in remote-files.

MKDIR directoryname

Makes a new directory called "directoryname" on the remote machine. Because MTS does not use directories like those found in other systems, you cannot create directories using this command when connected to MTS from a remote host.

MLS remotefiles [localfile]

> This command is like LS, except that you can specify a filename pattern by using a wildcard character, such as an asterisk (*), in “remotefiles”.

MODE modename

> Sets the file transfer mode to “modename”. Currently, the only mode in FTP is STREAM.

MPUT localfiles

> This command works like PUT, except that you can specify a filename pattern by using a wildcard character, such as an asterisk (*), in “localfiles”. If interactive prompting is in effect, FTP will prompt you for the files.

OPEN hostname

> Establishes a connection to the FTP server on the specified host.

PROMPT

> Toggles interactive prompting. If prompting is on (the default), then when you use any of the multiple remote commands (e.g., MGET, MPUT, and MDELETE), FTP will prompt you for a confirmation for each file, either the name of the file, or a Yes or No.

PUT localfile [remotefile]

> Transfers “localfile” to the remote machine. If you do not specify “remotefile”, the name you specify as “localfile” will be used. The file transfer uses the current settings for type, mode, and structure.

PWD

> Displays the name of the current working directory on the remote machine.

QUIT

> Terminates the FTP session with the remote server and exits FTP. It is a synonym for BYE and STOP.

QUOTE arg1 arg2 ...

> Sends the arguments you specify, verbatim, to the remote FTP server. This command is designed to let the advanced user issue internal FTP protocol commands. See the MTS file RFC:RFC959.TXT for more information.

RECV remotefile [localfile]

Retrieves “remotefile” and stores it in “localfile” on the local machine. RECV is a synonym for GET.

REMOTEHELP [command]

Requests help from the remote FTP server. If you specify the command, it is supplied to the server as well.

RENAME oldname newname

Renames the file “oldname” on the remote machine to “newname”.

RMDIR directoryname

Deletes a directory called “directoryname” on the remote machine. Because MTS does not use directories like those found in other systems, you cannot delete directories using this command when connected to MTS from a remote host.

SEND localfile [remotefile]

Transfers “localfile” to the remote machine and stores it in “remotefile”. SEND is a synonym for PUT.

STATUS

Shows the current status of FTP, which includes options that you can set, and whether or not you are connected to a remote FTP server.

STOP

Terminates the FTP session with the remote server and exits FTP. It is a synonym for QUIT and BYE.

STRUCT structname

Sets the file transfer structure to “structname”, which can be FILE, RECORD, or PAGE. The default is FILE. Note that RECORD may not be supported by FTP implementations on other hosts. Although TENEX supports PAGE, the implementation is incompatible with PAGE in MTS. STRUCT PAGE in MTS, in conjunction with TYPE LOCAL 32, is used to transfer files between MTS sites while preserving file line numbers.

TENEX

Sets the file-transfer type to TENEX for use with computers running TENEX.

TYPE typename

Sets the file transfer type to “typename”. If you do not specify “typename”, the current type is used. The default is network ASCII. Other typenames include BINARY (or IMAGE), EBCDIC, and LOCAL BYTE SIZE (for computers running TENEX or for MTS-to-MTS file line-number preservation). For example:

```
type binary
```

LOCAL BYTE SIZE is entered as “local #”, where “#” is the byte size you want to specify. For example,

```
type local 8
```

For transferring files between MTS sites while preserving line numbers, one should issue TYPE PAGE followed by TYPE LOCAL 32.

Note: If you want to run a binary file through a program on MTS, MTS FTP limits lines to 256 characters.

USER username [password]

Identifies you to the remote FTP server. If you do not specify a password and the server requires it, FTP will prompt you for it; the line will remain blank while you type. If the remote FTP server requires additional account information, it will prompt you for it also.

VERBOSE

Toggles verbose mode, in which all responses and messages from the FTP server are printed on your screen. In addition, when a file transfer is completed and verbose mode is on, you will see statistics regarding the efficiency of the transfer. By default, VERBOSE is on.

? [command]

? is a synonym for HELP.

## HELP, EXPLAIN

MTS Command Description

| | |
|---|---|
| Purpose: | To enter the MTS Help/Explain facility. |
| Prototype: | {HELP \| EXPLAIN [topic]<br><br>"topic" is an optional Help/Explain topic for which information is desired. If omitted, the user is queried for a topic. |
| Program Keys: | *MTS.HELP and *MTS.EXPLAIN |
| Description: | The HELP or EXPLAIN command provides direct on-line assistance for using MTS.<br><br>The MTS SET HELPMODE={LINE \| SCREEN} command specifies whether the help information is presented in line or full-screen format. The default is SCREEN if the user is using a terminal with a display screen. |

## IF

MTS Command Description

Purpose: To test the program execution return code.

Prototype: IF RUNRC condition integer, MTS-command

RUNRC

RUNRC is the execution return code resulting from the execution of the previous user program (executed or loaded by RUN, RERUN, START, RESTART, LOAD, or DEBUG).

condition

"condition" is a conditional comparison operator which may be =, ¬=, <, >, <=, >=, .EQ., .NE., .LT., .GT., .LE., or .GE.

integer

"integer" is an (optionally signed) constant against which the comparisons of RUNRC are made.

MTS-command

"MTS-command" is the MTS command to be executed if the comparison is true.

Program Key: *MTS.IF

Description: The IF command tests the value of the execution return code resulting from the execution of the previous user program. The test is in the form of a simple comparative against an (optionally signed) integer constant. If the comparison is true, the MTS command specified in the IF command is executed; if the comparison is false, the MTS command is not executed.

Blanks are allowed (but not required) between RUNRC, "condition", "integer", the comma, and "MTS-command"; the comma is required. The dollar-sign command flag is optional with "MTS-command".

The execution return code from a program is an indication of how well the program executed. For most compilers, the value returned as the execution return code usually matches the level of the most severe diagnostic error message issued by that compiler. Thus, a return code of 0 means that the program compiled successfully, 4 means that warning messages were generated, and 8 means that serious errors were detected in the source program. For compilers and other programs residing in public files, the appropriate documentation should be consulted for specific meanings of the return codes.

When the execution of a program (user program, compiler, or other public file) is initiated, MTS sets the execution return code to −1 (to mean "undefined"). When execution of the program is terminated, MTS sets the execution return code to the value that was in general register 15 when the program terminated. This is traditionally a small nonnegative number which is a multiple of 4. When the program terminates successfully, the value 0 is returned in general register 15, and the execution return code is therefore set to 0. If the program terminates by calling the system subroutine SYSTEM, MTS sets the execution return code to 0; if the program terminates by calling the system subroutine ERROR, the execution return code is set to 8. In all other cases of program suspension, such as attention interrupts, program interrupts, timer interrupts, or calls to the system subroutines MTS or MTSCMD, the execution return code remains set at −1 since the program may be restarted and run to a conclusion. For further information about return codes from programs, see the section "Calling Conventions" in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

A more powerful and comprehensive method of testing program return codes and other system variables is available by using the MTS command macro processor. Complete details are given in *MTS Volume 21: MTS Command Extensions and Macros*, Reference R1021.

Example:

```
RUN *FTN INPUT=SOURCE OBJECT=PROGRAM
IF RUNRC > 0, SIGNOFF
RUN PROGRAM 5=DATA 6=RESULTS
```

In the above example, the FORTRAN source program in the file SOURCE is compiled by *FTN into the file PROGRAM. If any errors are detected during compilation, the execution return code will be set to 4 or 8 by the compiler and the following IF command will terminate the job by signing off the user. If the compilation is successful, the resulting program will be executed.

## LIST

MTS Command Description

Purpose: To list a file or device. This is a new version of the LIST command which has its own subcommand language.

Prototypes: LIST FDlist [[{ON | TO}] FDname] [[WITH] options]
LIST FDlist WITH options [{ON | TO} FDname]
LIST

If no parameters are given, list command mode is entered (see below).

FDlist

"FDlist" specifies the file(s) or device(s) that contain the lines to be listed. "FDlist" may be a single file or device name, e.g.,

```
LIST FILE1
```

or a list of file and device names, separated by commas, with optional parentheses:

```
LIST (FILE1,FILE2,FILE3)
LIST FILE1,FILE2,FILE3
```

File names may contain the character "?" which represents zero or more arbitrary characters in the file name and forms a pattern for matching similar file names. In this case, all files of the userID are listed that match the file-name pattern. For example, the file name "DATA?" could match the files DATA, DATA1, DATA2, DATA10, DATAFILE, etc. The "?" cannot be used in the userID portion of a shared file name.

Each element in "FDlist" may be an explicit concatenation of file and device names with or without line-number ranges and modifiers, e.g.,

```
LIST (FILE1@IC,FILE2(1,10)+FILE3(1,10))
```

The "FDlist" parameter must be specified as the first parameter of the LIST command.

[{ON | TO}] FDname

"FDname" specifies the file or device that is to receive the listed lines. "FDname" may be an explicit concatenation of file and device names with or without line-number ranges, but not a list of FDnames or a filename pattern. If "FDname" is omitted, the output lines are written on *SINK*.

ON or TO is not required unless "FDname" follows other options.

[WITH] options

Several options may be specified that control the format of the listing. The word WITH implies that at least one option follows.

The WITH word is mandatory only when ON or TO FDname is given following "FDlist".

The options are given below. Free-verb options, options without equal signs (=), may be negated by prefixing them with NO or "~".

[NO]CC Default: CC

The CC option controls whether logical carriage control is used during a list request. If CC is specified, carriage control is generated according to the following rules:

The first page of a file is form-fed to the next sheet, and all subsequent pages are skipped to the next page. The first line of text on a page is started by positioning to the top of the page. The footing, if requested, is printed two lines below the last line of text on a page.

Note: The actual carriage-control characters generated during a file listing can be set using the NP, NS, PT, and SS options (see below).

{COLUMNS | COLS}={n | MAXIMUM} Default: COLUMNS=1

The file may be listed in a multi-column format specifying the COLUMNS option. "n" is the number of columns on a page (the column widths are determined by the page width as set by the PAGEWIDTH option), or to the maximum number of columns that will allow a listing of the file without folding any lines. Thus, if the longest line in the file is 30 characters, the page width is 100, and line numbers are being excluded from the listing, three columns would be printed per page for the COLUMNS=MAX option.

{COLWIDTH | CW}=n Default: COLWIDTH=121

The page is divided into as many columns of width "n" as will fit. The COLUMNS and COLWIDTH options interact with one another. Specifying COLWIDTH forces the maximum number of columns, if the COLUMNS option was not specified. Specifying COLUMNS forces the maximum (equal) column width, if the COLWIDTH option was not specified. If the values specified for COLUMNS and COLWIDTH are incompatible, an error message will be issued. COLWIDTH=121 is the maximum value that may be specified for single-column output.

The value of COLWIDTH is also dependent on the value of PAGEWIDTH. In this case,

```
LIST FILE WITH COLWIDTH=121...PAGEWIDTH=TERMINAL
```

will produce an error when the terminal's output length (OUTLEN) is 80.

[NO]CONCATENATION Default: NOCONCATENATION

When NOCONCATENATION is specified, files separated by commas are separated by a New Sheet form-feed during listing. This also applies to multiple files generated by use of a file pattern. When CONCATENATION is specified, file are listed one after another as if they were one contiguous file. The CONCATENATION option should not be confused with the SKIP option, as SKIP applies only to implicitly or explicitly concatenated files, whereas CONCATENATION controls the listing of file lists only.

[NO]DOUBLESPACE Default: NODOUBLESPACE

The DOUBLESPACE option allows the simulated double-spacing of a listing. If double-spacing of a line causes a line to be printed on a line number greater than the current lines-per-page setting, the form is skipped and printing is done at logical position Page Top. NODOUBLESPACE is synonymous with SINGLESPACE.

FOOTER[={"text" | RULER}] Default: NOFOOTER

Footers (if requested) are printed at the bottom of a listing with one blank line separating the footer from the main text of the listing. These two additional lines are subtracted from the space available for the main text. FOOTER=RULER prints a column-number ruler as the footer. If FOOTER is specified, the default footer is printed (same as FOOTER=RULER). The default footer may be changed by specifying FOOTER="text". The predefined "macros" <DATE>, <TIME>, <HOST>, <CCID>, and <FDNAME> may be used in the text string to substitute the corresponding values in the text string to substitute the corresponding values in the footer.

HEXCON[=(l,r)] Default: See text

The column range indicated (from "l" to "r", inclusive) is printed in hexadecimal. If "l" and "r" are omitted, the entire line is printed in hexadecimal. The printed line is increased in length by the size of the column range, since it takes two columns to print the hexadecimal code of a single character.

[NO]IC Default: NOIC

The IC option controls implicit concatenation. (See the section on file concatenation below.)

[NO]LENGTH Default: NOLENGTH

The LENGTH option specifies that the actual, untruncated length of each line is printed. This length is printed to the left of the printed line, or if line numbers are printed, to the right of the line numbers.

{[NO]LINENUMBERS | [NO]LNR}     Default: LINENUMBERS

The LINENUMBERS option specifies that line numbers are printed with the lines in the listing. NOLINENUMBERS suppresses the printing of line numbers.

If line numbers are requested, the line numbers returned from the read operation on "FDlist" are appended on the front of each line.

{LINES | LPP}={n | OFF | DEFAULT}     Default: LINES=DEFAULT

The LINES option sets the number of lines per page (including the header and the footer). The maximum value for "n" is 126. LINES=OFF suppresses pagination. LINES=DEFAULT suppresses pagination when listing to a terminal and sets LINES=60 when listing to a file or a printer.

MARGINS={([l],[r]) | l}     Default: MARGINS=(0,0)

The MARGINS option specifies that the left and right margins are to be indented by "l" and "r", respectively. MARGINS=(l,) is equivalent to MARGINS=l; MARGINS=(,) and MARGINS=(0,0) are equivalent.

{NPI | NEWPAGEINTERVAL}=n     Default: NPI=0

If the NPI option is specified, a new page is started after every "n" lines. NPI=0 disables the option.

NP=char     Default: NP=;

The NP (New Page) option sets the carriage-control character that is used to advance to the physical top of the next page (physical line 1).

NS=char     Default: NS=:

The NS (New Sheet) option sets the carriage-control character that is used to advance to the top of the next sheet.

PAGES={n | OFF}     Default: PAGES=OFF

The PAGES option limits the output per file to "n" pages. "FDlist" list elements are treated as single files, regardless of implicit or explicit concatenations. Thus, exceeding PAGES=n terminates processing of the current list element. PAGES=OFF implies no page limit. See the section below on file concatenation for further information about the behavior of this option.

{PAGEWIDTH | PW}={n | TERMinal}     Default: PAGEW=132

The PAGEWIDTH option controls the width of the printed page. The value of "n" must be in the range (0,250). For terminals, PAGEWIDTH=TERMINAL reduces the page width to the current %OUTLEN device-command setting (for terminals that do not support the

%OUTLEN command, the default is 132). The page width does not include the carriage-control character.

PLAIN

The PLAIN option produces listings without titles, without page skips between concatenated files, and with implicit concatenation controlled by the system and user defaults. Any other options specified along with the PLAIN option will be ignored.

PT=char Default: PT=1

The PT (Page Top) option sets the carriage-control character that is used to advance to the logical top of the next page (physical line 3).

{SEPARATOR | SC}={c | OFF} Default: SEP= |

The SEPARATOR option sets the column separator character to “c” or suppresses the printing of the separator character (SEPARATOR=OFF).

[NO]SINGLESPACE Default: SINGLESPACE

The SINGLESPACE option causes the listing of files to be single-spaced. SINGLESPACE is synonymous with NODOUBLESPACE.

[NO]SKIP Default: NOSKIP

Files either implicitly or explicitly concatenated are printed separately, if SKIP is specified, or as a single file, if NOSKIP is specified.

SS=char Default: SS=Blank

The SS (Single-Space) option sets the carriage-control character that is used to advance to the next line.

TITLE[={"text" | BLANK}] Default: See text

Titles (if requested) are printed at the top of the page, with one line separating the title from the listing. These two lines are subtracted from the space available for the listing. TITLE=BLANK produces no title other than a page number. The predefined “macros” <DATE>, <TIME>, <HOST>, <CCID>, and <FDNAME> may be used in the text string to substitute the corresponding values in the title. If TITLE is specified, the default title is printed:

```
"Listing of <FDname> at <time> on <date> for
CCid=<userid> on <host>"
```

The default title may be changed by specifying TITLE="text". The default is NOTITLE, when listing on a terminal, and TITLE, otherwise.

TRUNCATE={n | OFF} Default: TRUNCATE=OFF

If the TRUNCATE option is specified, only the first "n" columns of each line are printed.

[NO]WRAPPING Default: WRAPPING

When WRAPPING is specified, lines that are longer than the column width are not truncated but instead continued on the next line in the same column (or the next column if the previous line was the last line in the column). With TRUNCATE=OFF and WRAPPING specified, all lines can be listed regardless of length. When NOWRAPPING is specified (even if TRUNCATE=OFF), the line is ended when it reaches the end of the line on which it began.

Program Key: *LIST

Description: The LIST command performs a series of read and write operations. Lines are read sequentially from "FDlist" and written to "FDname" until the end of the file or the end of a line-number range is encountered on "FDlist".

The LIST command (unlike the COPY command) does not treat the first column of each line as a carriage control.

List Command Mode

If the LIST command is given without any parameters, list command mode is entered. In list command mode, separate list and control commands may be given to list multiple files in different formats for each file.

The following commands may be used in list command mode:

```
LIST FDlist [[{ON | TO}] FDname] [[WITH] options]
SET options
EXPLAIN [item]
MTS
RETURN
STOP
$mts-command
```

The LIST command in list mode is the same as in MTS command mode (indicated by the ">" prefix character). Options specified on the LIST command are effective only for that command and take precedence over options specified by the SET command.

The SET command may be used to set format options for subsequent LIST commands, e.g.,

```
SET TITLE NOLNR
```

enables the printing of titles and suppresses the printing of line numbers. The same options are available for the SET command as are described above for the

LIST command.

The EXPLAIN command may be used to obtain further information about the LIST command processor.

Both the RETURN and MTS commands return control to the caller (normally MTS command mode). The STOP command terminates the LIST command processor. Other MTS commands may be executed from list command mode by prefixing them with a dollar sign ($).

File Concatenations

Explicit concatenations of elements of an "FDlist" are treated as a single file; no page skips are made for each file of the concatenation.

Each element in a list of files of an "FDlist" is treated as a separate file; hence, page skips are made between each file in the list. If the CONCATENATION option is specified (the default is NOCONCATENATION), page skips are not made between files in the list. Note that files being listed as the result of a file name pattern are considered to be files in a list and are treated as such.

Page skips are made between each LIST command in list command mode unless the NOSKIP option is explicitly specified.

By default, implicit concatenations ($CONTINUE WITH lines) are ignored unless the IC option is specified or the @IC modifier is specified on the file names in "FDlist". If implicit concatenations are in effect, they are treated just like explicit concatenations (no page skips are made).

Examples:

```
LIST A
```

File A is listed on *SINK* (default). If A is a line file, the line numbers from file A are appended to the front of output lines before they are written on *SINK*. If file A is a sequential file, line numbers starting at 1 and incremented by 1 are appended to the output lines.

```
LIST A(5,20) OUTFILE
```

Lines 5 through 20 of file A are listed on the file OUTFILE. The line numbers from A are appended.

```
LIST A+B+C ON *PRINT*
```

Files A, B, and C are listed on *PRINT* with line numbers. A, B, and C are treated as a single FDname with no page skips between the files.

```
LIST (A,B,C) TO *PRINT*
```

Files A, B, and C are listed on *PRINT* with line numbers. A, B, and C are treated as separate FDnames with page skips between the files.

```
LIST (A,B,C) *PRINT* WITH TITLE NOLNR
LIST (A,B,C) *PRINT* TITLE NOLNR
```

Files A, B, and C are listed on *PRINT* with the default title and without line numbers. Both of the above commands are identical in effect.

```
LIST A,B,C WITH CON
LIST A+B+C
```

Files A, B, and C are listed to *SINK* as a single file. The CONCATENATION option suppresses page breaks in the file list. Both of the above commands are equivalent in terms of the output produced.

## LOAD

MTS Command Description

Purpose: To load a program without initiating execution.

Prototype:

```
LOAD [program] [I/Ounits] [limits] [mapoptions]
    [{EXECPKEY | PKEY}={key | OFF}] [PAR=parameters]
```

The following parameters may be given:

program

"program" specifies the file or device containing the program to be loaded. If omitted, the program is loaded from *SOURCE*.

I/Ounits

The keyword parameters "I/Ounits" are the assignments of logical I/O units to files or devices for use by the loaded program during execution. The logical I/O unit assignments are used to select appropriate I/O subroutines to be used by the loaded program for the input and output of data. Where no specifications are stated, the following default assignments occur:

```
SCARDS=*SOURCE*
GUSER=*MSOURCE*
SERCOM=*MSINK*
SPRINT=*SINK*
SPUNCH=*PUNCH*        (batch mode if global card est. > 0)
```

Synonyms may be used for the following logical I/O unit names:

```
INPUT for SCARDS
PRINT for SPRINT
OBJECT for SPUNCH
```

The logical I/O units 0 through 99 have no default specifications. See the section "Files and Devices" in this volume and the subroutine descriptions for INPUT (SCARDS), PRINT (SPRINT), OBJECT (SPUNCH), SERCOM, GUSER, READ, and WRITE in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for further details on the use of these subroutines.

FORTRAN users are reminded that MTS logical I/O units 0 through 99 are not necessarily the same as the FORTRAN logical I/O units 0 through 99.

limits

The keyword parameters "limits" specify local limits for CPU time, pages printed, cards punched, and plot time. These can be given in the form:

TIME={t | tS | tM}
PAGES=p
CARDS=c
{PLOTTIME | PT}={t | tM}

These local limits can be used in both batch and conversational mode and are effective only for the execution of the program loaded by the LOAD command. For further details on limits, see the section “UserIDs, Limits, and Sigfiles” in this volume.

MAP[=mapFDname] [NOMAP] [XREF] [UXREF]

The MAP parameter specifies the file or device “mapFDname” on which the loader is to write the loader map. If the MAP parameter is omitted, no loader map is written. If the MAP parameter is given in the form of MAP, “mapFDname” defaults to *SINK*. The NOMAP parameter suppresses the printing of the loader map. NOMAP need be given only if a previous MAP specification was made.

The XREF parameter specifies that a cross-reference listing of external symbols occurring in the loaded programs is to be produced. If XREF is given without the MAP parameter, MAP=*SINK* is assumed. The XREF parameter implies the UXREF parameter.

The UXREF parameter specifies that a cross-reference listing of undefined external symbols occurring in the loaded programs is to be produced. If UXREF is given without the MAP parameter, MAP=*SINK* is assumed.

{EXECPKEY | PKEY}={key | OFF}

The EXECPKEY parameter “key” specifies an override program key to be used instead of the program key associated with “program”. If OFF is specified, the override specified by the EXECPKEY option of the SET command is disabled. For further details on the use of program keys, see the section “Files and Devices” in this volume.

PAR=parameters

The PAR field specifies an arbitrary string of characters to be passed to the loaded program on initiation of execution. This is usually a parameter list for the program and its interpretation depends on the loaded program. The PAR field must be the last parameter specified in the command. The parameter list is terminated by the end of the command line. Note that the parameter field always has a blank added after the last character and the length count is increased by one.

Program Key: *MTS.LOAD

Description: The LOAD command invokes the dynamic loader to load the object program into virtual memory. If there are unresolved external symbol references after loading from “program”, loading continues from *LIBRARY (the system library). Only those parts of *LIBRARY required to resolve the references are loaded. If there are still unresolved external references, a fatal loading error exists. In conversational mode, the loader prompts for more loader input; in batch mode, an error comment is produced and the loading terminates immediately. The search of the system library may be modified or suppressed by the LIBR, LIBSRCH, and *LIBRARY options of the SET command. If SET DEBUG=ON has been specified, SDS processes any symbol table information associated with the loaded program. For a complete description of the loading process, see the section "The Dynamic Loader" in *MTS Volume 5: System Services*, Reference R1005.

If there are no fatal loading errors, control is returned to the user; he can use either debug mode or the DISPLAY and ALTER commands to display and modify parts of the loaded program. Execution of the program can be initiated by the START command or by the debug mode RUN command.

The parameter string (specified by the PAR field) is passed as follows: GR1 contains the location of a fullword address constant which points to a region containing a halfword count (halfword aligned) followed by an EBCDIC character region (of byte-length specified by the count) containing the parameter string. The leftmost bit of the address constant is 1.

If a file or device specified by "I/Ounits" is nonexistent or is not available, the logical I/O unit referring to the unavailable file or device is set up such that the first time the program being executed refers to the logical I/O unit, either the user is given an opportunity to respecify the FDname (in conversational mode) or execution is terminated (in batch mode).

If "program" has a nondefault program key that is not prefixed by the current userID, the program key will be set to the default value (for this invocation of the program only).

Example:

```
LOAD OBJPROG+PROGLIB 5=INPUT 6=OUTPUT
```

This loads the program and private library in OBJPROG+PROGLIB. Logical I/O units 5 and 6 are specified for the input and output of data during a later execution of the program initiated by the START command or the debug mode RUN command.

## LOCATE

MTS Command Description

Purpose: To determine the status of jobs released to the Resource Manager.

Prototype: LOCATE {SYSTEM | LOCAL | FULL | SHORT | HELP}

LOCATE {jobnumber | jobname} [option ...]

where

SYSTEM

Display status of all jobs in the system.

LOCAL or LCL

Display status of all local jobs in the system.

FULL

Display the status of the user's jobs. This is the default if no parameters are specified.

SHORT

Display the status of the user's jobs, excluding purged or deleted jobs.

{jobnumber | jobname | JOBNAME=jobname}

Display information about specified jobs.

option

Display information about certain types of jobs, where "option" is:

| | |
|---|---|
| PRINT | display all print jobs. |
| EXECUTE | display all batch jobs waiting execution. |
| IMPORT | display all import jobs. |
| VIEW | display all print jobs to ROUTE=VIEW. |
| PRINTQ | display all print jobs excluding ROUTE=VIEW. |
| SUMMARY | display summary information only. |

HELP

Display help information for the LOCATE command.

Program Key: *LOCATE

Description: The LOCATE command enables the user to determine the status of jobs that have been released to the Resource Manager, for example, *BATCH* or *PRINT* jobs.

The following examples illustrate the use of the LOCATE command.

```
LOCATE PRINT SUMMARY
```

displays summary information about all print jobs.

```
LOCATE 234567
```

displays information about job 234567.

Any time you need help on the LOCATE command, you can enter

```
LOCATE HELP
```

at the MTS "#" prompt.

You can only get information about jobs that were submitted by the same userID as that issuing the LOCATE command.

## LOCK

MTS Command Description

Purpose: To explicitly lock a file.

Prototype: LOCK filename [how] [options]

filename

"filename" is the name of the file to be locked.

how

"how" indicates how the file is to be locked and should be one of the following:

| | |
|---|---|
| READ | Lock the file for reading |
| WRITE | Lock the file for modification |
| MOD | Lock the file for modification |
| CHANGE | Lock the file for modification |
| EMPTY | Lock the file for modification |
| TRUNCATE | Lock the file for modification |
| RENUMBER | Lock the file for modification |
| DESTROY | Lock the file for destroying |
| RENAME | Lock the file for destroying |
| PERMIT | Lock the file for destroying |
| NONE | Unlock the file (the same as UNLOCK) |

If "how" is omitted, the file is locked for modification.

options

"options" indicates whether MTS should wait if locking is not possible at this particular time and/or whether MTS should terminate the job (quit) if the file cannot be locked as requested. The legal options are as follows:

```
{WAIT | NOWAIT}
{QUIT | NOQUIT}
```

The defaults are WAIT and NOQUIT.

Program Key: *MTS.LOCK

Description: The three locking classes, reading, modification, and destroying, are inclusive in the sense that locking a file for modification also locks a file for reading, and locking a file for destroying also locks a file for reading and modification.

Any number of tasks (jobs) can have a file locked for reading at any one time, but only one task can have a file locked for modification, and then only if no other task has the file locked for reading. Only one task can have a file locked for destroying, and then only if no other task has the file open or locked for reading or modification.

The userID must have the appropriate access to the file before the file is locked as requested.

The file is always locked as requested (assuming the job has no other locking requests associated with the file).

If the file cannot be locked because it is locked by another task, the LOCK command will wait until the file is released by the other task (unless the NOWAIT option is specified). This waiting may be canceled by an attention interrupt, in which case the message

```
     "FDname"-Wait to lock interrupted or cancelled.
```

is printed.

The file may be unlocked with the UNLOCK command.

A file may also be explicitly locked from a program by calling the LOCK subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

See Appendix D to the section “Files and Devices” for further details about using shared files.

Examples:

```
LOCK FILE1 EMPTY
```

The file FILE1 is locked for modification.

```
LOCK FILE2 DES NOWAIT
```

The file FILE2 is locked for destroying; if locking is not immediately possible (e.g., because another job has the file locked), the locking is not done.

## LOCKSTATUS

MTS Command Description

Purpose: To display the locking status of a file or of the current job.

Prototype: LOCKSTATUS [item [options]]

LSTATUS is an alternate name for LOCKSTATUS.

item

"item" specifies the item whose locking status is to be displayed. If the parameter is omitted, the locking status of the current job is displayed.

filelist

"filelist" is a single file, a file-name pattern, or a parenthesized list of files.

JOB={nnnnnn | ME}

"JOB=nnnnnn" displays a list of files locked by the specified task number. JOB=ME specifies the current task number (the same as giving no "item" parameter).

USER=userid

"USER=userid" displays a list of files locked by the specified userID.

options

REPEAT=seconds

Print locking information every "n" seconds.

OUTPUT=FDname

Print information to the file or device specified by "FDname".

The following options may be specified to filter unwanted information.

| | |
|---|---|
| LOCK | Print information only for locked files, not for files that are just open or waiting to be locked. |
| LOCKDESTROY | Print information only for files locked for destroying. |
| LOCKMOD | Print information only for files locked for modification. |

| | |
|---|---|
| LOCKREAD | Print information only for files locked for reading. |
| WAIT | Print information only for files waiting to be locked, not for files that are already locked. |
| WAITDESTROY | Print information only for files waiting to be locked for destroying. |
| WAITMOD | Print information only for files waiting to be locked for modification. |
| WAITREAD | Print information only for files waiting to be locked for reading. |
| WAITOPEN | Print information only for files waiting to be locked as open. |
| INVALID | Print information only for files marked as invalid, that is, the file was locked and is now unlocked, but in the meantime another task changed the file so this task's file buffers are no longer accurate. |
| OPEN | Print information only for files that are open (another task can read or write to them, but cannot destroy or permit them). |
| NOTONOTL | Print information only for files that are not open and not locked. |
| ANY | Print information only for all files. |

Program Key: *MTS.LSTATUS

Description: If "filelist" is given, the system indicates all the jobs (tasks) and associated userIDs which have opened, locked, or are waiting on the file(s).

If "JOB nnnnnn" is given, the system indicates the files which that job (or the current job, if no parameter or the ME parameter is given) has open and/or locked.

The responses and their meanings are as follows:

| | |
|---|---|
| NOTL | The file is not locked. |
| LOCKR | The file is locked for reading. |
| LOCKM | The file is locked for modification. |
| LOCKD | The file is locked for destroying. |
| OPEN | The file is open. |
| NOTO | The file is not open. |

| | |
|---|---|
| WAITR | The specified job is waiting for read access to the file. |
| WAITM | The specified job is waiting for modification access to the file. |
| WAITD | The specified job is waiting for destroy access to the file. |
| WAITO | The specified job is waiting to open the file. |
| INVLD | This is of internal significance only. It signifies that while the file was unlocked by the job which had it open, another job locked and modified information in the file. Because of this, certain information concerning the file is rendered invalid and will be retrieved implicitly from the file again if it is reused. |

Examples:

```
LOCKSTATUS MFT
```

A possible response is

```
0021(IDEB)-OPEN LOCKR 0037(SX99)-WAITM
```

which means that job 0021 with userID IDEB has file MFT open and locked for reading, and job 0037 with userID SX99 is waiting to modify file MFT. The numbers 0021 and 0037 are the supervisor task numbers for those jobs.

```
LOCKS
```

A possible response is

```
X-OPEN LOCKM Y-NOTO LOCKD
```

which means that the current job has file X open and locked for modification, and file Y not open, but locked for destroying.

## LOG

MTS Command Description

| | |
|---|---|
| Purpose: | To record the input/output activity of a file or device. |
| Prototype: | LOG [FDname1] {[ON] FDname2 [format] [options] \| OFF} |

FDname1

"FDname1" is the name of the file or device that is the source for the logging operation, i.e., the information that is to be logged. "FDname1" must be a simple file or device name (no line-number ranges or concatenations of files or devices is allowed). The default is *MSINK*.

FDname2

"FDname2" is the name of the file or device that is to receive the logged information. "FDname2" may not be a concatenation of files or devices, but may contain line-number ranges or modifiers.

format

"format" may be one of the following:

| | |
|---|---|
| ASIS | print log output in the same format as it appears in the session |
| SYMBOLIC | print log output in keyword format (provides more extensive information that may be useful for debugging programs that perform direct I/O to files) |

The default is ASIS.

options

"options" may be one or more of the options given below.

| | |
|---|---|
| READS | log all read operations (input) |
| WRITES | log all write operations (output) |
| CONTROL | log all control operations |
| BINARY | log @BIN I/O operations |
| ALL | log all operations |

The default is READS WRITES for ASIS format and READS WRITES CONTROL for SYMBOLIC format.

Program Key: *MTS.LOG

Description: The LOG command may be used to record the input/output activity of a file or device. The most common use of this command is to obtain a hard copy of a terminal session from a display-screen terminal. Normally, this will be done in the following manner:

```
LOG ON -LOG
    .
    .
  (terminal session)
    .
    .
LOG OFF
COPY -LOG *PRINT*
```

Note that the logged output should be first saved in a separate log file and then copied to *PRINT* at the end of terminal session so as not to interfere with any use of *PRINT* during the terminal session.

Several files or devices may be logged at one time as long as the following conditions are met:

(1) No FDname can be the source or destination for more than one logging operation.

(2) A logging source cannot also be used as a logging destination.

(3) A logging destination cannot also be used as a logging source.

The following is an example of a legal combination of LOG commands.

```
LOG ON -LOG
LOG -X ON -Y
```

The following are examples of illegal combinations of LOG commands.

```
LOG ON -Y
LOG -X ON -Y

LOG -X ON -Y
LOG ON -X

LOG ON -LOG
LOG -LOG ON -Y
```

Warning: If the user is logging terminal I/O and either LISTs the log file or EDITs the log file and issues the “PRINT /F” edit command, the list is potentially infinite, since each line that is listed or printed is also added to the end of the log file. If this occurs, the user should issue an attention interrupt.

For display-screen terminals and microcomputers, the logged output will contain both the input and output. For batch jobs, the logged output will contain only the job output. To log the job input, issue the command

LOG *MSOURCE* ON FDname

The DISPLAY LOGSTATUS command may be given to display information about what is being currently logged.

## MAKE

MTS Command Description

| | |
|---|---|
| Purpose: | To keep interrelated files in a consistent state. |
| Prototype: | MAKE program [WITH] options |
| Program Key: | *MTS.MAKE |
| Description: | The MAKE command provides a facility to keep interrelated files in a consistent state. Specifically, when one file is produced from another by some operation, then the output file depends on the input file. Usually, changes in the input file should be reflected in the output file. The most common example of such a dependency is between source and object files for some program. |
| | The MAKE command is described in *The MAKE Command*, Reference R1072. |

## MESSAGE

MTS Command Description

| | |
|---|---|
| Purpose: | To invoke the MTS Message System. |
| Prototype: | MESSAGE [message-command] |

message-command

"message-command" is an optional message-system command. The command is executed as a single command and an immediate return is made to the caller.

| | |
|---|---|
| Program Key: | *MESSAGESYST |
| Description: | The MTS Message System may be used to send messages to other users. For further details, see *MTS Volume 23: Messaging and Conferencing in MTS*, Reference R1023. |

## MODIFY

MTS Command Description

| | |
|---|---|
| Purpose: | To alter the contents of a general register, floating-point register, or specified virtual memory location(s). |
| Prototype: | MODIFY location value **... ...** |
| | This command is identical to the ALTER command. |
| Program Key: | *MTS.MODIFY |

## MOUNT

MTS Command Description

Purpose: To mount magnetic tapes or UMnet/Michnet Computer Network connections.

Prototype: MOUNT [request [;request] ...]

request

"request" is the mount request for the item to be mounted. One or more requests separated by semicolons may be entered directly on the MOUNT command; if the requests are omitted from the command line, they must be entered as separate input lines following the command (read from *SOURCE*) until terminated by an end-of-file.

The form of the request for mounting 9-track magnetic tapes is:

```
tapename [*pdn* [keywords]]
```

The form of the request for mounting a UMnet/Michnet Network connection is:

```
MNET *pdn* [keywords]
```

Program Key: *MTS.MOUNT

Description: For mounting magnetic tapes, the user must specify the tape name. When a tape is submitted to ITD, a tape name is assigned (the receipt code is used as the tape name by default; this name may be changed by the user to a more meaningful name). If a magnetic pool tape is to be mounted, the word POOL must be used in place of the tape name.

For UMnet/Michnet connections, the MNET device type must be specified.

For magnetic tapes, the pseudodevice name (*pdn*) defaults to *T*, but other names may be specified. For UMnet/Michnet connections, a pseudodevice name must be specified. After the item is mounted, all command and program references to the item are made using its pseudodevice name. The pseudodevice name is used in the same context as a file or device name "FDname" (see "Pseudodevice Names" in the section "Files and Devices" in this volume for further details on specifying and using pseudodevice names).

The DISPLAY PDNS command may be used to display the pseudodevice names for all currently mounted items.

The user may specify many different keyword parameters as part of the mount request. Successive keyword parameters may be entered in any order and must be separated by blanks or commas. For a complete description of the effect of each of the keyword parameters, see the appropriate sections in *MTS Volume 19: Magnetic Tapes in MTS*, Reference R1019. The tables on the following pages summarize the keyword parameters currently available. The default values are underlined where appropriate.

Magnetic tapes and UMnet/Michnet Computer Network connections may also be mounted from a program by calling the MOUNT subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Magnetic tapes and UMnet/Michnet connections may be dismounted by specifying

RELEASE *pdn*

where "*pdn*" is the pseudodevice name of the mounted volume.

Examples:

```
MOUNT
MYTAPE *OUTPUT* WRITE=YES
POOL *POOL*
$ENDFILE
```

The labeled magnetic tape with tape name MYTAPE is mounted on a 9-track magnetic tape unit; the tape is mounted with the file protect ring in so that it may be written on (WRITE=YES); the tape is assigned the pseudodevice name *OUTPUT* (if *OUTPUT* were omitted, the default pseudodevice name *T* would be used instead). A pool tape is also mounted on a 9-track tape unit; it is assigned the pseudodevice name *POOL*.

```
MOUNT MNET *WSU* D=WU
```

A Merit connection at Wayne State University is mounted and assigned the pseudodevice name *WSU*.

Keyword Parameter Summary

For magnetic tapes:

| | |
|---|---|
| BLKPFX=n | Specify block-prefix length for ANSI labeled tapes (0≤n≤99); defaults to 0 for initialized tapes |
| {BLOCKING \| BLK}={ON \| OFF} | Enable or disable blocking; defaults to ON |
| CC={A \| M} | Specify nonblank control character for first new file written |
| {DSNAME \| DSN \| NAME}=name | Specify file name to be used for next new file written |
| DTCHK={ON \| OFF} | Enable or disable expiration date checking; defaults to ON |
| EXPDT=[{mm-dd-yy \| mm/dd/yy}] | Specify expiration date for new files |
| {FORMAT \| FMT \| RECFM}=fmt[([size][,lrecl])] where "fmt" is {U \| F \| FB \| FBS \| V \| VB \| VBS \| D \| DB} | Specify blocking format and, optionally, block size and/or logical record length; defaults to U(255, 255) for unlabeled tapes |
| INIT={YES \| NO} | Initialize labeled tape; defaults to NO |
| LBLTYPE={MTS \| OS/VS \| VLO \| ANSI} | Specify label type; defaults to MTS |
| LP={ON \| OFF} | Enable or disable label processing; defaults ON for labeled tapes, OFF otherwise |
| LRECL=n | Specify logical record length (1 ≤ n ≤ 32767); defaults to 255 for unlabeled tapes |
| MINSIZE=n | Specify minimum block size (1 ≤ n ≤ 100) |
| MODE=mode | Specify tape mode; defaults to the current mode |
| POSN={* \| *n* \| *EOT* \| name} | Position tape to beginning of current file, *n*th file, end-of-tape, or data set name |
| QUIT={YES \| NO} | Control termination of batch job if mount fails; defaults to YES |
| RETRY=n | Specify retry count for read errors (0 ≤ n ≤ 15); defaults to 10 |
| RING={IN \| OUT} | Specify placement of file-protect ring; defaults to OUT |

| | |
|---|---|
| {SIZE \| BLKSIZE}=n | Specify maximum block size (18 ≤ n ≤ 32767); defaults to 255 for unlabeled tapes |
| TIMER={ON \| OFF \| n} | Enable, disable, or specify elapsed-time interval (in minutes) for inactive tape warning message; defaults to ON, n=15 minutes |
| TRANSLATE={MTS(parity) \| MTS \| IBM \| NONE} | Specify translation scheme for ANSI-labeled or ASCII unlabeled tapes; parity is EVEN, ODD, ZERO, or ONE; defaults to MTS(ZERO) |
| {VOLUME \| VOL \| LABEL \| VOLSER}={name \| 'name'} | Specify volume name of labeled tape |
| WAIT={YES \| NO \| PROMPT} | Control terminal-user waiting for queuing of 9TP requests if no tape drive available; defaults to PROMPT |
| WRITE={YES \| NO} | Specify placement of file-protect ring; defaults to NO |

For UMnet/Michnet Computer Network connections:

| | |
|---|---|
| {DEST \| D}={MS \| UM \| WU} | Specify connection destination; defaults to UM |
| QUIT={YES \| NO} | Control termination of batch job if mount fails; defaults to YES |

## MTS

### MTS Command Description

Purpose: To return to MTS command mode.

Prototype: MTS

Program Key: *MTS.MTS

Description: If this command is executed from MTS command mode, a message is printed stating that the user is already in MTS command mode.

If this command is executed from another command-language mode, an explicit return is made to MTS command mode. This differs from issuing the command-language MTS command whereby a return is made to the caller. For example, if the MTS File Editor were invoked by another program such as the MTS Message System, entering the edit command

```
MTS
```

would return the user to the Message System. However entering the MTS command

```
$MTS
```

will return the user to MTS command mode.

## NET

MTS Command Description

Purpose: To enter network mode.

Prototype: NET [{hostID | *pdn*}] [.network-command}

hostID

"hostID" specifies the two-character identifier of the remote host with which the user wishes to communicate. The hosts currently available are:

| | |
|---|---|
| UB | The University of Michigan (UB-MTS system) |
| UM | The University of Michigan (UM-MTS system) |
| WU | Wayne State University |

*pdn*

"*pdn*" is the pseudodevice name of a previously acquired UMnet/Michnet Network connection. Network connections may be acquired via the MOUNT command or the MOUNT subroutine.

If no hostID or pseudodevice name is given, the user is prompted. If the user previously left network command mode by entering the MTS or RETURN network commands, the previous connection is retained unless a new hostID or pseudodevice name is specified.

network-command

".network-command" is an optional network command. The period (.) prefix must be given with the command. The command is executed as a single command and an immediate return is made to the caller.

Program Key: *NET

Description: In network mode, the user's terminal communicates with any of the host computers interconnected by the UMnet/Michnet Computer Network. Input from the terminal is transmitted by MTS to the remote computer; output from the remote computer is printed at the terminal. In this manner, the user's terminal functions similarly to a terminal dialed directly to the remote computer. However, several additional facilities are available via network mode that would not otherwise be possible, e.g., the user may transfer files between the several UMnet/Michnet host computers.

The first character of each input line from the user's terminal is examined for the current network-mode selection character, normally a period (.). If found, the input line is taken as a network command instead of a line to be transmitted to the remote host. Network commands allow the user, for example, to return to MTS command mode (the MTS or RETURN command), to transfer the contents of a file to a file at the remote host (the COPY command), etc.

Information about the currently active UMnet/Michnet connections may be displayed by issuing the command

```
DISPLAY PDNS
```

For a complete description of how to use network mode with the UMnet/Michnet Computer Network, see the section "The NET Command" in *MTS Volume 4: Terminals and Networks in MTS*, Reference R1004.

Examples:

```
NET WU
```

A network connection to Wayne State University is established and network command mode is entered.

```
NET *W*
```

Network command mode is entered for the pseudodevice *W*.

## PERMIT

MTS Command Description

Purpose: To allow other userIDs or groups of userIDs to access files belonging to the user or to which the user has been given permit access.

Prototype: PERMIT filelist [access [accessor]]

PERMIT filelist LIKE filelist2 [EXCEPT access [accessor]]

filelist

“filelist” may be a single file name, a file-name pattern, or a parenthesized list of either, each of which is to be permitted identically. List items must be separated by blanks and/or commas.

access [accessor]

“access” specifies the access type to be allowed and “accessor” specifies who is to have the access. Several access-accessor pairs may be specified in the form

access [accessor] [,access [accessor]] **...**

“access” may be a single access type or a parenthesized list of access types. If not specified, READ is assumed. The six classes of access types are:

| *Category* | *Name* |
|---|---|
| Read | READ |
| Write-expand | WRITEXP, WE |
| Write-change and empty | WRITCHG, WC, EMPTY |
| Truncate and renumber | TRUNCATE, RENUMBER, RNU |
| Destroy and rename | DESTROY, RENAME |
| Permit | PERMIT |

Names representing the combinations of different categories may also be given:

| *Combination* | *Name* |
|---|---|
| Write-change and write-expand | WRITE |
| Read and write-change | RWCHG |
| Read and write-expand | RWEXP |
| Read, write-change, and write-expand | RW |
| Everything except permit | FULL |
| Everything | UNLIM |
| Default access | DEFAULT |

| | |
|---|---|
| No access | NONE |
| Run only | RUN |
| Edit only | EDIT |

The names given for each of the above categories and combinations are equivalent.

"accessor" is a single item or parenthesized list of items specifying the userIDs, project numbers, and program keys being given access to the file. If not specified, OTHERS is assumed.

| | |
|---|---|
| ALL | Specifies all userIDs except the owner of the file. All previous access information for this file is discarded. |
| ME | Specifies the userID issuing the PERMIT command. |
| OTHERS | Specifies all userIDs not otherwise explicitly mentioned in the accessor information. |
| OWNER | Specifies the owner of the file. |
| xxxx | Specifies a specific userID. |
| CCID=xxxx | Specifies a specific userID. |
| ID=xxxx | Specifies a specific userID. |
| PROJECT=xxxx | Specifies a projectID. |
| PKEY=key | Specifies a program key. |

A specific userID or projectID and a program key may be specified as

```
xxxx&PKEY=key
ID=xxxx&PKEY=key
PROJECT=xxxx&PKEY=key
```

filelist2

"filelist2" may be a single file name, a file-name pattern, or a parenthesized list of either whose access information is to be copied to the file specified by "filelist". If "filelist2" is a list, the copying proceeds left to right.

All previous access to the files in "filelist" is removed before the new access is applied. If the file being permitted has a different owner than the LIKE-file, the owner of the LIKE-file is made an explicit sharer of the file being permitted.

Program Key: *MTS.PERMIT

Description: For a general description of the PERMIT command and the meanings of the various parameters, see "Shared Files" in the section "Files and Devices" in this volume.

If "filelist" is a pattern, a set of files is permitted according to the access/accessor specifications. If "filelist2" is a pattern, the LIKE-names are generated by replacing each question mark in "filelist2" with the string matched by the corresponding question mark in "filelist". "filelist" must have at least as many question marks as "filelist2". For example

```
PERMIT ?.NEW LIKE ?.S
```

is equivalent to

```
PERMIT A.NEW LIKE A.S
PERMIT B.NEW LIKE B.S
```

if ?.NEW matches A.NEW and B.NEW.

If only "?" is specified for "filelist", a special confirmation request

```
Do you really want to permit ALL your files?
```

is printed. If the reply is OK, the files are permitted.

The user must have PERMIT access to the files.

Verification of the new file-access information is printed.

A file may be permitted from a program by calling the PERMIT subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Further information about the syntax of the PERMIT command and the use of the PERMIT command with program keys is given in the section "Files and Devices" in this volume.

Examples:

```
PERMIT X
```

Read access is given to OTHERS for the file X.

```
PERMIT Y WRITE WABC
```

Write-change and write-expand access is given to the userID WABC for the file Y.

```
PERMIT Z READ W?
```

Read access is given all userIDs that begin with the character W.

```
PERMIT (A,B,C) READ OTHERS
```

Read access is given to OTHERS for the files A, B, and C.

```
PERMIT X LIKE Y
```

The access list of file Y is applied to file X.

```
PERMIT DATA?  RW WXYZ
```

Read-write access is given to the userID WXYZ for all files beginning with the string DATA.

```
PERMIT OBJ RUN OTHERS
```

Run-only access is given to OTHERS for the program in the file OBJ.

```
PERMIT TEXT EDIT ME
```

Edit-only access is given to the owner for the file TEXT. This prevents the owner from inadvertently damaging the contents of the file.

## RELEASE

MTS Command Description

Purpose: To release a pseudodevice.

Prototype: RELEASE {*PRINT* | *PUNCH* | *BATCH* | *pdn*}

Program Key: *MTS.RELEASE

Description: If the parameter specifies *PRINT*, *PUNCH*, or *BATCH*, the associated job is released to the system for processing. This is the same as if the command

CONTROL *...* RELEASE

were given.

If the parameter specifies a pseudodevice name (*pdn*) previously mounted by the MOUNT command or the MOUNT subroutine, that device is dismounted.

Users should be aware that *PRINT*, *PUNCH*, *BATCH*, or *pdn* are never actually released until all outstanding references to the parameter are terminated; at that time, a message is printed indicating that the parameter has been released. If no message is printed immediately following the RELEASE command, the user still has an outstanding reference to the specified parameter (e.g., a program is currently loaded which refers to the parameter). When all outstanding references are finally terminated (e.g., by unloading the program), the specified parameter is released and the message is printed.

Information about the currently active *...* jobs or pseudodevices may be displayed by issuing the command

DISPLAY PDNS

Examples:

```
RELEASE *PRINT*
```

*PRINT* is released.

```
RELEASE *TAPE*
```

The pseudodevice *TAPE* is released.

## RENAME

MTS Command Description

Purpose: To rename a file or a set of files.

Prototype: RENAME oldname [AS] newname [{OK | ALLOK | PROMPT}]

Program Key: *MTS.RENAME

Description: The “oldname” parameter specifies the file or the set of files to be renamed and may be a single file name or a file-name pattern. The “newname” parameter specifies the new names for the files being renamed.

If “oldname” is a pattern, a set of files are renamed. If “newname” is a pattern, the new names are generated by replacing each question mark in “newname” with the string matched by the corresponding question mark in “oldname”. “oldname” must have at least as many question marks as “newname”. “newname” must be a pattern if “oldname” matches more than one file. For example,

```
RENAME ?.S ?.OLD
```

is equivalent to

```
RENAME A.S A.OLD
RENAME B.S B.OLD
```

if ?.S matches the files A.S and B.S.

If a single file name is specified, confirmation is requested in conversational mode. If more than one file is specified, a single summary confirmation is requested. If the reply is OK, then all the files specified are renamed; otherwise, none are renamed. Confirmation is not requested in batch mode.

When only a single file name is specified, the OK option may be used to bypass the confirmation request; the OK option is ignored if more than one file name is specified. The ALLOK option may be used to bypass the confirmation request when several files have been specified. The PROMPT option causes prompting for confirmation for each individual file.

The response to a prompt for confirmation may be OK to rename the file, NO to skip the file but continue with the next file in the list, or CANCEL to terminate the command.

The user must have RENAME access to the file (or files) given by “oldname”, and “newname” must specify a file belonging to either the same userID as “oldname” or the current userID. If “newname” is a pattern, it must specify a group of files belonging to the same userID as “oldname” or the current userID.

Permanent files may be renamed to temporary files, and temporary files may be renamed to permanent files if the user's file space allocation allows it.

A file or a set of files may be renamed from a program by calling the RENAME subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Examples: The following examples all assume that user WABC is signed on:

```
RENAME DATA1 AS NEWDATA1
```

The file DATA1 is renamed to NEWDATA1. The terminal user is prompted for confirmation.

```
RENAME DATA2 NEWDATA2 OK
```

The file DATA2 is renamed to NEWDATA2. Confirmation is given on the command.

```
RENAME DATA?  NEWDATA?  ALLOK
```

All files beginning with the string DATA are renamed to corresponding files beginning with the string NEWDATA. Confirmation is given on the command.

```
RENAME WDEF:PROG AS STATPROG
```

The file WDEF:PROG is renamed as STATPROG. In the process, the owner of the file changes from WDEF to WABC. The command will fail if WABC does not have sufficient disk space. After the command is executed, WDEF will still have the same access as it previously had (by default, UNLIMITED). WABC may subsequently change or remove that access.

```
RENAME -DATA AS DATA
```

The file –DATA will be changed to a permanent file. The command will fail if WABC does not have sufficient disk space.

## RENUMBER

MTS Command Description

Purpose: To renumber a line file (or subset thereof).

Prototype: RENUMBER filelist [first [last [beg [inc]]]] [{ALLOK | PROMPT}]

RENUMBER filelist [[[first], last] [TO [beg], inc]]
[{ALLOK | PROMPT}]

filelist

“filelist” specifies the file or the set of files to be destroyed and may be a single file name, a file-name pattern, or a parenthesized list of either.

first, last

“first” and “last” refer to the first and last line numbers in the range to be renumbered. If omitted, “first” and “last” default to the actual first and last line numbers in the file, respectively.

beg, inc

“beg” and “inc” refer to the new beginning line number and the increment to be used in the renumbering. If omitted, “beg” and “inc” default to 1.

If a single file name is specified, no confirmation is requested. If more than one file is specified, a single summary confirmation is requested. If the reply is OK, then all the files specified are renumbered; otherwise, none are renumbered. Confirmation is not requested in batch mode.

The ALLOK option may be used to bypass the confirmation request when several files have been specified. The PROMPT option causes prompting for confirmation for each individual file.

Program Key: *MTS.RENUMBE

Description: An error comment is produced if the file does not exist or renumber access is not allowed. The renumbering is not attempted if the process would result in duplicate line numbers or line numbers out of sequence in the file.

A file may be renumbered from a program by calling the subroutines RENUMB, RETLNR, CNTLNR, or SETLNR (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Examples:

```
RENU FILE1
RENU FILE1 *F,*L
RENU FILE1 *F *L 1
RENU FILE1 *F *L 1 1
```

In each of the above examples, the file FILE1 is renumbered starting at line 1 and in increments of 1.

```
RENU DATA?
```

In the above example, all files beginning with the characters “DATA” are renumbered.

## RERUN

MTS Command Description

Purpose: To reissue the previous RUN command.

Prototype: RERUN [{ECHO | NOECHO}] [I/Ounits] [options]

{ECHO | NOECHO}

If ECHO is specified, the text of the reissued command is echoed on the user's sink file or device. ECHO is the default if *SOURCE* and *MSOURCE* are the same (the default) or if the ECHO option of the SET command is ON (the default).

[I/Ounits] [options]

Any of the parameters that are valid for the RUN command may be given. If a logical I/O unit or another option is given without a right-hand side, that unit or option setting is removed from the next invocation of the program.

Program Key: *MTS.RUN valid for the RUN command.

Description: The previous RUN or RERUN command is reissued (as originally entered) with any parameters specified on this invocation overriding the corresponding parameters of the previous invocation. The previous program is unloaded (if still loaded) and a new copy of the program is loaded.

Only the RUN and RERUN commands are used as the base for the RERUN command; any invocations of the DEBUG, LOAD, RESTART, and START commands are ignored.

The text of the last RUN or RERUN command may be displayed by issuing the command

DISPLAY RERUN

Examples: If the command

```
RUN PROG INPUT=FILE1
```

is issued, then RERUN reruns the program PROG exactly as before. If the command

```
 RERUN PRINT=FILE2
```

is issued, then the program is rerun with the file FILE2 assigned to PRINT; this is equivalent to entering the RUN command as

```
RUN PROG INPUT=FILE1 PRINT=FILE2
```

If the command

```
RERUN INPUT=FILE3
```

is subsequently issued, then the program is rerun with the file FILE3 assigned to INPUT instead of the file FILE1; this is equivalent to entering the RUN command as

```
RUN PROG INPUT=FILE3 PRINT=FILE2
```

If the command

```
RERUN PRINT=
```

is subsequently issued, then the program is rerun with the unit PRINT unassigned; this is equivalent to issuing the command

```
RUN PROG INPUT=FILE3
```

## RESTART

MTS Command Description

Purpose: To restart (or initiate) execution of a program following either initial loading, an interrupt, or a subroutine call to ERROR, MTS, or MTSCMD.

Prototype: RESTART [[AT] location] [I/Ounits] [limits] [mapoptions]
[{EXECPKEY | PKEY}={key | OFF}]

The following parameters may be given:

[AT] location

"location" specifies the address at which execution is to begin. "location" is a virtual memory location given by an *optional* local relocation factor (RF) and a displacement in the form

[RF={hhhhhh | GRx}] xxxxxx

where "hhhhhh" is the hexadecimal value of a local relocation factor; GRx indicates the general register whose contents are used as a local relocation factor, and "xxxxxx" is the hexadecimal value of a displacement. The displacement is added to the current value of the relocation factor to provide an absolute virtual memory address. If a local relocation factor is not specified, the global relocation factor is used. The global relocation factor is initially zero, but may be changed by the RF option of the SET command or by calling the subroutine CUINFO. Since this value replaces the righthand 32 bits of the PSW, "location" specifies the instruction length code, the condition code, and the program mask as well as the address.

If "location" is specified and the currently loaded program has a nondefault program key that is not prefixed by the current userID, the program key will be set to the default (for this invocation of the program only).

This command is invalid if the currently loaded program is a "run-only" program.

I/Ounits

The user can reassign the logical I/O units. See the RUN command description.

FORTRAN users are reminded that MTS logical I/O units 0 through 99 are not necessarily the same as the FORTRAN logical I/O units 0 through 99.

limits

The user can respecify local limits for CPU time, pages printed, and cards punched. If no limits are specified, the remaining amounts in any limits specified on the previous RUN, RERUN, START, RESTART, LOAD, or

DEBUG commands are used. For a complete description of the use of limits, see the RUN command description and the section “UserIDs, Limits, and Sigfiles” in this volume.

MAP[=mapFDname] [NOMAP] [XREF] [UXREF]

The user can redesignate the destination of future loader maps with the MAP parameter. This is useful if the program is loading other modules dynamically by calling the subroutines LOAD, LINK, or XCTL. The NOMAP parameter can be used to suppress the printing of all future loader maps. See the RUN command description.

{EXECPKEY | PKEY}={key | OFF}

The EXECPKEY parameter “key” specifies an override program key to be used instead of the program key associated with “program”. If OFF is specified, the override specified by the EXECPKEY option of the SET command is disabled. For further details on the use of program keys, see the section “Files and Devices” in this volume.

Program Key: *MTS.RESTART

Description: The RESTART command restarts (or initiates) execution of the currently loaded program. This, for example, may be a program loaded by the LOAD command, a program that was interrupted during execution by a program or attention interrupt, or a program that terminated by a call to ERROR, MTS, or MTSCMD.

A 32-bit value is computed from the “location” specification and replaces the righthand 32 bits of the PSW. If “location” was omitted, the PSW remains unaltered, and execution begins at the entry point for a program loaded by the LOAD command, or the point of interruption for a program that was interrupted by a program or attention interrupt; for a program that returned by a call to ERROR, MTS, or MTSCMD, execution restarts at the point following the subroutine call as if the subroutine had returned to the program.

If logical I/O units have been reassigned, the files and devices originally assigned are released, and the newly assigned files and devices are attached.

Examples:

```
RESTART PRINT=A
```

This restarts the currently loaded program with PRINT output reassigned to the file A.

```
RES AT RF=820800 28000258
```

The program is restarted at location 820A58 with the condition code set to 2, fixed-point overflow interruption enabled, and the other program mask interrupts disabled.

## RUN

MTS Command Description

Purpose: To load and execute a program.

Prototype:

```
RUN [program] [I/Ounits] [limits] [mapoptions]
    [{EXECPKEY | PKEY}={key | OFF}] [PAR=parameters]
```

The following parameters may be given:

program

"program" specifies the file or device containing the program to be loaded. If omitted, the program is loaded from *SOURCE*.

I/Ounits

The keyword parameters "I/Ounits" are the assignments of logical I/O units to files or devices for use by the loaded program during execution. The logical I/O unit assignments are used to select appropriate I/O subroutines to be used by the loaded program for the input and output of data. Where no specifications are stated, the following default assignments occur:

```
SCARDS=*SOURCE*
GUSER=*MSOURCE*
SERCOM=*MSINK*
SPRINT=*SINK*
SPUNCH=*PUNCH*        (batch mode if global card est. > 0)
```

Synonyms may be used for the following logical I/O unit names:

```
INPUT for SCARDS
PRINT for SPRINT
OBJECT for SPUNCH
```

The logical I/O units 0 through 99 have no default specifications. See the section "Files and Devices" in this volume and the subroutine descriptions for INPUT (SCARDS), PRINT (SPRINT), OBJECT (SPUNCH), SERCOM, GUSER, READ, and WRITE in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for further details on the use of these subroutines.

FORTRAN users are reminded that MTS logical I/O units 0 through 99 are not necessarily the same as the FORTRAN logical I/O units 0 through 99.

limits

The keyword parameters "limits" specify local limits for CPU time, pages printed, cards punched, and plot time. These can be given in the form:

TIME={t | tS | tM}
PAGES=p
CARDS=c
{PLOTTIME | PT}={t | tM}

These local limits can be used in both batch and conversational mode and are effective only for the execution of the program loaded by the RUN command. For a complete description of the use of limits, see the section "UserIDs, Limits, and Sigfiles" in this volume.

MAP[=mapFDname] [NOMAP] [XREF] [UXREF]

The MAP parameter specifies the file or device "mapFDname" on which the loader is to write the loader map. If the MAP parameter is omitted, no loader map is written. If the MAP parameter is given in the form of MAP, "mapFDname" defaults to *SINK*. The NOMAP parameter suppresses the printing of the loader map. NOMAP need be given only if a previous MAP specification was made.

The XREF parameter specifies that a cross-reference listing of external symbols occurring in the loaded programs is to be produced. If XREF is given without the MAP parameter, MAP=*SINK* is assumed. The XREF parameter implies the UXREF parameter. (See also the SET XREF option.)

The UXREF parameter specifies that a cross-reference listing of undefined external symbols occurring in the loaded programs is to be produced. If UXREF is given without the MAP parameter, MAP=*SINK* is assumed. (See also the SET UXREF option.)

{EXECPKEY | PKEY}={key | OFF}

The EXECPKEY parameter "key" specifies an override program key to be used instead of the program key associated with "program". If OFF is specified, the override specified by the EXECPKEY option of the SET command is disabled. For further details on the use of program keys, see the section "Files and Devices" in this volume.

PAR=parameters

The PAR field specifies an arbitrary string of characters to be passed to the loaded program on initiation of execution. This is usually a parameter list for the program and its interpretation depends on the loaded program. The PAR field must be the last parameter specified in the command. The parameter list is terminated by the end of the command line. Note that the parameter field always has a blank added after the last character and the length count is incremented by one.

Program Key: *MTS.RUN

Description: The RUN command invokes the dynamic loader to load the object program into virtual memory. If there are unresolved external symbol references after loading from “program”, loading continues from *LIBRARY (the system library). Only those parts of *LIBRARY required to resolve the references are loaded. If there are still unresolved external references, a fatal loading error exists. In conversational mode, the loader prompts for more loader input; in batch mode, an error comment is produced and the loading terminates immediately. The search of the system library may be modified or suppressed by the LIBR, LIBSRCH, or *LIBRARY options of the SET command. If SET DEBUG=ON has been specified, SDS processes any symbol table information associated with the loaded program. For a complete description of the loading process, see the section “The Dynamic Loader” in *MTS Volume 5: System Services*, Reference R1005.

If there are no fatal loading errors, control is transferred to the entry point of the program by a standard subroutine call (the entry point address in GR15, the return address in GR14, the save area location in GR13 and the parameter list location in GR1).

The parameter string (specified by the PAR field) is passed as follows: GR1 contains the location of a fullword address constant which points to a region containing a halfword count (halfword aligned) followed by an EBCDIC character region (of byte-length specified by the count) containing the parameter string. The leftmost bit of the address constant is 1.

If a file or device specified by “I/Ounits” is nonexistent or is not available, the logical I/O unit referring to the unavailable file or device is set up such that the first time the program being executed refers to the logical I/O unit, either the user is given an opportunity to respecify the FDname (in conversational mode) or execution is terminated (in batch mode).

If “program” has a nondefault program key that is not prefixed by the current userID, the program key will be set to the default (for this invocation of the program only) in any of the following cases:

(1) “program” includes explicit concatenation, line-number ranges, and/or I/O FDname modifiers,

(2) a loading error that requires prompting for additional input occurs, or

(3) the shared-file separator character has been set to something other than a colon (:).

In addition, the effect of the LIBSRCH option will be ignored.

If “program” is a “run-only” program, the program key used by the system during the loading process, normally *MTS.RUN, is changed to *MTS.ETC.RUN in the following cases:

(1) “program” includes explicit concatenation, line-number ranges, and/or I/O FDname modifiers,

(2) if implicit concatenation is encountered during the loading process,

or

(3) the shared-file separator character has been set to something other than a colon (:).

In addition, the effect of the LIBSRCH option will be ignored. And finally, if a loading error occurs that requires prompting or additional input, the loading process will be prematurely terminated.

If the program terminates execution by restoring the registers and returning to MTS via GR14 or by calling the SYSTEM subroutine, the program is unloaded.

All storage, files, and devices used for this RUN command are automatically released unless execution was not terminated normally (e.g., the program calls the ERROR subroutine, or an attention or program interrupt has occurred).

If storage, files, and devices are not released, the user can use debug mode or the DISPLAY, ALTER, and RESTART commands to debug or continue the program.

The return code for the program is initially set to −1 (undefined) when execution is started. If the program terminates execution by returning to MTS, the return code is set to the contents of general register 15. If the program terminates execution by calling the subroutine SYSTEM, the return code is set to zero; if the program terminates by calling the subroutine ERROR, the return code is set to 8.

Examples:

```
RUN -LOAD
```

This loads and initiates execution of the program in the temporary file −LOAD which could, for example, contain the object module from a FORTRAN compilation.

```
RUN OBJPROG+*PLOTSYS MAP 5=INPUT 6=OUTPUT
```

This loads and initiates execution of the program in OBJPROG which contains references to subroutines in *PLOTSYS. A load map is printed on *SINK*. Logical I/O units 5 and 6 are specified for the input and output of data to the program.

```
RUN *C87 T=2
```

This loads and initiates execution of the C87 compiler. A local time estimate of 2 seconds is specified.

```
RUN *ASMH INPUT=SOU OBJECT=OBJ 2=MACROS PAR=TEST,ESD,RLD
```

This loads the 360/370 assembler *ASMH and initiates execution of the assembler with source input from the file SOU and object module output to the file OBJ. The macro library MACROS is attached to logical I/O unit 2. The parameter string in the PAR field is passed to the assembler by the RUN command.

## SDS

MTS Command Description

| | |
|---|---|
| Purpose: | To enter or return to debug mode. |
| Prototype: | SDS [debug-command] |
| Program Key: | *SDS |

Description: Control is transferred to debug command mode. In debug mode, the user has the facilities of the symbolic debugging system (SDS) at his disposal. For further details, see *MTS Volume 13: The Symbolic Debugging System*, Reference R1013.

If a debug command is specified with the SDS command, that debug command is executed, and control is returned immediately to the caller (normally MTS command mode).

If the currently loaded program has a nondefault program key that is not prefixed by the current userID, the program key will be set to the default value (for this invocation of the program only).

Examples:

```
SDS
```

Control is transferred to debug mode.

```
SDS SET ERRORDUMP=ON
```

The automatic errordumping option of SDS is enabled; control is returned to MTS command mode.

## SET

MTS Command Description

| | |
|---|---|
| Purpose: | To set MTS global options. |
| Prototype: | SET option ... |

Any number of the following options may be given in a single SET command. The options must be separated by blanks; there must be no blanks within any option.

Unless otherwise stated, the values set by these options are in effect only until sign-off.

Almost all of the values set by these options may be interrogated by programs via the GUINFO subroutine and may be changed by programs via the CUINFO subroutine. See the subroutine description for GUINFO and CUINFO in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for details.

ADDRESS="line1;line2;..." Default: No address

Specify the campus mail address for delivered output (when DELIVERY=MAIL is set). This option applies only to page-printer output, not to line-printer or local-printer output.

AUTOHOLD={ON | OFF} Default: OFF

If the AUTOHOLD option is ON, any *PRINT*, *PUNCH*, or *BATCH* job is automatically held without notice when it is first used (at the time the receipt message is printed). If AUTOHOLD is OFF, the job is automatically released when finished. This may be locally overridden by the CONTROL command HOLD option.

CARDS=n Default: No limit

The CARDS option specifies a card limit for a punch job. It applies only to jobs directed to *PUNCH* or user-defined pseudodevices with TYPE=PUNCH.

CLASS=char Default: A

The CLASS option specifies a file class to associate with a job. If no file class is assigned to a file, a class of A will be used; otherwise, CLASS is a single letter from A to Z. If a class is assigned, *IMPORT* is restricted to only reading jobs of the assigned class. This option applies only to BITNET connections. For further information, see *BITNET in MTS*, Reference R1039.

CMDSCAN={AMBIGUOUS | UNAMBIGUOUS}
Default: UNAMBIGUOUS

This option affects the MTS command recognition procedure. If the CMDSCAN option is UNAMBIGUOUS, all abbreviations of MTS command verbs are required to be unambiguous. Furthermore, all command-verb text beyond the minimum required abbreviation to identify the command must be a substring of the full command text (e.g, SYSSTAT is not an acceptable abbreviation of SYSTEMSTATUS). If the user is in conversational mode and if the SPELLCOR option is PROMPT or ON, spelling correction of commands is attempted. If the CMDSCAN option is AMBIGUOUS, the MTS command recognition procedure uses the abbreviations given at the beginning of the section “MTS Command Mode.”

Because a few MTS commands have popular abbreviations which, although ambiguous, are harmless if erroneously specified, these commands retain their minimum one-character abbreviations when CMDSCAN is UNAMBIGUOUS. These commands are COPY, DISPLAY, LIST, and RUN. In addition, SIG is recognized both as SIGNON and SIGNOFF depending on the user’s signon status.

CMDSKIP={ON | OFF} Default: OFF

If the CMDSKIP option is ON, a page-skip is produced in batch automatically after any command that may generate printed output (LIST, COPY, RUN, etc.). If the CMDSKIP option is OFF, only a triple space is produced. The CMDSKIP keyword is ignored in conversational mode.

COMMENT="text" Default: No comment

The COMMENT option specifies a a comment to be printed on the head sheet of the job.

CONTCHAR=character Default: –

The CONTCHAR option specifies a single character to be used as an indicator for continuing an input line in MTS command mode on another input line. The continuation character must be the last character of the line to be continued.

COPIES=n Default: 1

The COPIES option specifies the default number of copies of printed output for any *PRINT* job. This may be locally overridden for a given job by the CONTROL command. This option is ignored for printed output routed via the UMnet/Michnet Computer Network. The COPIES keyword is ignored in batch mode.

COST={ON | OFF} Default: OFF

If the COST option is ON, the cost since the last cost was printed (not necessarily the cost of the last command) and the total current cost of the session are printed after every MTS command. These costs are approximate and subject to roundoff; items which are not immediately charged for, such as tape-drive time, are not included until the actual charge is made. If COST is OFF, these costs are not printed.

CROUTE=station Default: CNTR

The CROUTE option specifies the default destination for *PUNCH* output. The parameter “station” is the destination identification code to which the output is being sent. This may be locally overridden for a given job by the CONTROL command. The station specified may not be a UMnet/Michnet Computer Network host. The default is CNTR except for certain hardwired terminals located at remote batch stations equipped with card punches. The CROUTE keyword is ignored in batch mode.

DEBUG={ON | OFF} Default: OFF

If DEBUG is ON, all programs loaded are processed by SDS for any symbol table information associated with them. This allows debug mode commands to refer to symbols in programs compiled with the TEST option. Also, all programs run in execution mode are monitored by SDS and any errors produce debug mode error comments.

DELIVERY={station | MAIL | NONE} Default: NONE

The DELIVERY option specifies destination for output delivered by courier service. The parameter “station” is the destination identification code for the station to which output is being sent. The default is NONE which specifies that output is to remain at the station where it was printed. If MAIL is specified, the output will be delivered by campus mail or the US Postal Service. The DELIVERY keyword is ignored in batch mode. The delivery schedule is given in the public file *DELIVERY.

DESTINATION=userid@node Default: None

The DESTINATION option specifies a BITNET destination, e.g., userID@site, proute@site, or croute@site. A destination must be specified for all file transfers (from MTS) over BITNET. The destination specifies where the file is to be sent. The “node” part is the remote machine name; the “userid” part is the individual user on that machine.

The node name of UM-MTS is UMICHUM and UB-MTS is UMICHUB. The “userid” under MTS is your MTS userID preceded by the word USER. For example, if your MTS userID is ABCD, your “userid” for network purposes would be USERABCD.

This option applies only to BITNET connections. For further information, see *BITNET in MTS*, Reference R1039.

DISPATCHES={ON | OFF}        Default: ON
DISPATCHES(filter)={ON | OFF}

The DISPATCHES option specifies whether dispatch messages may be received. If the DISPATCHES option is OFF, dispatch messages are refused (not printed on the terminal). If the DISPATCHES option is ON, dispatch messages are printed immediately when received. See the DISPATCH command in the section “The Message System” of *MTS Volume 23: Messaging and Conferencing in MTS*, Reference R1023, for details on sending dispatch messages.

The use of dispatch filters to selectively filter dispatches from various types of users is also described in *MTS Volume 23*. The valid filters are VIEW, LOCAL, NETWORK, USER, SYSTEM, and ALL.

EBM=characters
ETM=characters

The EBM and ETM options control the format of the “execution begins” and “execution terminated” messages, respectively. “characters” consists of from 0 to 7 characters selected from the following sets:

For EBM and ETM:

| | |
|---|---|
| W | means print the words “Execution begins”, “Execution terminated”, or “Error return”. |
| H | means print the time of day in the form HH:MM:SS. |

For ETM only:

| | |
|---|---|
| T | means print the CPU time used to execute the program (does not include the time to load the program). |
| R | means print the return code produced by the program (subject to the RCPRINT option). |
| $ | means print the cost of executing the program (does not include the cost of loading the program). |

The defaults are

| | | |
|---|---|---|
| EBM=W | ETM=WR | (conversational mode) |
| EBM=WH | ETM=WHTR$ | (batch mode) |

ECHO={ON | OFF}        Default: ON

If the ECHO option is ON, MTS command lines are echoed to *SINK* if *SINK* differs from *SOURCE* and to *MSINK* if *MSINK* differs from both *SINK* and *SOURCE*. If ECHO is OFF, echoing is suppressed.

ENDFILE={ALWAYS | SOURCE | NEVER}
Default: SOURCE

If the ENDFILE option is ALWAYS, a $ENDFILE line is recognized as an end-of-file whenever it is read; if ENDFILE is SOURCE, a $ENDFILE line is recognized as an end-of-file only when read from *SOURCE* or *MSOURCE*; if ENDFILE is NEVER, a $ENDFILE line is never recognized as an end-of-file. This option may be overridden by the @ENDFILE and @~ENDFILE I/O FDname modifiers. The alternate form for this command is ENDFILE={ON | OFF | NEVER}, where ON and OFF are the same as ALWAYS and SOURCE, respectively.

ERRMAP={ON | OFF} Default: ON

If the ERRMAP option is ON, the loader map is printed if execution terminates abnormally in batch mode and if a map has not already been printed. If ERRMAP is OFF, the loader map is not printed. This option has no effect in conversational mode.

ERRORDUMP={NOLIBRARY | OFF | LIBRARY}
Default: OFF

If the ERRORDUMP option is NOLIBRARY and execution terminates abnormally in batch mode, a storage dump of the user's loaded program is given; if ERRORDUMP is LIBRARY, the storage dump includes any library subroutines loaded; if ERRORDUMP is OFF, no dump is given. NOLIBRARY is the same as LIBRARY if SYMTAB=OFF was specified before the program was loaded. If a global time or page limit is exceeded while producing a program dump, the dump is completed and the CPU time and page costs are charged to the user's account. This option has no effect in conversational mode. Note that this is *not* the same as the similar debug mode command which produces a symbolic dump. The alternate form for this command is ERRORDUMP={ON | OFF | FULL}. The ERRORDUMP keyword is ignored in conversational mode.

ERRORPROMPT={ON | OFF} Default: ON

If the ERRORPROMPT option is ON, the user is prompted for error correction for invalid files, commands, and keyword parameters.

{EXECPFX | EXECPREFIX}=character Default: blank

The EXECPFX option specifies the default prefix character to be used for executing programs.

EXECPKEY={key | OFF} Default: OFF

The EXECKEY option specifies the override program key to be used instead of the program key associated with the program object file for all subsequent program invocations. If EXECPKEY is OFF, an override program key is not in effect. For further details on programs keys, see the section "Files and Devices" in this volume.

FILE={filename | "file name"}                Default: Blanks

The FILE option specifies a file name to associate with the job. If the file name contains blanks or special characters, it must be enclosed in quotes. For *IMPORT*, this option can be used to select from a number of incoming jobs. This option applies only to BITNET connections. For further information, see *BITNET in MTS*, Reference R1039.

FORMAT={LANDSCAPE | PORTRAIT | TWOUP | format-name}
Default: LANDSCAPE

The FORMAT option specifies the format for page-printer output (see *Using the Xerox 9700 Page Printer*, Reference R1038, for a complete description and list of formats).

HELPMODE={LINE | DEFAULT | SCREEN}
Default: DEFAULT

The HELPMODE option specifies the format in which the user assistance information is to be displayed. If LINE is specified, the information is printed in line format. If SCREEN is specified, the information is printed in full-screen format. If DEFAULT is specified, the information is printed in full-screen format if the terminal supports full-screen output; otherwise, the information is printed in line format.

IC={ON | OFF}                Default: ON

If the IC option is ON, the line “$CONTINUE WITH” specifies implicit concatenation; if IC is OFF, this line is treated as a data line. The IC option can be overridden by the @IC modifier on I/O operations (see Appendix A to the section “Files and Devices” in this volume).

INITFILE(command)={FDname | OFF}     Default: OFF

The INITFILE option specifies a file to be read by the specified MTS command processor (such as the EDIT or DEBUG command) when it is initialized. “command” is the name of the command or any valid abbreviation of the command. The file “FDname” may contain initialization commands for the command processor to preset various options. Any command may be included that is recognized by the command processor. The initialization processing by a command processor may be disabled by specifying INITFILE(command)=OFF.

JOBNAME={jobname | DEFAULT}          Default: RMnnnnnn

The JOBNAME option specifies a job name of 1 to 8 alphanumeric characters for all subsequent jobs sent to the printer. The first character must be a letter. DEFAULT specifies the default format of “RM” plus six digits.

LIBR={ON | OFF} Default: ON

The LIBR option interacts with the LIBSRCH and *LIBRARY options. If LIBR is ON, all the library files specified by LIBSRCH, and the file *LIBRARY and LCSYMBOL (the low-core symbol directory) specified by the *LIBRARY option are searched for unresolved external symbols after a program is loaded; if LIBR is OFF, this search is not made.

LIBSRCH={OFF | FDname} Default: OFF

The LIBSRCH option indicates that specific public or private libraries are to be searched if there are unresolved symbols in a loaded program. This option interacts with the LIBR and *LIBRARY options. If LIBR is OFF, then the LIBSRCH option is ignored. If LIBR is ON, but LIBSRCH is OFF, then only *LIBRARY (the system library) and LCSYMBOL (the low-core symbol directory) are searched (if the *LIBRARY option is ON). If LIBR is ON and LIBSRCH is not OFF, then LIBSRCH specifies a library or concatenation of libraries to be searched. The libraries are searched in the order specified in the option. After these libraries are searched, *LIBRARY and LCSYMBOL are searched if there are still any unresolved symbols (if the *LIBRARY option is ON). If the program being loaded has a nondefault program key that is not prefixed by the current userID, the program is loaded as if LIBSRCH were OFF.

MACROS={ON | OFF} Default: ON

If the MACROS option is ON, the MTS command extensions and macro processor is enabled. If MACROS is OFF, the processor is not enabled.

MAILCALL={ON | OFF} Default: ON

The MAILCALL option specifies that conversational users will be notified immediately when an MTS Message System message is received.

MAPDOTS={ON | OFF} Default: ON

If the MAPDOTS option is ON, the dotted lines which delimit the loader map are printed whenever the loader map is printed. If MAPDOTS is OFF, the dotted lines are not printed.

MARGIN={n.nn | NO} Default: See text

The MARGIN option sets the left margin for page-printer output to “n.nn” inches. “n.nn” must be less than the current page width (8.5 for portrait orientation, 11.0 for landscape). MARGIN=NO turns off the margin override and resets the margins to the default for the current format (0.5 for PORTRAIT and 0.65 for LANDSCAPE).

NAME={name | 'name' | NONE}

The NAME option assigns an individual name to the current userID. This is useful for routing mail messages to the current userID. See *MTS*

*Volume 23: Messaging and Conferencing in MTS*, Reference R1023, for further details.

NAMELIB={FDname | OFF}

The NAMELIB option specifies a name-library to be used with the MTS Message System. This is useful for sending messages to group names. See *MTS Volume 23: Messaging and Conferencing in MTS*, Reference R1023, for further details.

{NEWFILEACCESS | NFA}={'string' | "string" | OFF}
Default: OFF

The NEWFILEACCESS option specifies the permit access to be given to all permanent files created by the CREATE command. The access string may be any access/accessor pair that is valid for the PERMIT command. If the NEWFILEACCESS option is OFF, only the access UNLIM OWNER is given.

NUMBER={(b,l,c) | NO} Default: NO

NUMBER=(b,l,c) numbers page-printer output pages automatically, starting with number “b”, printing the number on line “l”, ending in column “c”. NUMBER=NO (the default) disables automatic page-numbering. The page number is always printed in the first font of the current format or FONTLIST specification.

OVERLAY={NONE | SHADED | LINED} Default: NONE

The OVERLAY option specifies an overlay for page-printer output (see *Using the Xerox 9700 Page Printer*, Reference R1038, for a complete description of overlay options).

PAGES=n Default: No limit

The PAGES option specifies a page limit for a print job. It applies only to jobs directed to *PRINT* or user-defined pseudodevices with TYPE=PRINT.

PAPER={PLAIN | 3HOLE | LABEL24 | LABEL33}
Default: PLAIN

The PAPER option specifies the output form to be used for the page printer. PLAIN specifies unpunched (plain) paper; 3HOLE specifies 3-hole punched paper; LABEL24 and LABEL33 specify 24-up and 33-up label stock, respectively. The obsolete ANY option is synonymous with PLAIN.

PARFIELDCASE={UC | MC} Default: UC

If the PARFIELDCASE option is UC, the PAR field of the RUN command that is passed to programs is mapped to uppercase before program execution begins. If this option is MC (mixed case), the PAR field is not changed.

PDMAP={ON | OFF}　　　　Default: OFF

If the PDMAP option is ON, the predefined symbol section of the loader map is printed whenever the loader map is printed. If PDMAP is OFF, this section is not printed.

{PFX | PREFIX}={ON | OFF}　　　　Default: ON

If the PFX option is ON, the prefix character is printed at the front of each input or output line at the user's terminal. If PFX is OFF, the printing of the prefix character is suppressed. The terminal prefix character indicates which programs are communicating with the user. The prefix character is not printed in batch mode.

PRINT={T3 | TN}　　　　Default: T3

The PRINT option specifies the default line printer character set to be used for *PRINT* output, either the T3 or TN character set. See Appendix G to "Files and Devices" in this volume for further information. This option may be locally overridden by the CONTROL command. This option is ignored for printed output routed via the UMnet/Michnet Computer Network. The PRINT keyword is ignored in batch mode.

PRINTER={PAGE | LINE}　　　　Default: PAGE

The PRINTER option specifies the type of printer to be used for printed output. PAGE specifies the Xerox 9700 page printer. LINE specifies the standard line printer which is available only at the Computing Center batch station.

PRMAP={ON | OFF}　　　　Default: OFF

If the PRMAP option is ON, the pseudoregister section of the loader map is printed whenever the loader map is printed. If PRMAP is OFF, this section is not printed.

PROJECTPWCHANGE={ON | OFF}　　　　Default: ON

The PROJECTPWCHANGE option specifies whether the project director may change the password of the userID via the Accounting Management system. If this option is ON, the password may be changed. This option only needs to be set once (not in each session) and may only be used by nonstudent userIDs (userIDs not beginning with a digit).

PROUTE=station　　　　Default: CNTR

The PROUTE option specifies the default destination for *PRINT* output. The parameter "station" is the destination identification code to which the output is being sent. This may be locally overridden by the CONTROL command. The station specified may not be a UMnet/Michnet Computer Network host. The default is CNTR except for certain hardwired terminals located at remote batch stations. The PROUTE keyword is ignored in batch mode.

{PW | PASSWORD}

The PW option specifies a new password to be used for the user's account. Terminal users will be prompted for the new password and confirmation. Batch users may enter the new password and confirmation on the next line. See the section "UserIDs, Limits, and Sigfiles" for details.

RCPRINT={NEVER | OFF | POSITIVE | NONZERO | NONNEGATIVE | ALWAYS | ON}

The RCPRINT option controls the printing of the user program return code. The return code is set to the contents of general register 15 at the time of program termination. If the R appears in the ETM option, the return code is printed as follows:

| | |
|---|---|
| NEVER or OFF | means never print the return code |
| POSITIVE | means print if >0 and <256 |
| NONZERO | means print if $\neq 0$ |
| NONNEGATIVE | means print if $\geq 0$ and <256 |
| ALWAYS or ON | means always print the return code |

The default is POSITIVE for conversational mode and NONNEGATIVE for batch mode.

RF={hhhhhh | GRx} Default: 000000

The RF option sets a global relocation factor which is used in the DISPLAY, ALTER, and RESTART commands. "hhhhhh" is a hexadecimal constant to be used as the global relocation factor; GRx specifies a register whose contents are to be used as the relocation factor. The relocation factor is set to zero initially. If GRx is specified, the current contents of that register at the time of the SET command is used as the relocation factor.

ROUTE=station Default: CNTR

The ROUTE option specifies the default destination for *PRINT* or *PUNCH* output. The parameter "station" is the destination identification code to which the output is being sent. This may be overridden by the CONTROL command. This option is equivalent to specifying both the CROUTE and PROUTE options. The station specified may not be a UMnet/Michnet Computer Network host. The default is CNTR except for certain hardwired terminals located at remote batch stations. The ROUTE keyword is ignored in batch mode.

SEPCOPY={YES | NO}    Default: NO

Specify whether each copy will have separate head and tail sheets (YES) or whether all copies will be printed together as one job with a single head and tail sheet (NO). The option is only effective for page-printer output and if COPIES=n is specified. If each copy is more than 50 pages, then SEPCOPY=YES is forced.

SEQFCHK={ON | OFF}    Default: ON

Normally, an attempted indexed operation on a sequential file or an attempted sequential operation starting at other than line 1 on a sequential file causes an error condition and an error message to be printed. If SEQFCHK is OFF, the message is not issued and the operation is performed as if not indexed.

SHIFT={YES | NO}    Default: NO

The SHIFT option specifies whether page-printer output is shifted away from the binding edge.

SHOWNAME={ON | OFF}    Default: ON

The SHOWNAME option specifies whether the user's name is printed on the header sheet for batch jobs.

SIGFILE={OFF | FDname}    Default: OFF

The SIGFILE option specifies a file as the special signon file to be used as an implicit SOURCE file each time the user signs on. The user may, for example, place commands in this file to set various global options. The total number of characters in the sigfile name is limited to 50. If SIGFILE is set to OFF, the file or device still exists, but is not used as a sigfile. The setting of SIGFILE affects only subsequent signons. For further details, see the section "UserIDs, Limits, and Sigfiles" in this volume.

SIGFILEATTN={ON | OFF}    Default: ON

If the SIGFILEATTN option is ON, an attention interrupt during the "last signon" message or while the sigfile is being processed, interrupts the processing and resets *SOURCE* to *MSOURCE*. If SIGFILEATTN is OFF, attention interrupts are stacked, but not taken, during the sigfile processing. There is one exception: if the sigfile runs a program which calls the subroutine ATTNTRP, the attention is taken at that point and given to the program. If the sigfile cannot be processed and SIGFILEATTN is OFF, the signon is not allowed. The setting of SIGFILEATTN affects only subsequent signons. For further details, see the section "UserIDs, Limits, and Sigfiles" in this volume.

SPELLCOR={OFF | PROMPT | ON}          Default: PROMPT

If the SPELLCOR option is PROMPT, conversational users are prompted for corrections to some errors detected in MTS command parameters; the corrected form presumed by MTS is printed and the user is prompted for confirmation to the correction. If SPELLCOR is ON, the assumed correct form to an error is used without prompting for both batch and conversational mode (a statement of what was assumed is printed). If SPELLCOR is OFF, no correction is performed. For batch users, PROMPT and OFF are equivalent.

SRVREPLY={ON | OFF}          Default: OFF

If the SRVREPLY option is ON, a server can function in interactive mode. If the SRVREPLY option is OFF, the server may function only in batch mode (users may not reply to queries by the server). This option is useful only for server programs.

SYMTAB={ON | OFF}          Default: ON

If the SYMTAB option is ON, the loader symbol table is retained whenever a program has been loaded. This allows external symbols used in a program to be used by MTS and other user programs; if SYMTAB is OFF, the loader symbol table is not retained.

TDR={ON | OFF}          Default: OFF

If the TDR option is ON, MTS prints on *SINK* after each MTS command the number of page-in operations performed since the last time they were printed. This allows a user to easily obtain a timing estimate for the execution of a program.

TERSE={ON | OFF}          Default: OFF

If the TERSE option is ON, many informational and error messages from MTS are suppressed or abbreviated. If TERSE if OFF, all messages are given in full. BRIEF is a synonym for TERSE; VERBOSE is an antonym for TERSE.

TIME={t | tS | tM | OFF}          Default: OFF

The TIME option specifies the default *local* CPU time limit to be used for all RUN, RERUN, START, RESTART, LOAD, or DEBUG commands that do not have an explicit local time limit or that are not continuing execution with a remaining local time limit. If given in the form “t” or “tS”, the time limit is in seconds; if given in the form “tM”, the time limit is in minutes. The maximum value that can be specified is 27000 seconds. If OFF is specified, the local time limit will be disabled.

TRIM={ON | OFF} Default: OFF

If the TRIM option is ON, lines read or written to files (either line or sequential) are trimmed (trailing blanks are removed). If TRIM is OFF, the lines are not trimmed. The TRIM option can be overridden by the @TRIM modifier on I/O operations (see Appendix A to the section "Files and Devices" in this volume).

TWOSIDED={YES | NO} Default: YES

The TWOSIDED option specifies printing on one or both sides of the paper for page-printer output.

USMSG={ON | OFF} Default: ON

If the USMSG option is ON, any undefined symbols that occur during the loading process are noted and the user is prompted for additional input. If USMSG is OFF, the error message is suppressed and the user is not prompted for more input. This is equivalent to providing an NCA (no care) load record for the undefined symbols.

UXREF={ON | OFF}

If the UXREF option is ON, all references to undefined symbols are printed whenever a program is loaded. This is the same as the UXREF option on the RUN command. If UXREF is OFF, these symbols are not printed. The default is ON for batch mode and OFF for conversational mode.

VERSION(command)={NEW | OLD | CURRENT}
Default: CURRENT

The VERSION option may be used to invoke a different version of an MTS command. "command" may be any MTS command or its abbreviation, which may be optionally preceded by a dollar sign. This feature is normally used to phase in new versions of MTS commands. CURRENT refers to the standard, default version of a command; NEW and OLD refer to new and old versions of the commands. Most MTS command will not have new or old versions; a warning message will be issued to that effect if such a version is requested.

XREF={ON | OFF} Default: OFF

If the XREF option is ON, a loader cross-reference listing is printed whenever a program is loaded. This is the same as the XREF option on the RUN command. If XREF is OFF, the cross-reference listing is not printed.

*LIBRARY={ON | OFF} Default: ON

If the *LIBRARY option is ON, the file *LIBRARY is searched for any unresolved external symbols after a program is loaded (if the LIBR option is ON). If *LIBRARY is OFF (or if LIBR is OFF), this search is not made.

Program Key: *MTS.SET

Examples:

```
SET IC=OFF ENDFILE=OFF
```

This example forces the lines “$CONTINUE WITH” and “$ENDFILE” to be interpreted as data lines rather than implicit concatenation or end-of-file indicators.

```
SET PRINTER=PAGE
```

This example specifies that all output is to be printed on the page printer.

```
CREATE SFYLE
COPY *SOURCE* SFYLE
SET COST=ON AUTOHOLD=ON ERRORDUMP=ON
$ENDFILE
SET SIGFILE=SFYLE
```

This example creates a file SFYLE to be used as a signon file. This signon file contains a SET command to set the COST, AUTOHOLD, and ERRORDUMP options to ON. Each time the user signs on, this file is invoked causing the specified options to be set.

## SIGNOFF

MTS Command Description

Purpose: To notify the system that the current job is finished.

Prototype: SIGNOFF [{SHORT | $ | LONG}]

> In conversational mode, an abbreviated form of the signoff statistics is produced when the SHORT option is given; if $ is specified, only the time and date of the user's signoff, the approximate cost of the session, and the user's remaining balance of funds are printed. In batch mode, the parameter is ignored and the long form is always printed.

Program Key: *MTS.SIGNOFF

Description: The user is signed off the system. All storage acquired and devices attached are released, and all files are closed (temporary files are destroyed). A summary of the job statistics is printed at the end of the job. A list of the statistics produced is given in the section "Batch Use of MTS" in this volume.

If the abbreviation SIG is used, its meaning is taken in context: if the user is not signed on, SIG means SIGNON; if the user is signed on, SIG means SIGNOFF.

## SIGNON

MTS Command Description

Purpose: To identify a user to the system.

Prototype: SIGNON {userid | *} [option ...] ['comment']

Only the “userid” parameter giving the user’s ITD identification number (userID) or “*” specifying the current userID (for *BATCH* jobs) is required. The other legal option parameters that may be given are:

{NOMESSAGES | NOMSGS}

> The NOMESSAGES option specifies that the user’s mailbox will not be checked for incoming messages and no message notification will be printed. The default is that the mailbox is checked.

{PLOTTIME | PT}={t | tM} Default: None

> The PLOTTIME option gives the maximum plot time in “t” seconds or “tM” minutes allowed for any plotting associated with this job. If no plot time limit is specified, the plot time allowed is unrestricted. If a plot time is specified and is exceeded, no plot is generated and the user is signed off immediately (if batch mode).

QUICK

> The QUICK option is a combination of the SHORT and NOMSG options. A condensed signon message will be printed and the mailbox will not be checked for incoming incoming messages and no message notification will be printed.

{SHORT | LONG} Default: LONG

> The SHORT option specifies that a condensed signon message is to be printed giving only the userID and current signon date and time, and the signon date and time of the previous sign-on. The job-type, rate period, and billing class are omitted. If LONG is specified, the full form of the signon message is printed. SHORT is effective only in conversational mode. In addition, the SHORT parameter bypasses the printing of the “Enter password.” message and prompts the user for the password with a question mark (?).

SIGFILE={ON | OFF} Default: ON

> If SIGFILE=OFF is specified on the SIGNON command, user and project sigfile processing will be suppressed if possible. If the SIGFILEATTN option has been set to OFF, user sigfile processing cannot be suppressed. If the project SFATTN option has been set to OFF, project sigfile processing cannot be suppressed.

TIME={t | tS | tM}    Default: 3S

The TIME option specifies the global CPU time limit for the job. If given in the form of the number “t” or “tS”, the time limit is in seconds; if given in the form “tM”, the time limit is in minutes. If the TIME option is not specified, a default time limit of 3 seconds is enforced for batch jobs or enough time to expend the remainder of the user’s funds for a terminal job. The maximum value that can be specified is 27962 seconds.

'comment'

The comment field is used to enter a comment on the SIGNON command line and the front of the printed output (for batch). This may be used to specify the user’s name, project, etc. The comment must be enclosed in primes. Any primes internal to the comment field must be doubled. Other parameters may appear after the comment field only if it is properly terminated with a prime.

The following options are effective in batch mode only (they are ignored in conversational mode).

ADDRESS="line1;line2;..."    Default: No address

Specify the campus mail address for delivered output (when DELIVERY=MAIL is set). This option applies only to page-printer output, not to line-printer or local-printer output.

CARDS=c    Default: 0

The CARDS option specifies the global punched card limit for the job. If the CARDS option is not specified, a default card limit of 0 is enforced and *PUNCH* is undefined. The maximum value that can be specified is 99999 cards.

COPIES=n    Default: 1

The COPIES option specifies the number of printed copies of the output to be produced. If the COPIES option is not specified, one copy of the output is printed. This option is ignored for printed output routed via the UMnet/Michnet Computer Network.

DELIVERY={station | MAIL | NONE}    Default: NONE

The DELIVERY option specifies destination for output delivered by courier service. The parameter “station” is the destination identification code for the station to which output is being sent. The default is NONE which specifies that output is to remain at the station where it was printed. If MAIL is specified, the output will be delivered by campus mail or the US Postal Service. The delivery schedule is given in the public file *DELIVERY.

FORMAT={LANDSCAPE | PORTRAIT | TWOUP | format-name}
Default: LANDSCAPE

The FORMAT option specifies format for page-printer output (see *Using the Xerox 9700 Page Printer*, Reference R1038, for a complete description and list of formats).

JOBNAME={jobname | DEFAULT} Default: RMnnnnnn

The JOBNAME option assigns a job name of 1 to 8 alphanumeric characters to the current job. The first character must be a letter. DEFAULT specifies the default format of "RM" plus six digits.

{LANDSCAPE | PORTRAIT | TWOUP} Default: LANDSCAPE

These options specify the orientation of page-printer output (synonymous with FORMAT for the corresponding values).

MARGIN={n.nn | NO} Default: See text

The MARGIN option sets the left margin for page-printer output to "n.nn" inches. "n.nn" must be less than the current page width (8.5 for portrait orientation, 11.0 for landscape). MARGIN=NO resets the margins to the default for the current format (0.5 for PORTRAIT and 0.65 for LANDSCAPE).

{ONESIDED | TWOSIDED} Default: TWOSIDED

The ONESIDED option specifies that page-printer output will be printed on one side only. TWOSIDED specifies that output will be printed on both sides.

OVERLAY={NONE | SHADED | LINED} Default: NONE

The OVERLAY option specifies an overlay for page-printer output (see *Using the Xerox 9700 Page Printer*, Reference R1038, for a complete description of overlay options).

PAGES=p Default: 50

The PAGES option specifies the global printed page limit for a single copy of the job. If the PAGES option is not specified, a default page limit of 50 pages is enforced. The maximum value that can be specified is 99999 pages.

PAPER={PLAIN | 3HOLE | LABEL24 | LABEL33}
Default: PLAIN

The PAPER option specifies the paper type to be used for page-printer output. PLAIN specifies unpunched (plain) paper; 3HOLE specifies 3-hole punched paper; LABEL24 and LABEL33 specify 24-up and 33-up label stock, respectively. The obsolete ANY option is synonymous with

PLAIN.

PRINT={T3 | TN} Default: T3

The PRINT option specifies the line-printer character set to be used for printing the job, either the T3 or TN character set. See Appendix G to "Files and Devices" in this volume for further information. This option is ignored for printed output routed via the UMnet/Michnet Computer Network.

PRINTER={PAGE | LINE} Default: PAGE

The PRINTER option specifies the type of printer to be used for printed output. PAGE specifies the Xerox 9700 page printer. LINE specifies the standard line printer which is available only at the Computing Center batch station.

RATEGROUP={NORMAL | LOW | DEFERRED | MINIMUM}

The RATE option specifies the rates at which the job is to be charged. NORMAL-rate jobs are eligible for immediate execution. LOW-rate jobs are automatically held until the next low-, deferred-, or minimum-rate period occurs. DEFERRED-rate jobs are automatically held until the next deferred- or minimum-rate period occurs. MINIMUM-rate jobs are automatically held until the next minimum-rate period occurs. Details on the assignment of rates and the schedule of rate periods is given in the section "Batch Use of MTS" in this volume. PRIORITY is a synonym for RATEGROUP.

RERUN={YES | NO} Default: YES

The RERUN option controls the number of times a job may be rerun. If RERUN=NO is specified, the job will not be allowed to be executed more than once. Thus, if the system crashes or the job must be rerun for any reason (such as the lack of tape-drive availability), the job will be aborted. No output listing will be produced for the aborted job. The default is RERUN=YES, in which case the job may be rerun from the start.

ROUTE=station
PROUTE=station
CROUTE=station

These options specify the destination for printed and punched output. PROUTE and CROUTE specify only printed or punched output, respectively; ROUTE specifies both. The parameter "station" is the destination identification code for the station to which the output is being sent. If these options are not specified, the output is returned to the station where the job originated, except if that station has no card punch, then punched output is routed to CNTR. If "station" is CNTR or LOCAL, the appropriate output is routed to the Computing Center. These options may be used to route printed and punched output to stations at Wayne State University (see the section "UMnet/Michnet Computer Network

Batch" in this volume).

SEPCOPY={YES | NO}  Default: NO

Specify whether each copy will have separate head and tail sheets (YES) or whether all copies will be printed together as one job with a single head and tail sheet (NO). The option is only effective for page-printer output and if COPIES=n is specified. If each copy is more than 50 pages, then SEPCOPY=YES is forced.

SHIFT={YES | NO}  Default: NO

The SHIFT option specifies whether page-printer output is shifted away from the binding edge.

TAPES=n  Default: 0

The TAPES option specifies the number of tape drives required at any one time. Although this is not absolutely necessary, it is highly recommended as it will improve that chances that tape drives are available when your job starts executing.

TWOSIDED={YES | NO}  Default: YES

The TWOSIDED option specifies printing on one or both sides of the paper for page-printer output.

WAITUNTIL='time and/or date'

If the WAIT option is given, the batch job will not be executed before the specified time. If a date alone is given, the time is assumed to be 00:00 on the morning of that date. If a time alone is given, the current date is assumed provided that the time is later in the day; otherwise the next day is assumed.

The time should be given as a 24-hour time (hh:mm) with a limit of resolution in minutes, e.g., '13:00'. The date should be given as a month spelled out, followed by the day, followed by an optional year separated by a comma, e.g., 'January 3, 1983' or 'July 10'. If the year is omitted, it will be assumed to be the first future occurrence of the specified date. The time must precede the date if both are given, e.g., '8:00 August 30, 1983'.

Note that the system only guarantees that the job will not be run before the specified time and date. No guarantee is made about the exact time the job will execute. The job will execute at the first opportunity after the specified time (subject to the usual limits of execution time limit requested, system availability and load, and the time of job submission).

Program Key: *MTS.SIGNON

Description: The SIGNON command identifies the user to the system (signs him on), and in the case of batch jobs, establishes certain constraints for the job. The SIGNON command must be the first command of the user's job.

For batch jobs, the system expects to find the password starting in column 1 on the card following the SIGNON card.

For terminal jobs, the system prompts for the password. Conversational users are given three attempts to enter a valid userID and password before being disconnected. If an end-of-file is given while the user is being prompted for the password, the signon will be aborted immediately.

If errors are detected while processing the options on the SIGNON command, the signon will be aborted immediately if in batch mode.

A user may be signed on more than once at any one time for any given userID, if the userID was so authorized when it was created or by the project director using accounting mode.

If the abbreviation SIG is used, its meaning is taken in context: if the user is not signed on, SIG means SIGNON; if the user is signed on, SIG means SIGNOFF.

The user may designate a signon file (sigfile) which is used as an implicit SOURCE file after the user signs on. For the complete details of using sigfiles, see the section "UserIDs, Limits, and Sigfiles" in this volume.

Examples:

```
SIGNON 2AGA 'JOHN Q.  DOE'
```

The user with userID 2AGA is signed on to the system. All global limits and output specifications are defaulted.

```
SIG 2AGA T=1M P=100 C=50 'JOHN Q.  DOE'
```

The user with userID 2AGA is signed on to the system. The global limits are 1 minute of CPU time, 100 pages of printed output, and 50 punched cards.

## SINK

MTS Command Description

Purpose: To change the destination or "sink" for normal output lines.

Prototype: SINK {FDname | PREVIOUS}

One of the two following parameters must be given:

FDname

The name of the file or device that is to become the current sink of output lines.

PREVIOUS

The PREVIOUS parameter specifies that the previous sink is to be restored as the current sink.

Program Key: *MTS.SINK

Description: When the SINK command is given, the pseudodevice *SINK* is reassigned to the file or device specified. Output which defaults to *SINK* is written on that file or device. The master sink *MSINK* remains as the terminal in conversational mode or the printer in batch mode. Initially, *SINK* has the same assignment as *MSINK*.

In conversational mode, error messages requiring user interaction are directed to *MSINK*.

An attention interrupt reverts *SINK* to *MSINK*.

A one-level pushdown list of sink devices is maintained. The PREVIOUS parameter uses this pushdown list to restore the previous sink device.

The current status of *SINK* and the previous sink file or device (if any) may be displayed by issuing the command DISPLAY *SINK*.

Example:

```
SINK A
```

The file A becomes the current sink for output lines.

## SOURCE

MTS Command Description

Purpose: To change the source of input lines.

Prototype: SOURCE {FDname | PREVIOUS}

One of the two following parameters must be given:

FDname

> The name of the file or device that is to become the current source of input lines.

PREVIOUS

> The PREVIOUS parameter specifies that the previous source is to be restored as the current source.

Program Key: *MTS.SOURCE

Description: When the SOURCE command is given, the pseudodevice *SOURCE* is reassigned to the file or device specified. The next input line in MTS command mode is read from that file or device. Input that defaults to *SOURCE* is read from this file or device. The master source *MSOURCE* remains as the terminal in conversational mode or the card reader in batch mode. Initially, *SOURCE* has the same assignment as *MSOURCE*.

In conversational mode, responses to error messages requiring user interaction are read from *MSOURCE*.

An attention interrupt, an end-of-file condition on *SOURCE* when trying to read an MTS command, or an invalid MTS command reverts *SOURCE* to *MSOURCE*.

A one-level pushdown list of source devices is maintained. The PREVIOUS parameter uses this pushdown list to restore the previous source device.

The current status of *SOURCE* and the previous source file or device (if any) may be displayed by issuing the command DISPLAY *SOURCE*.

Example:

```
SOURCE A
```

> The file A becomes the current source for input lines.

## START

MTS Command Description

| | |
|---|---|
| Purpose: | To restart (or initiate) execution of a program following either initial loading, an interrupt, or a subroutine call to ERROR, MTS, or MTSCMD.<br><br>This command is identical to the RESTART command. |
| Program Key: | *MTS.START |

## SYSTEMSTATUS

MTS Command Description

Purpose: To provide information about the status of jobs and other various aspects of the system operation.

Prototype: SYSTEMSTATUS [systemstatus-command]

systemstatus-command

> "systemstatus-command" is any single systemstatus command. The systemstatus command is executed and an immediate return is made to MTS command mode. If "systemstatus-command" is omitted, systemstatus mode is entered.

Program Key: *SYSTEMSTATU

Description: Systemstatus mode provides status information about various aspects of the system. This information includes the status of individual jobs, the current system load, the current tape-mount queue, and the current batch execution, print, and punch queues.

If "systemstatus-command" is omitted, systemstatus mode is entered. In systemstatus mode, all input lines are treated as systemstatus commands until either an end-of-file, MTS, MCMD, RETURN, or STOP command is entered. In addition, any input line beginning with a dollar sign ($) is treated as an MTS command.

Commands: Any of the systemstatus commands described below may be used either as a single parameter on the SYSTEMSTATUS command or directly in systemstatus mode.

DELAY T=n

> The DELAY command causes a "n"-second real-time delay before the next command is processed.

DISPLAY location[+disp[+disp...]] [count]

> The DISPLAY command displays a block of virtual memory beginning at the address specified by "location". The number of words in the block is specified by "count". "location" must be a valid hexadecimal address. "count" must be a decimal integer; if omitted, the default is 1. Optional hexadecimal displacements may be *added* to "location" in specifying the beginning address of the block. If the currently loaded program is a "run-only program", then user and system storage may not be displayed.

LOAD [job]

If “job” is omitted, various information about the total system operation is displayed at 20-second intervals. The information included is as follows:

| | |
|---|---|
| DT | (delta time) is the actual time between successive intervals of output (nominally 20 seconds). |
| EXQ | (execution queue) is the number of batch jobs waiting to execute. |
| PRT | (print queue) is the number of jobs waiting to print. |
| PCH | (punch queue) is the number of jobs waiting to punch. |
| BP | (batch preferred) is the desired number of batch jobs the system would prefer to execute under the current system load. An asterisk (*) following this number indicates that the system is currently overloaded. |
| AB | (actual batch) is the number of batch jobs currently executing. |
| BPH | (batch per hour) is the approximate batch throughput rate (in jobs per hour) measured over the last several minutes. |
| AL | (actual lines) is the total number terminal users currently signed on. |
| VP | (virtual pages) is the total number of pages of virtual memory currently in use by all jobs. |
| RP | (real pages) is the number of virtual pages occupying real pages for all jobs. |
| DPA | (drum pages available) is the number of pages available for use on the primary paging devices. |
| PA | (paging activity) is the average number of page-in operations (drum reads) per second over the previous interval. |
| DA | (disk activity) is the average number of disk I/O operations per second over the previous interval. |
| CA | (channel activity) is the average number of channel I/O operations per second over the previous interval (this includes DA and a fraction of PA). |
| %PI | (percent processor idle) is the average CPU idle (wait state) time over the previous interval. |
| Q | (queue) is the current number of jobs on the system CPU queue. |
| TQ | (tape queue) is the current number of jobs in the tape-mount queue. |

The values for DPA, PA, DA, CA, and %PI may be followed by a plus sign (+) indicating that the condition represented by the number is excessive, i.e., that the system is overloaded in this aspect. The presence of any plus sign causes the asterisk summary mark to be shown after the BP value.

If the optional job number parameter is specified, then a sampling of information about the specified job is displayed every 20 seconds as follows:

| | |
|---|---|
| DT | (delta time) is the same as above. |
| DPT | (delta processor time) is the number of seconds of CPU time used by the job during the previous interval. |
| VP | (virtual pages) is the number of pages of virtual memory |

| | |
|---|---|
| | currently in use by the job. |
| RP | (real pages) is the number of virtual pages owned by the job currently in real memory. |
| RPN | (real pages needed) is the number of pages currently estimated to constitute the job's working set. |
| PA | (paging activity) is the average number of page-in operations per second for the job over the previous interval. |
| DA | (disk activity) is the number of disk I/O operations during the previous interval. |
| %PT | (percent processor time) is the percentage of total system processor time used by the job during the previous interval. |
| %VP | (percent virtual pages) is the percentage of the total virtual memory being used by the job. |
| %RP | (percent real pages) is the percentage of the total real memory being used by the job. |
| %PA | (percent paging activity) is the percentage of the total paging activity being done by the job during the previous interval. |
| %PI | (percent processor idle) is the same as above. |

This command continues to print information every 20 seconds until interrupted by an attention interrupt.

MCMD MTS-command

The MCMD command executes an MTS command while in systemstatus mode. As an alternative, any input line beginning with a dollar sign ($) is also executed as an MTS command in systemstatus mode.

MTS

The MTS command returns control to the caller (normally MTS command mode). This command is identical to the RETURN command.

QUEUE [{receipt | USER | ALL | ROUTE=station | *} **...**]

The QUEUE command displays the status of individual batch, *PRINT*, or *PUNCH* jobs and/or the status of any or all remote batch stations.

If a receipt number is specified, the queue status of the specified job (e.g., awaiting execution, executing, awaiting print, printing, etc.), the rate period and routing of the job, and the relative position of the job within its queue are printed.

If USER is specified, all jobs awaiting execution for the current user are displayed.

If ALL or no parameter is specified, counts of the total number of pages waiting to be printed and cards waiting to be punched along with the print, punch, and execution queue lengths are printed.

If a particular station is specified, the connect status of that station, in addition to the information specified by ALL, is printed. The print and punch counts and queues reflect only the jobs routed to that station. The execution queue reflects the entire execution queue, not just the jobs submitted from that station. Codes for some of the remote stations are as follows:

| | |
|---|---|
| CNTR | Computing Center (North Campus) |
| NUBS | North University Building Station |
| UNYN | Michigan Union Station |
| DRBN | U–M Dearborn Campus |
| FLNT | U–M Flint Campus |

If "*" is specified, the status of all queued jobs waiting to be imported is displayed.

REPEAT [T=n]

The REPEAT command repeats the most recent non-REPEAT command entered. It may be used conveniently to reenter an input line. If T=n is specified, the command is repeated every "n" seconds until interrupted by an attention interrupt.

RETURN

The RETURN command returns control to the caller (normally MTS command mode). This command is identical to the MTS command.

STOP

The STOP command (or an end-of-file) returns control to the caller (normally MTS command mode).

TAPEQUEUE [LIST]

The TAPEQUEUE command displays information about the current tape-mount queue. If the LIST parameter is given, the command prints one line for each user waiting in the queue (the display gives the job number, userID, project number, total number of 9TP drives requested, and number of 6250 bpi and 800 bpi drives required, if known). If no parameter is given, the command prints the current number of terminal users and batch jobs waiting in the queue, and the total number of 9TP drives required.

TASKS [descriptor]

The TASKS command prints one line of output for each job which fits the descriptor category. Each output line contains, from left to right:

The job number
The job name
The job table address
The number of virtual memory pages in use by the job
    if relocatable (such as MTS)

The job status
The job table parameters (if any)
The I/O devices attached to the job (if any)

For MTS, the job table parameters are the four-character userID (blank if the job has no user signed on), the four-character project number (IDLE if no user is attached, and BUSY if a user is attached, but not signed on), and the receipt number (if it is a batch job).

The descriptors specify the types of jobs to be displayed. The valid descriptors are:

| | |
|---|---|
| xxxx | job number “xxxx” |
| B | all batch jobs |
| F | all jobs |
| M | all MTS jobs |
| N [xxxx] | all non-MTS jobs, or job name “xxxx” |
| D xxxx | device name “xxxx” |
| T xxxx | device type “xxxx” |
| U xxxx | user userID “xxxx” |
| P xxxx | projectID “xxxx” |

If a job number is specified, the first nonnumeric character terminates the number; the remainder of the input line is ignored. If the first nonblank character in the descriptor is B, F, or M, then the remainder of the input line is ignored, e.g., it is permissible to enter MTS instead of M. If the first nonblank character is D, T, U, or P, then the argument for the descriptor begins with the next nonblank character, unless there are more than four characters given, in which case only the last four are used. The N descriptor followed by a 1 to 8-character job name (e.g., PDP) specifies all jobs with the given name. If no name is given, all non-MTS jobs are printed. In the argument for the D, U, and P descriptors, the pound sign (#) is used as a fill character which matches any character.

USERS

The USERS command displays counts of the current number of terminal users, batch jobs executing, available terminal lines, non-MTS jobs running, pages of virtual memory in use, and pages of real memory in use.

Examples:

```
SYSTEMSTATUS
DISPLAY 803064+20 4
```

The above command sequence enters systemstatus mode and displays four words of virtual memory beginning at address 803084.

```
SY LOAD
```

The above command displays system load information at 20-second intervals until interrupted by an attention interrupt.

```
SY QUEUE 608999 MS1336
```

The above command displays the status of the batch, *PRINT*, or *PUNCH* jobs with receipt numbers 608999 and MS1336. Control returns to MTS command mode.

```
SY
Q 610112
REP T=15
STOP
```

The above command sequence is similar to the preceding example except that the QUEUE command is entered in systemstatus mode. In this manner, the REPEAT command may be used to repeat the QUEUE command every 15 seconds until interrupted by an attention interrupt.

## TRUNCATE

MTS Command Description

Purpose: To deallocate unused space at the end of a file or a set of files.

Prototype: TRUNCATE filelist [{ALLOK | PROMPT}]

Program Key: *MTS.TRUNCAT

Description: The “filelist” parameter specifies the file or the set of files to be truncated and may be a single file name, a file-name pattern, or a parenthesized list of either.

Confirmation is not requested in conversational mode if a single file name is specified. If more than one file is specified, a single summary confirmation is requested. If the reply is OK, then all the files specified are truncated; otherwise, none are truncated. Confirmation is not requested in batch mode.

The ALLOK option may be used to bypass the confirmation request when several files have been specified. The PROMPT option causes prompting for confirmation for each individual file.

The response to a prompt for confirmation may be OK to truncate the file, NO to skip the file but continue with the next file in the list, or CANCEL to terminate the command.

The user must have TRUNCATE access to the files. This command only truncates files. It does not compact or optimize files; the DUPLICATE command may be used to do this.

Any unused space at the end of the each file truncated is deallocated and the accounting and billing information is correspondingly adjusted.

A file or a set of files may be truncated from a program by calling the TRUNC subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

Example:

```
TRUNCATE DATA1
```

The file DATA1 is truncated.

```
TRUNCATE DATA?  PROMPT
```

This command truncates all files beginning with the string DATA. The terminal user is prompted for confirmation for each file to be truncated.

## UNLOAD

MTS Command Description

| | |
|---|---|
| Purpose: | To unload the currently loaded program in virtual memory or command language subsystem (CLS). |
| Prototype: | UNLOAD [CLS=clsname] [EVERYTHING] |

CLS=clsname

If CLS=clsname is specified, the corresponding command language subsystem (CLS) is unloaded. This is normally only useful for system programmers.

EVERYTHING

If EVERYTHING is specified, the currently loaded program and all loaded command language subsystems (CLSs) are unloaded.

Program Key: *MTS.UNLOAD

Description: If no keywords are specified, the UNLOAD command unloads the current program (if any) in virtual memory that was previously loaded by a LOAD, RUN, RERUN, or DEBUG command. If a program returns normally by returning to the system with a return code of zero or by calling the SYSTEM subroutine, it is unloaded automatically at that time (unless it was loaded via the DEBUG command). All storage allocated to the program is released, and all files and devices opened at execution time are closed.

## UNLOCK

MTS Command Description

Purpose: To explicitly unlock a file that has been previously locked explicitly by the LOCK command.

Prototype: UNLOCK filename

filename

"filename" is the name of the file to be unlocked.

UNLK is an alternate name for UNLOCK.

Program Key: *MTS.UNLK

Description: The file is unconditionally unlocked (if the job has no other locking requests associated with the file). This command has no effect on any locks set implicitly or via the LOCK subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003). Implicit locks or locks set via the LOCK subroutine can be reset by unloading the currently loaded program or command language subsystem (e.g., the $EDIT command).

A file may be unlocked from a program by calling the UNLK subroutine except that a file locked by the LOCK command cannot be unlocked by the UNLK subroutine.

Example:

```
UNLOCK FILE1
```

The file FILE1 is explicitly unlocked.

## VIEW

MTS Command Description

Purpose: To look at print and batch jobs that are queued by the Resource Manager for execution or printing.

Prototype: VIEW [jobnumber [;view-command]]

Commands: The following commands are recognized by the VIEW command in view mode:

CANCEL

This cancels the entire job like the MTS $CANCEL command.

DISPLAY

This displays information about the currently viewed job.

HELP

This provides assistance with using the VIEW command.

MTS

This returns control to the calling program, normally MTS. Reissuing the VIEW command will restart the VIEW program.

ROUTE station

This reroutes the job to the specified station. Another VIEW command must be given in order to continue viewing the job.

STOP or end-of-file

This terminates the VIEW program and returns control to the calling program (normally MTS).

TAILSHEET

This displays the signoff statistics (for batch jobs only).

edit-command

Any MTS File Editor command may be given to look at the job, including the VISUAL command. Editor commands that attempt to change the contents of the job will not be effective.

Program Key: *VIEW

Description: The VIEW command allows users to look at batch and print jobs that are queued by the Resource Manager for execution or printing.

For example, you can look at the contents of the print job 123456 by first entering the command

```
VIEW 123456
```

at the MTS "#" prompt. Then, you can use MTS File Editor commands (including visual mode) to examine the job. You can copy sections of the job to other files, cancel it, or reroute it. If the job is queued for execution, you will see the commands that make up the batch job. If the job is queued for printing, you will see the output produced by the job.

Print jobs must be released to the Resource Manager before they can be viewed. That is, you cannot give the commands

```
# COPY file *PRINT*
> *PRINT* assigned job number 123456
# VIEW 123456
```

but you can give

```
# COPY file *PRINT*
> *PRINT* assigned job number 123456
# RELEASE *PRINT*
  *PRINT* RM123456 released to ...
# VIEW 123456
```

You can view print output only while it is queued for printing. Once printed, it can no longer be viewed.

You can send jobs to the VIEW command by using the ROUTE=VIEW option on the $CONTROL or $SIGNON commands. Jobs submitted with this routing will not print but will remain queued in the system for up to 11 days until rerouted or canceled (both described later).

For example, to use ROUTE=VIEW for a print job, use the commands

```
CONTROL *PRINT* ROUTE=VIEW
COPY file *PRINT*
RELEASE *PRINT*
```

You can then use the VIEW commands described below. For batch jobs, place the parameter ROUTE=VIEW on your $SIGNON command.

If you currently submit batch jobs and only examine the printed output to determine the success or failure of the job, then VIEW can be of use to you. If you submit your job with a ROUTE=VIEW, you can examine the output from your terminal. Having looked at it, you can then cancel it.

If you submit a batch job that is going to generate a lot of output, it might be worth first routing the output to VIEW. You can then examine the output and, if the job is correct, reroute it to the printer.

The VIEW command requires a job number to identify the job. You can find the job number by using the LOCATE command. To see what jobs you have in the system, enter the command:

    LOCATE

or, to see only the jobs with ROUTE=VIEW:

    LOCATE VIEW

You will see a message such as

```
RM299444 (234567) waiting print, posn 0, Route=VIEW.
RM299593 (234568) waiting print, posn 1, Route=VIEW.
```

Then, you can use the VIEW command to view a job; e.g.,

```
VIEW 234567
```

You will see a message such as

```
* Print job 234567 RM234567 1ABC   3 pages submitted at
    11:34:32 Tue Jul 30/91
```

The asterisk "*" is a prompt that indicates the VIEW command is active. To examine the job in visual mode, enter the following command at the "*" prompt:

    V

Subsequently, you could cancel the job with the command

    CANCEL

To exit the VIEW command, enter

    STOP

at the "*" prompt. For more help on the VIEW command, enter

    HELP

while you are in the VIEW command.

You can reroute jobs from VIEW to a printer by entering the following command at the "*" prompt:

    ROUTE station

For example,

```
ROUTE CNTR
```

will route the output of a job to the CNTR printer. You can send jobs to other printers, such as the printer at NUBS, by substituting NUBS for CNTR. Please note that you will probably not want to view text-formatted jobs, such as .DVI files, since their output normally is not readable using the File Editor.

A one-shot VIEW command may be given in the form

VIEW nnnnnn; view-command

The ";" after the job number is required. For example, you could reroute a job from one printer to another by giving the commands

```
# LOCATE PRINT
  RM123456 (123456) waiting print, posn 0, Route=UNYN.
# VIEW 123456; ROUTE NUBS
```

If you work under more than one MTS userID, please note that you can only view jobs from the userID on which you originally executed them.

# ABNORMAL CONDITIONS

The following paragraphs describe various abnormal conditions that may occur during the execution of a program, how the resulting interrupts are usually handled, and how the user may prepare to intercept them if they should occur.

There are several categories of interrupts which may occur while a program is executing. In this section, four categories of interrupts which may be controlled by a program are discussed. They are program interrupts, attention interrupts, timer interrupts, and input and output errors.

## PROGRAM INTERRUPTS

There are fifteen different classes of program interrupts (exceptions). They are the following:

| *Program Interrupt Class* | *Interruptio Code (hex)* |
|---|---|
| Operation | 1 |
| Privileged operation | 2 |
| Execute | 3 |
| Protection | 4 |
| Addressing | 5 |
| Specification | 6 |
| Data | 7 |
| Fixed-point overflow | 8 |
| Fixed-point divide | 9 |
| Decimal overflow | A |
| Decimal divide | B |
| Exponent overflow | C |
| Exponent underflow | D |
| Significance | E |
| Floating-point divide | F |

For a complete description of the meaning of these program interrupt classes, see the *IBM System/370 Principles of Operation*, form GA22-7000. Normally, when a program interrupt occurs in a program, the message

```
    interrupt exception at xxxxxxxx
```

is printed, where "interrupt" is the program interrupt class (as given in the table above) and "xxxxxxxx" is the hexadecimal or symbolic address of the instruction that caused the interrupt.

Addressing and protection exceptions are caused by the user's program specifying an invalid address. An addressing exception is caused by an address specified outside the range of virtual memory. A protection exception is caused by specifying an address in virtual memory which is not legal for the user's program to specify (usually an address reserved for the system supervisor).

After a program interrupt has occurred, the following happens:

(1) The general registers, floating-point registers, and PSW are saved. In batch mode, the

general registers, floating-pointing registers, and PSW are displayed even if the ERRORDUMP option is OFF.

(2) A storage dump is given for batch jobs, if previously requested by the user via one of the following MTS or debug commands:

```
SET ERRORDUMP=ON
SET ERRORDUMP=LIB
```

(3) Control is returned to MTS command mode or debug mode.

When a return is made to MTS command mode, the user can give any MTS command, including RESTART, which causes the program to continue execution at the instruction following the one causing the program interrupt.

There are several ways that a programmer can alter the normal processing of a program interrupt.

The BPI (*b*ranch on *p*rogram *i*nterrupt) macro may be used in 360/370-assembler programs to specify a branch address to be taken when one of a specified class of interrupt types occurs. See the BPI macro description in *MTS Volume 14: 360/370 Assemblers in MTS*, Reference R1014, for the complete description, calling sequences, and examples of use.

The subroutine PGNTTRP (*p*ro*g*ram i*nt*errupt *tr*a*p*) allows the user to specify an exit routine to transfer to when a program interrupt occurs. After the interrupt occurs and the exit is taken, the intercept is cleared so that another call to PGNTTRP is necessary in order to intercept the next program interrupt. PGNTTRP can be called directly from a 360/370-assembler program and indirectly through the subroutine RCALL in FORTRAN and other higher-level languages. See the PGNTTRP subroutine description in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for the complete description, calling sequences, and examples of use.

The subroutine SPIE (*s*pecify *p*rogram *i*nterrupt *e*xit), which is callable from a 360/370-assembler program, not only allows the user to specify an exit routine, but also provides the facility to specify the class of program interrupts for which this exit should be used. For any of the fifteen program interrupts not specified, the normal program interrupt procedure is followed. The specifications set up by a call to SPIE remain in effect until a subsequent call overrides them except while executing a SPIE exit routine. This subroutine is normally called by using the SPIE macro. See the SPIE subroutine and macro descriptions in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, and *MTS Volume 14: 360/370 Assemblers in MTS*, Reference R1014, for the complete descriptions, calling sequences, and examples of use. SPIE and PGNTTRP are mutually exclusive; either one overrides the other.

When a program interrupt occurs, a check is first made to determine if the instruction(s) immediately following the instruction that caused the interrupt is a BPI macro instruction. If it is, the type of program interrupt that occurred is compared with the type specified by the BPI macro. If there is a match, the condition code is set to reflect the interrupt that occurred and the branch is taken. If there is no BPI transfer made (either because there was no BPI instruction or because the program interrupt type did not match the BPI type), then a check is made to determine if a PGNTTRP (or SPIE) exit is active. If there is one, the exit is taken; otherwise, the program interrupt message is printed and a return is made to MTS command mode or debug mode.

In PL/I, many asynchronous conditions are predefined as ON-conditions. Therefore the user can determine by means of the ON statement what processing should occur when one of these conditions occurs. The following program interrupts are among these predefined conditions:

| *Program Interrupt* | *PL/I On-Condition Code* |
|---|---|
| Fixed-point overflow | FIXEDOVERFLOW |
| Fixed-point divide | ZERODIVIDE |
| Exponent overflow | OVERFLOW |
| Exponent underflow | UNDERFLOW |
| Floating-point divide | ZERODIVIDE |

Those asynchronous conditions that are not predefined cannot be intercepted. Those that cannot be intercepted and those that the user chooses not to intercept are handled by the PL/I library. For further details, see the *IBM System/360 PL/I (F) Language Reference Manual*, form GC28-8201.

## ATTENTION INTERRUPTS

Attention interrupts usually occur when a terminal user depresses the ATTN or BREAK key. The system responds with

```
Attn!
```

or, if a program was in execution

```
Attention interrupt at xxxxxxxx
```

where “xxxxxxxx” is the hexadecimal or symbolic address of the point of interruption. The system returns to MTS command mode or debug mode.

Normally, an attention interrupt is taken by MTS when it occurs. However, under the following circumstances, the interrupt is not taken, but is left pending:

(1) The interrupt has occurred during the execution of a sensitive portion of the system such as an I/O subroutine call.

(2) The ATTNOFF bit (accessible through the CUINFO subroutine) has been set to inhibit attention interrupts.

(3) The interrupt has occurred during the processing of a sigfile with the SIGFILEATTN option set to OFF and no ATTNTRP exit routine is active.

When an interrupt occurs in these cases, but is not immediately taken, the ATNBIT bit (accessible through the subroutine GUINFO) is set, indicating that an interrupt is pending. The ATNBIT bit also may be explicitly set to cause a pending attention interrupt. Each time the device support routine interface is entered in MTS (when an I/O subroutine is called or returns), the ATNBIT bit is tested for a pending interrupt; if there is one, it is taken at that point. The pending interrupt may be explicitly suppressed by resetting the ATNBIT bit before it is tested.

The subroutine ATTNTRP (*att*entio*n* interrupt *tr*a*p*) allows the user to specify an exit routine to transfer to upon the occurrence of an attention interrupt. When the interrupt occurs and the exit is taken, the intercept is cleared so that another call to ATTNTRP is necessary to intercept the next attention interrupt. ATTNTRP can be called directly from a 360/370-assembler program and indirectly through the subroutine RCALL in FORTRAN and other higer-level languages. See the ATTNTRP subroutine description in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003, for the complete description, calling sequences, and examples of use.

## TIMER INTERRUPTS

Three different types of timer interrupts may occur while a program is executing. These are:

(1) Global time limit exceeded (which happens if the time estimate specified by the TIME parameter on the SIGNON command is exceeded or if the balance of funds remaining in the user's account is depleted).

(2) Local time limit exceeded (which happens if the time estimate specified by the TIME parameter on the RUN, RERUN, START, RESTART, LOAD, DEBUG, or SET commands is exceeded during the execution of a program).

(3) Program-specified timer interrupts, which are established by calls on certain system subroutines.

The first two types result in the system message

```
Global time limit exceeded [at xxxxxxxx]
```

or

```
You have run out of money [at xxxxxxxx]
```

or

```
Local time limit exceeded at xxxxxxxx
```

where "xxxxxxxx" is the hexadecimal or symbolic address of the point of interruption. These conditions cause a dump in batch mode if the ERRORDUMP option is specified in MTS command mode or debug mode. Neither of these interrupts can be intercepted by a program, but it is possible to determine the times at which they may occur (via the GUINFO subroutine) and set up a program-specified timer interrupt to occur before the global or local limit is reached. Thus, a program can be organized to print more useful diagnostic information in cases of infinite loops, etc.

Program-specified timer interrupts are set up by calls to the subroutines TIMNTRP, SETIME, GETIME, RSTIME, TWAIT, and TICALL which are described in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003. These subroutines provide for the enabling and disabling of timer interrupts, and the specification of time intervals in several forms, including real time or task time, relative or absolute time, and in units of microseconds, timer units, or character string time of day.

## INPUT AND OUTPUT ERRORS

The input and output of data by programs is handled by I/O subroutines called from the program. The I/O subroutines always provide a return code, the exact meaning depending on the file or device used in the operation. A description of the return codes that may occur with a particular file or device is given in the section "I/O Subroutine Return Codes" in *MTS Volume 3: System Subroutine Descriptions*, Reference R1003.

In general, a return code of zero means successful completion of the input or output operation, and a return code of 4 means end-of-file-or-device. A return code greater than 4 normally signifies an error condition and is not passed back to the caller, but instead causes an error comment to be printed and saved for later reference by the GUINFO subroutine and control to be returned to MTS command mode or debug mode. The error comment usually describes the error condition and its location.

Normally, when an input or output error occurs in a program, the message sequence

```
error-message
I/O interrupt at xxxxxxxx
```

is printed where "error-message" is the error comment describing the error condition and "xxxxxxxx" is the hexadecimal or symbolic address of the point of the interruption. The system returns to MTS command mode or debug mode.

The user may suppress the error comment and return control to the calling program by specifying the ERRRTN I/O modifier. If this modifier is specified in an I/O subroutine call, then when an error occurs, no error comment is printed, the SETIOERR-SIOERR exit is ignored, and the return code is passed back to the calling program. The error comment that would have been printed is available by calling the GUINFO subroutine (see *MTS Volume 3: System Subroutine Descriptions*, Reference R1003).

PL/I has predefined many of the I/O errors as ON-conditions. Thus, processing of these interrupts can be determined by ON statements.

A call to the CONTROL subroutine always returns without printing an error message, but it may return a message to the caller and also save it for later reference by the GUINFO subroutine.

## OTHER SYSTEM ERRORS

If an error occurs during the execution of a system subroutine, the following message will be printed in addition to any other error messages that may be appropriate:

```
Error occurred in a system subroutine.
RESTART inadvisable. Callers GR14=xxxxxxxx
```

where "xxxxxxxx" is the contents of the calling program's general register 14 at the time the system subroutine was called, i.e., the location from which the system subroutine was called. The most common reason for this type of error is a bad parameter list to a system subroutine, although other types of errors such as an internal system error may also produce this error comment.

If a program interrupt occurs within MTS or one of its subcomponents such as the File Editor or SDS, or if an invalid SVC instruction is executed, a message will be printed in one of the following forms:

```
MTS program interrupt. PSW=xxxxxxxx xxxxxxxx
SDS program interrupt. PSW=xxxxxxxx xxxxxxxx
Invalid SVC(n). PSW=xxxxxxxx xxxxxxxx
Fatal exit. Code=xxxx
```

If one of the above messages or any other similar message is printed, please return the output to the ITD Consulting Staff. This will aid in detecting and correcting system errors.

Users should report errors in MTS, public-files, or other software supported by ITD to the consultants. If the consultants are not available and the error is potentially serious, the system operators should be notified. Equipment problems should be reported to the system operators.

If the system crashes (halts due to a catastrophic and nonrecoverable error), the user will not be charged for the session; hence, rebates will not be granted for the work lost.

# INDEX

Reader's Comment Form

---

**The Michigan Terminal System**

November 1991

---

Errors noted in publication:

---

Suggestions for improvement:

---

Date ______________________________

Name ______________________________

Address ______________________________

Your comments will be greatly appreciated. Please fold the completed form as shown on the reverse side, seal or staple, and drop in Campus Mail or in the Suggestion Box at any Campus Computing Site.

fold here

---

Documentation Group
ITD Consulting and Support Services
The University of Michigan
611 Church, 2nd Floor
Ann Arbor, Michigan 48104-3056
USA

---

fold here