NORTHUMBRIAN
UNIVERSITIES
MULTIPLE
ACCESS
COMPUTER

MTS
USERS GUIDE

KEYWORDS: USERS
REFERENCE
MTS

DECEMBER 1978

NEWCASTLE
UPON TYNE

DURHAM

PREFACE

There are three main publications about the Michigan Terminal System (MTS). The first, which all users should read, is "Introduction to MTS". Many people, especially those who use one of the major packages, will find that introduction is all they need. The second is this "MTS Users' Guide" which describes the commonly used facilities of MTS as it is implemented on NUMAC's IBM 370/168. This is intended as a reference guide for those who write their own programs and thus need to use a wider range of MTS facilities. Finally, the third publication is the University of Michigan's manual Volume 1: "The Michigan Terminal System" which serves as a system reference guide for NUMAC staff and those users who need sophisticated facilities. This document assumes that you have read "Introduction to MTS" and had some initial experience with the system. It expands the content of that document with only a necessary minimum of repetition.

Throughout this guide, reference is made to "NUMAC Service Documents" : these describe the administrative and operational arrangements for providing NUMAC's computing services. There are several documents each of which describes the service offered to a particular group of users:

"NUMAC's Newcastle Service"
"The External Service"
University of Durham's "Guide to the Computing Service"

Certain conventions are used in the examples. "$" is used to denote the MTS command prefix character. Items which are to be replaced by specific data are shown in lower case, all upper case characters should be typed as shown. Finally in illustrating interactive terminal sessions, lines typed by a user are given in lower case and reponses from MTS in upper case.

Doc. Class. No.: U32
Author: J.M.Harrison
Editor: J.M.Harrison

TABLE OF CONTENTS

1 A BRIEF OVERVIEW OF MTS

The Michigan Terminal System, MTS, was developed at the University of Michigan in the U.S.A. It is a terminal oriented, time sharing, operating system which was designed specifically for a university environment. It provides both terminal and batch services through one command language.

This section gives a brief overview of MTS as it appears to a user and defines some of the more commonly used terms. This is included deliberately in this document rather than in "Introduction to MTS" as it can be appreciated much better when a user has had some preliminary experience of computing in general and MTS in particular.

1.1 ACCESS TO THE SYSTEM

Work may be submitted to MTS both conversationally, by means of a terminal connected to the computer, or by means of a batch job. Access from a terminal is called conversational mode (or terminal mode); that from a batch job is called batch mode. In conversational mode the user sits at a terminal and communicates directly with MTS through a typewriter-like keyboard. MTS processes each request as it is received and reports the results. The user can then decide what to do next. Thus the conversational mode of operation is highly interactive and such terminals are sometimes referred to as interactive terminals.

In batch mode users do not interact with their work while it is being carried out. They must preplan a sequence of instructions and submit them either as a deck of cards via a card reader or by using a terminal to enter, edit and then pass the whole sequence to the system for later execution. All batch jobs are held in a queue, the batch stream, and run only as sufficient computing resources become available.

Example of a simple MTS job:

```
$SIGNON XX22
password
$CREATE DATAFILE
$COPY *SOURCE* DATAFILE
   .
   .
Data cards
   .
   .
$ENDFILE
$LIST DATAFILE
$RUN ANALPROGRAM SCARDS=DATAFILE
$SIGNOFF
```

1.2 MTS COMMAND LANGUAGE

Whichever mode of access is used, communication with MTS is made via the Command Language. Each command, whether it is entered at a terminal or read from the batch stream, is interpreted as a unit of work to be performed. There is a large range of commands which cover a wide variety of functions. Specifications of the majority of MTS commands are included in Appendix A: in addition, the capabilities of those most commonly needed are illustrated in the various sections of the main text.

The command language, its use and the action of each command is essentially the same for both batch and conversational modes. In the latter, however, MTS replies after most commands confirming what it has done or prompting for further information. Wherever a command offers a choice of options, the most plausible or frequently required version is chosen as the default option Whenever a command does not specify a value for a particular option, the default is assumed, so that work may continue. Each command begins with a special character, a prefix character, which distinguishes MTS commands from editor commands, program instructions or data, e.g.

```
$COPY MYFILE
```

The character used is the primary currency symbol - hex '5B' and EBCDIC card code 11-3-8, which is the '#' in the United Kingdom and the '$' in North America. In this document the '$' is used throughout to avoid confusion. If necessary an MTS command may be continued into the following record by terminating the present one with a hyphen, "-". The latter, known as the continuation character, may be changed by means of the SET command.

1.3 MAIN STORAGE

Each terminal session or batch job is allocated its own storage on demand, separate from that of other sessions or jobs. The MTS system uses the concept of virtual memory which allows the use of more addressable memory than is physically available in the main storage of the computer. Virtual storage is allocated in units called pages, where one page is 4096 bytes (characters), which are kept on fast access storage units such as a fixed head disc when not required in main storage. These storage units are referred to as backing store. The MTS system breaks the job into pages and manages the transfer of pages between main and backing store. This process, called paging, is hidden from users, to whom all pages appear to be permanently resident in main storage. Active users thus compete for main storage as well as for the central processing unit, CPU, in which computation occurs.

1.4 FILES AND DEVICES

A file is a named unit of backing store, which refers to a collection or set of information. Files are classified in two ways: according to the way the information is organised within them and also to their accessibility. The various types are defined in Section 3.1. MTS provides a number of commands to manipulate files: they can be created, truncated, listed, copied, renamed, emptied and destroyed.

Information is transferred between users and computer backing store by means of card readers, line printers, terminals, magnetic tapes etc. These are referred to collectively as devices and each is given a unique device name. For example the lineprinters are referred to as PTR1,PTR2 etc. All devices are normally referenced by means of general pseudo-device names since a user does not know, and indeed should not need to know, which specific lineprinter will be used by a particular job. Such pseudo-device names are very similar to file names and may be used much the same way: for example

$COPY MYFILE *PRINT*

will copy the contents of the file named 'MYFILE' to any suitable, available line-printer, designated by the pseudo device name *PRINT*.

$COPY *SOURCE* MYFILE

will copy information from the job's main input stream - normally the terminal in an inter-active session or the card-reader in a batch job - into the file named 'MYFILE'.

Since file-names, device names and pseudo-device names are alternative ways of referencing data in MTS, they are referred to collectively as FDnames. Thus the COPY command has two FDnames as its parameters; information is read from the file or device specified by the first parameter and written to that specified by the second.

1.5 EDITING

An editor is a set of instructions which can be used to enter data into a file or to modify data already there. The University of Michigan editor is entered by means of the command:

$EDIT filename

The editor acts as a separate subsystem within the MTS Command Language. While using it at a terminal, the user will be prompted with a specific editor command prefix character, hex code '7A' usually printed as :, which serves as a reminder that the session is in edit mode.

Further details about the editor are given in Section 3.9.

1.6 PROGRAM EXECUTION

The most powerful command, and the one most often used, is the RUN command, which causes a program to be loaded and executed. Connection of logical I/O units to the program is carried out via parameters to the RUN command, e.g.

```
$RUN  MYPROG  SCARDS=DATA1  SPRINT=RESULTS1
```

From the viewpoint of the system, compilers and interpreters are programs which can be executed via the RUN command just like ordinary user programs. Sections 4 and 5 deal with running programs in MTS.

1.7 DEBUG MODE

The execution of a program may be monitored by using debug mode. Like the editor this is a command subsystem: it is entered by means of the DEBUG command and has the prefix character '+', hexadecimal code '4E'. The user can set breakpoints and step through the program; variables and instructions can be displayed and modified symbolically. Section 5 discusses a variety of program development aids including DEBUG.

2 CONTROLLING ACCESS TO MTS

The use of MTS has to be controlled to prevent abuse of the facilities offered and to share out the available resources amongst users. Since it is a terminal system, where many users submit work at sites remote from the computer, the control must be exercised by programs. Details about who may use the computing service and the conditions under which it may be used are given in the "NUMAC Service Documents".

This section discusses four aspects of the control mechanisms: identifiers by which the system recognises authorised users; the password which serves as a double-check to prevent the misuse of an identifier's resources by unauthorised persons; the accounting routines which control the proportion of the total resources available to each user; and some administrative programs which allow a user to assess the system's workload and the progress of his jobs.

2.1 IDENTIFIERS

Each user is assigned a four character identifier by which the system recognises that he or she is authorised to use the computing facilities. The identifier is also used to differentiate the work of individual users for accounting purposes.

Each identifier is associated with a particular project. A project is simply a group of one or more identifiers which have some common properties for accounting purposes: they may belong to users who are at the same establishment, or are funded from a particular research grant, or attending the same course. Each project has its own project number or identifier upon which resource control and usage reports are based. Under certain circumstances a project director may request permission to use the ACCOUNTING command to allocate resources among the user identifiers in that project. Project numbers may also be used to share programs or data amongst members - see Section 3.2.

The user identifier must be included as part of the $SIGNON command at the beginning of every MTS job or terminal session and, preceded by "Q", on the job card of each batch job, as described in NUMAC Document: "Introduction to MTS".

Example of an MTS Batch Job:

```
S8                QXX12               DECK
$SIGNON XX12 TIME=3S PAGES=50
SESAME
$CREATE MYDATA
$COPY *SOURCE* MYDATA
(data)
$ENDFILE
$LIST MYDATA
$RUN MYPROG SCARDS=MYDATA
$SIGNOFF
```

2.2 PASSWORDS

Associated with each identifier is a password which serves as a double-check to prevent unauthorised people from using the identifier and thus its files and resources. When an identifier is first assigned, its password is the five characters "BLANK". This should be changed as soon as possible using the command

$SET PW=password

where "password" may consist of 1-12 non-blank characters. This command changes the password only after a session has been ended by executing a SIGNOFF command.

The system stores an encoded form of the password in the accounting record for each identifier. Whenever a SIGNON command is entered, the system encodes the accompanying password and compares the result with that stored in the accounting record.

Please choose a password which you will remember easily. The algorithm for encoding passwords is nearly irreversible; hence it is not possible for anyone, including NUMAC staff, to recover a lost password. If you do forget your password you will have to apply in person to the Program Librarian at Newcastle or your local site co-ordinator for it to be reset to "BLANK". However you will need to prove that you have a right to use the identifier.

Please take steps to safeguard your password by concealing it. The following hints may help.

The password should always be punched on a separate card which is placed after the SIGNON card.

While punching the password card, turn off the "AUTOPRINT" switch on the punch so that the card is not interpreted.

This makes it more difficult to read the password at a glance.

Most terminals either do not echo the password, or overprint it so that it cannot be read. Some however, like the ITT 3210's in Newcastle, do echo it. This can be prevented: the operating instructions beside the terminal should describe how to do this.

Finally we recommend that you should change your password every two or three months.

Programs which perform further security checks may be included in a <u>sigfile</u>: see Section 4.5.2.

2.3 RESOURCE CONTROL

Under MTS the use of computing resources such as the amount of machine time, disc space, or graph plotting can be both monitored and controlled. The computer resources used by an identifier can be limited at three levels: in the accounting record, for each batch job or terminal session, or for the execution of individual programs within such a job or session.

In its accounting record, each identifier has at least three limits: money, file space and an expiration date. The current procedures for allocating and charging for resources are described in the "NUMAC Service Documents" together with the various resources which an individual job may request. As a reminder to users the latter resources are also displayed as a table on the inside back cover of alternate issues of the NUMAC Newsletter which is sent to every registered user. The table gives the maximum amount of each resource which may be requested and the default value. The principal resource which each job must request is CPU time expressed in either seconds or minutes - seconds are assumed if a unit is not given. Most jobs will also need some lineprinter output. These two requests are referred to as the job's global time and page limits. "Introduction to MTS" described how to request these resources via the SIGNON command:

```
   $SIGNON CLX9 TIME=2S PAGES=10
or $SIGNON CLX9 TIME=2M PAGES=50
```

Remember that, at present, $SIGNON CLX9 is equivalent to the first example above as the current default values for TIME and PAGES are 2S and 10 respectively.

The job is begun only if there are sufficient funds remaining to cover the resources requested; otherwise it is rejected with the message:

... YOU HAVE RUN OUT OF MONEY

Once its global time limit is exceeded, a batch job is terminated with the message:

GLOBAL TIME LIMIT EXCEEDED [AT xxxxxx]

where the location xxxxxx is printed only if a program was being executed when the interrupt occurred. Exceeding the page limit produces a similar message:

GLOBAL PAGE LIMIT EXCEEDED [AT xxxxxx]

If the job is a terminal session, a global time limit is still requested, either explicitly or by default, as part of the SIGNON command. When the limit is exceeded the balance of funds is recomputed. If it is still positive, and there was no explicit time limit given with the SIGNON command, a new time limit is set, equal to the default, and the job continues. If an explicit limit was given, the "global time limit exceeded" message is printed as before. However when the balance of funds becomes negative the job is interrupted with the warning message:

YOU HAVE RUN OUT OF MONEY

After either message, a new default global time limit is given, any negative balance of funds is printed and the user is prompted to continue the session. The warning message is repeated thereafter at the end of each default allocation of time.

The system imposes no default time limits on the execution of individual programs. However the user can specify such a <u>local</u> time limit on any RUN, RERUN, LOAD, DEBUG, RESTART and START command, e.g.

$RUN *SPSS TIME=20S

When the local time limit is exceeded a message stating this is printed, namely:

LOCAL TIME LIMIT EXCEEDED AT xxxxxx

2.4 MONITORING RESOURCES

There are a number of administrative programs and an MTS command which can help users to keep track of their resource usage, jobs and the system's workload see also Sections 4.8 and 3.11.

The program *STATUS prints information about the identifier's current accounting status. An abbreviated form is printed at the terminal unless PAR=FULL is included in the RUN command:

```
$run *status
EXECUTION BEGINS

STATUS OF XXX1 AT 02/16/78 12:33:00  USED     MAXIMUM  REMAINING

CUMULATIVE CHARGE          ($)        416.38   742.11    325.73
PERMANENT DISK SPACE       (PAGES)        68       70        02
CURRENT SIGNONS                            1
CUMULATIVE CONNECT TIME    (HR)        58.60

EXECUTION TERMINATED
```

3 FILES

3.1 TYPES OF FILE

In MTS, files are classified in two ways - by organisation and by accessibility. There are two different types of organisation: line files and sequential files.

A line file is an ordered set of zero or more lines, where each line consists of from 1 to 32767 characters or bytes. Each line in the file has associated with it a line number, which is not part of the line. Any line in the file may be referenced, deleted or changed through its line number. "Introduction to MTS" dealt almost entirely with this type of file as it is by far the most widely used.

A sequential file consists of a sequence of records, each containing 1 to 32767 bytes, which are accessed in the order in which they are stored.

Every file, whatever its organisation, is also classified according to its accessibility as a public or a private file and the latter are further subdivided into permanent or temporary (scratch) files.

Public files contain components of the system such as language translators, applications packages and administrative programs. All such files have file names beginning with an asterisk, "*". They may be read by all users' jobs but are protected against modification. Appendix B consists of a complete list of the public files available, while MTS Volume two: "Public Files Descriptions" contains a complete description of each one. Some of those files which contain useful administrative programs are described in more detail elsewhere in this document.

Private files belong to a specific identifier and may be accessed only by the owner or such other users as he or she permits to do so - see Section 3.2.

Permanent private files must be created explicitly by the user. Once created they exist until the user explicitly destroys them. Their filenames consist of 1-12 characters which may be letters, digits or special characters except for the following:

, ; : ( ) @ + = ' " ? & blank

The name must not begin with "*" or "-". Users are recommended to use only the alphanumeric characters, A-Z and 0-9, "." and "_". The following names are all valid:

MYFILE
PROGRAM.SRC
DATA_RUN1

while these are invalid:

*BESTJOB
PROGRAM,SRC
DATA RUN1

If too many characters are given MTS will truncate the name, e.g. "PROGRAM_SOURCE" would become "PROGRAM_SOUR". If a blank is included in a name as in DATA RUN1 above the portion of the name after the blank will give rise to an error message.

Temporary or scratch private files may be created explicitly or implicitly. The latter will happen the first time a temporary file is referenced. Temporary files may be destroyed explicitly at any time during a session; all remaining ones are destroyed automatically when the user signs off. These files are distinguished by their names, which must consist of one to eight characters prefixed by a minus sign "-".

For all permanent private files the complete name of the file is:

identifier:filename

e.g. CLXX:DATAFILE. However, by default, MTS will assume that "identifier" is the same as the signon identifier, so the full form of a filename need only be used when referring to someone else's files.

3.2 CONTROLLING ACCESS TO FILES

A user can allow others to have selective access to specific files by means of the PERMIT command. The command takes the form:

$PERMIT filename how whom

(Memory Tip: Parameters are cited in alphabetical order - f,h,w!)

There is a considerable number of types of access which can be given: these are listed under the command name in Appendix A. Two of the most useful forms are:

$PERMIT filename

which allows all other users read only access to the file named, and

$PERMIT MYFILE READ CLX2

which allows user identifier CLX2 read only access to file MYFILE. 'Read only' access allows another user to specify the file name in such commands as LIST, COPY and SOURCE but not to alter or supplement the information already in the file.

To remove access again the means of access is given as 'NONE', e.g.

$PERMIT MYFILE NONE CLXO

or just

$PERMIT MYFILE NONE

The latter removes access from all other identifiers.

Where a user has a file which it is particularly important to preserve, it is possible to remove write access from himself.

$PERMIT MYFILE READ ME

However a user can always alter access to his or her own files so the situation can be reversed by:

PERMIT MYFILE UNLIM ME

where "UNLIM" stands for unlimited access and "ME" is a pseudonym for the identifier under which the session is currently running.

Where a group of users are all working together and have identifiers belonging to a project, files can be shared with all the group by specifying the project identifier in the PERMIT command. Example:

$PERMIT MYFILE READ PROJNO=C9Z9

would allow all user identifiers belonging to project C9Z9 to read MYFILE.

3.2.1 Program Access To Files

For most applications involving the use of shared files, it is usually sufficient to be able to permit files to be accessed in a particular manner (read access, write access, etc.) as described in the previous section. However, occasionally, it is necessary to control the access to a file more precisely, e.g. to allow a data file to be read only by a specific program or to allow colleagues to run a program but not to list or edit it. This can be done by means of <u>program keys</u>.

Each MTS file has exactly one program key associated with it: this is a code comprising a string of up to eight characters. Access to a file is controlled by comparing the program key of a running program or MTS command processor against that stored in the permission list of the file being accessed. For example: suppose user CXXX has a file DATAS1 which he wishes to be read only by his program PROG. The program key of program PROG is set using the CONTROL command:

```
$CONTROL PROG PKEY=UPDATE
```

In the computer system the program key is stored with the current signon identifier as a prefix, i.e. CXXX:UPDATE. The matching program key is then added to the access information of the file "DATAS1" using the PERMIT command:

```
$PERMIT DATAS1 READ PKEY=CXXX:UPDATE
```

If another user tries to access "DATA" he could only do so by using the program in file CXXX:PROG and then only if user CXXX has given him access to "PROG".

Each of the MTS command processors has a program key. They are generally of the form "*MTS.command"; for example RUN has the key "MTS.RUN". The exceptions are:

$ACCOUNTING	*ACC
$CALC	*CALC
$DEBUG	*SDS
$EDIT	*EDIT
$SDS	*SDS
$SYSTEMSTATUS	*SSTA
$RERUN	*MTS.RUN

To allow a program to be "run-only" use the following command:

```
$PERMIT filename READ PKEY=*MTS.RUN
```

which may be abbreviated to:

```
$PERMIT filename RUN
```

This allows the RUN processor (that is the loader) to read the program, but nobody else (except the owner) can access the file in any way. For example, the program cannot be listed, copied or debugged. Such "run-only" programs must be completely self-contained: they may not be concatenated with other files; a portion of the file may not be specified by means of line number ranges; and additional input or alterations may not be made during loading.

A detailed description of program keys is given in Appendix I of MTS Volume One: "The Michigan Terminal System". We recommend that you read this before using them.

3.3 DEVICES

As was pointed out in Section 1.4 pseudo-device names are used to enable users to refer to devices without needing to know the specific identifier of the physical device which will be used. A pseudo-device name may consist of from one to fourteen alphanumeric characters, preceded and followed by an asterisk. Users may define pseudo-devices of their own, for example when using magnetic tapes - see "Introduction to Magnetic Tapes".

Certain pseudo devices are part of MTS and have standard names. Their names and functions are as follows:

- *MSOURCE* Master Source: the keyboard of a terminal, or the card reader when in batch mode. It is the primary source of commands.
- *MSINK* Master SINK: the display of a terminal, or the line printer when in batch mode. It receives printed output such as the listing of the job commands executed.
- *SOURCE* The current source of input. Initially it is the same as *MSOURCE* but may be changed by the SOURCE command - see Section 4.5.1.
- *SINK* The current destination of output. Initially it is the same as *MSINK*, but it may be changed by the SINK command.
- *PRINT* A lineprinter.
- *PUNCH* A card punch.
- *DUMMY* Information written to this device is discarded, and attempts to read from it result in an end-of-file condition.
- *BATCH* This submits jobs to the HASP queues - see Section 4.5.

The SOURCE and SINK commands may be used to alter the devices to which *SOURCE* and *SINK* refer. The ability to change *SOURCE* allows a user to store a series of frequently used commands in a file and incorporate them into a job conveniently when required - see Section 4.5.1.

NOTE: Certain of these pseudo-device names refer to devices at the central site which have only limited capacity; this applies to *PRINT* in particular. Such pseudo-devices may be accessed simultaneously by only a few jobs out of the 80-100 which may be running simultaneously. Please remember this and order your work so that a pseudo-device is held for the minimum necessary period. See also Sections 4.5 and 4.7.2.

3.4 CREATING FILES

All files except temporary files must be explicitly created using the CREATE command:

```
$CREATE filename [SIZE=size] [TYPE=type]
```

The default values for the parameters are SIZE=1P for private files, SIZE=10P for temporary files, and TYPE=LINE. So to create a sequential file a type parameter must be given:

```
$CREATE filename TYPE=SEQ
```

'P' is an abbreviation of page which is a unit of memory which will hold 4096 bytes or characters of information. This is approximately 50 fully punched cards or 30-50 lines of printer output. The size parameter is treated only as an estimate of the final file size: if the file becomes full during input, MTS will try to extend its size in increments of one page. Obviously for very large amounts of information, it is extremely inefficient to create a file of default size and expand it page by page. If you suspect that a file is going to be larger than the default please specify a size parameter.

Temporary files do not need to be created explicitly: they will be created the first time the file is referenced in a command. However the default size is 10 pages and if you suspect this will not be large enough to hold the information - as it frequently is not with output files for instance - then the CREATE command should be used, e.g.

```
$CREATE -OUTPUT SIZE=40P
```

3.5 PUTTING INFORMATION INTO A FILE

There are several ways of putting information into a file; which you use depends on the circumstances and to some extent on personal preference.

When a large amount of data is being read from a batch job the recommended method is to use the COPY and ENDFILE commands, e.g.

```
$CREATE EXPT1 SIZE=20P
$COPY *SOURCE* EXPT1
            .
     <data cards>
            .
$ENDFILE
$TRUNCATE EXPT1
```

After reading information into any file which has been explicitly created with an estimated size, a TRUNCATE command should be issued to free any unused space at the end of the file.

At a terminal, we recommend that the Michigan editor be used to input data to a file, as this allows mistakes to be corrected as you go along.

Example:

```
$create testdata
$ed testdata
: i
?  First data line
?  Second daat
?  (null line to end insertion)
: c ;daat;data line;
:     2      SECOND DATA LINE
: i
?      .
? (rest of data)
?      .
?  (null line)
: stop
$
```

Once information has been read into a file, a printed listing of its contents is useful both to cross-check that the information is correct and as a record for later reference. The LIST command will provide this:

```
$LIST filename          ( in batch or - )
$LIST filename *PRINT* ( at a terminal )
```

The LIST command should be used in preference to the COPY command for two reasons. It prints the line number associated with each line on the left-hand side of the listing (useful when editing) and it does not use the first character of each line as carriage control - see Section 4.7.1. However LIST uses the line numbers when printing the file so please read the next section carefully: especially the warnings at the end.

3.6 LINE NUMBERS

When a line file is stored in MTS, the line number of each line within the file is kept in an associated index. This enables the action of an MTS command to be restricted to only a part of the file by qualifying the filename with a <u>line number range</u>. For example

$LIST FRED(5,25)

will list only lines 5 to 25 inclusive of the file FRED.

"Introduction to MTS" described clearly how to specify a line number range and the meaning of the various parameters. The general formula is repeated here as a convenient reminder:

filename(b,e,i)

where "b" is the beginning line number, "e" is the ending one and "i" is the increment by which "b" progresses to "e".

A line number has two associated representations. Externally it appears to the user as a number in the range

-99999.999 to +99999.999

Internally the number is stored as an integer in the range

$$-10 < I < 10$$

and it can be used within a program in this form. The relationship between the internal (i) and external (e) forms is

$$i = 1000e$$

The line numbering of a file can be specified implicitly or explicitly. It is specified implicitly in the first example in Section 3.5 where the command

$COPY *SOURCE* EXPT1

will result in the data following being read into file EXPT1 starting at line number +1.0 with subsequent line numbers incremented by 1.0 each time. For the majority of jobs these defaults will be quite satisfactory. However just occasionally a different arrangement of line numbers may be required: these must be stated explicitly. For example program source statements might be read into a file starting at line 20 to allow MTS commands to be added at the front and with an increment of 10 to allow for program amendment and development:

```
$CRE TASK.PSRC
$COP *SOURCE* TASK.PSRC(20,,10)
.
(program source statements)
.
$ENDFILE
```

Warning: Please remember the following points:

The default value for the begining line is +1.0 - this may not be the first line in a file. It is most important to be aware of this when using LIST or COPY commands. To denote the first or last lines of a file use the words 'FIRST' or 'LAST' as the line number; or the abbreviations '*F' and '*L' respectively. If no line number increment is given, every line in the file is accessed. However, if an increment is given, some lines may well be passed over. This is particularly important when listing. To list all the lines in the file TASK.PSRC mentioned in the last example use:

```
$LIST TASK.SRC(FIRST)
```

The command:

```
$LIST TASK.SRC(20,,10)
```

may have the same effect initially but as soon as the file is modified, there is the possibility of omitting any lines which have been inserted at the beginning or amongst the original statements.

3.7 CONCATENATION

Where information is contained in more than one file it is possible to indicate to MTS that the files are to be concatenated. This can be done explicitly by listing the filenames, linked by '+' signs, in the order in which they are to be accessed, e.g.

```
$COPY A+C+D+B TO E
```

or

```
$COPY A(1,20)+B+A(21,40) TO C
```

Alternatively, files can be concatenated implicitly by including the CONTINUE WITH command in a file:

```
$CONTINUE WITH filename RETURN
```

This command transfers control to the file named. Once the end of this file is reached control returns to the original file begining with the line after the one containing the CONTINUE WITH command. If the RETURN keyword is omitted control returns to the next line in the MTS command stream, *MSOURCE*. Any number of files may be concatenated by a mixture of the two methods. MTS deals with each as it is encountered.

3.8 I/O MODIFIERS

These are codes of one to four characters which indicate how the information in a file is to be read or written. They are appended to the file name by means of an at sign, "@". The modifiers act as switches, each having two forms to indicate the opposing states of a particular effect. Descriptions of some of the most useful modifiers follow:

@S or @I
This modifier indicates that the information in the file is either to be accessed sequentially or indexed. The default state is S. @I is principally used to indicate that the original line numbering is to be retained when copying the file, e.g.

```
$COPY A TO B@I
```

Remember that the command:

```
$COPY A TO B
```

results in the lines from A being renumbered as they are copied to B, starting with line +1.0 and continuing in increments of 1.0.

@TRIM or @¬TRIM
The default is @TRIM which causes all but one of the trailing blanks to be deleted from the end of the lines in a file. Example:

```
$COPY *SOURCE* MYFILE@¬TRIM
```

will cause data to be copied from *SOURCE* to the file MYFILE including all trailing blanks at the end of each line.

Note: When lines are read into a line file, MTS will remove all but one of any trailing blanks at the end of each line. However if there are no trailing blanks none will appear in the file.

@IC or @¬IC
The default is @IC. The effect of @¬IC is to cause lines comprising an MTS CONTINUE WITH command to be treated as data rather than a command. This is useful when listing or editing such lines, e.g.

$LIST MYPROG@¬IC

will cause the contents of the file MYPROG to be printed but any CONTINUE WITH commands will not be obeyed, merely listed.

@CC or @NOCC
@CC requests that the first byte of each line be treated as a carriage control character if it is a legal one. See Section 4.7.1. @CC is the default on the line printer and at typewriter terminals; otherwise it is @NOCC.

@MCC or @¬MCC
@MCC is used with files directed to the lineprinter or to a terminal to indicate that machine rather than logical carriage control is used. The default is @¬MCC.

@PEEL or @¬PEEL
On input @PEEL causes the first part of a line to be taken as a line number and peeled off. Example:

$RUN MYPROG SCARDS=INPUT@PEEL

The lines which the program reads from the file INPUT have line numbers at the front of each line. These will be recognised and stripped off by the system. On output, the modifier requests that the line number written is to be returned to the program doing the writing. @¬PEEL is the default.

@UC or @LC
This requests upper case conversion. The default is @LC, requesting that all alphabetic characters should be transmitted unchanged. Example:

$RUN MYPROG SCARDS=DATAS1@UC

will cause data for MYPROG to be read from the file DATAS1 converting any lower case characters to upper case ones. Please note that when entering data at some terminals, the software supporting the terminal forces characters into upper case - see Section 4.4.3.

3.9 EDITING: CHANGING THE CONTENTS OF A FILE

This section is concerned with editing files, i.e. altering part of the information within a file while leaving the rest intact. It applies mainly to line files. The contents of a sequential file may be altered only if the replacement is the same size as the original data or by appending lines to the end of the file.

3.9.1 The File Editor

This is a subsystem of MTS commands which enables alterations to be made by line number or by context. It is described in NUMAC documents:

"An Introduction to the University of Michigan File Editor"

"The Michigan Editor"

The command subsystem is entered by issuing the command:

$EDIT filename

There are facilities within the editor for safeguarding the contents of a file while it is being edited. The editor has a CLOSE command - see Section 3.9.3 - and an UNDO command. The latter reverses the effect of the immediately preceding editor command. The editor also has checkpoint/restart facilities to allow whole sections of an editing session to be reversed.

When used from a 3270 terminal the editor has extra facilities for character and line editing by means of the terminal's screen cursor and special function keys. This "visual mode" of the editor is described in NUMAC Document: "The Michigan Editor". Visual mode offers very powerful facilities for rapid editing of files. However like all powerful and sophisticated tools it can create havoc if misused. Please heed the following warnings.

> Do not attempt to edit a valuable file in visual mode until you are fully familiar with the 3270 terminal and the effect of all its function keys <u>in editing mode</u>. (These are not necessarily the same as <u>in MTS command mode</u>.)
>
> At NUMAC the 3270 terminals display upper case letters only. Care must be taken when editing information which is a mixture of upper and lower case letters in visual mode. Always issue an editor SET command to specify upper and lower case input
>
> $SET LC=ON
>
> Further this command must be repeated every time a 'stop' command is executed by the editor.

3.9.2 How Changes Are Effected

Unlike those of some other operating systems, the MTS editor changes files as the commands are entered: it does not make a working copy of the file incorporating the changes as it does so. It is very important to remember this and to have some appreciation of the mechanisms involved and their implications.

When a file is first referenced it is 'opened', that is the first few physical blocks of information are read from the disc and placed into buffers (blocks of storage in virtual memory). This is done for reasons of machine efficiency. When alterations are made to the file, the amendment is made immediately if the altered lines physically and logically can replace the original ones in the buffer. If not, an existing buffer must be emptied, the appropriate physical block read from the disc into the buffer and the line fitted into place. Whenever information read into a buffer is altered, a flag is set. The updated information in the buffer is written back on to secondary storage only when necessary. Only when the user has finished updating a file does MTS finish writing out on to disc all virtual memory buffers that have changed. Only at this point is the file safely on disc in a consistent and up-to-date state. This final process is called "closing the file".

If the system fails abruptly while a file is being updated some of the changes (not necessarily the most recent) may not have been written back to disc. In most cases the file will still be usable but some laborious checking may be needed to identify which changes were not effected. Only occasionally is a file so damaged that it can no longer be used - see Section 3.9.4. Should this happen NUMAC advisory staff must be consulted. By far the most common cause of 'damage' to files however is the user. Either a mistake is made in giving editing instructions or the user has second thoughts and regrets an editing session.

3.9.3 Safety Precautions When Editing

There are precautions which can be taken to safeguard a file against such eventualities but they must be taken before the editing commands are issued. A limited amount of remedial action can also be taken against mistakes made during editing but once the session is ended, the only remedy is recreation of the file or laborious re-editing. When altering large and valuable files <u>please</u> follow this recommended code of practice:

Make a copy of the file and edit that. This is the simplest precaution and guards against all eventualities.

When making a lot of amendments to a file, use the editor's CLOSE command to force MTS to 'close the file' at more frequent intervals than it might otherwise do so. The command causes all outstanding changes to be written out to disc so, in the event of a system failure, all alterations up to the last CLOSE command will have been effected.

When using the editor for large updates, take advantage of the checkpoint/restart facilities. These will help with user mistakes but not system failures.

Finally, but most important, keep an accurate and orderly record of what has been done. Section 3.9.5 describes some further aids for this.

3.9.4 Damaged Files

If you suspect that a file has become corrupted, e.g. editor commands produce unexpected results such as lines of a file appearing in the wrong place or being overwritten, stop editing at once! Further attempts to edit the file could cause even greater damage. Instead investigate the problem using the program in public file *VALIDATEFILE. This program checks the structure of a line file but not its contents: it checks that all lines are the right length and in the right place according to the file's directory. The following example illustrates its use.

```
$r *validatefile
EXECUTION BEGINS
TYPE NAME OF FILE TO BE VALIDATED
cxyz:lssourcev2
FILE: CXYZ:LSSOURCEV2
THIS FILE IS INCONSISTENT
TYPE "EXPLAIN", "LINES", OR "STOP"
explain
TO PREVENT FURTHER DAMAGE DO NOT WRITE IN THIS FILE
THE FILE MUST EITHER BE RESTORED FROM A COMPUTING CENTER
SAVE-TAPE OR COPIED TO ANOTHER FILE.  WITH EITHER METHOD
SOME INFORMATION MAY BE LOST.  TO HAVE THE FILE RESTORED
CONTACT THE PROGRAM LIBRARIAN AT NEWCASTLE.
IF YOU WANT TO ATTEMPT TO FIX THE FILE YOURSELF,
FIRST COPY IT TO ANOTHER FILE USING A COMMAND SUCH AS:
$COPY BADFILE(FIRST)@¬TRIM@¬IC NEWFILE@INDEXED@¬TRIM
THEN EXAMINE "NEWFILE" FOR ERRORS.
PLEASE CONTACT THE ADVISORY SERVICE IF YOU HAVE ANY
QUESTION OR DOUBT.
TO GET A LIST OF LINE NUMBERS THAT ARE LIKELY
TO BE BAD, TYPE "LINES", OTHERWISE TYPE "STOP".
Lines
    300.500
    308.100
    308.120
TYPE NAME OF FILE TO BE VALIDATED
mts.us.g
FILE SEEMS OK
TYPE NAME OF FILE TO BE VALIDATED
$endfile
EXECUTION TERMINATED
```

3.9.5 Identifying The Changes In A File

When editing large and important files, especially on a regular basis, it is essential to keep adequate records of the changes. There are some useful programs available to help with this.

The program in public file *APC is an all purpose compare program which is very useful for checking that two files are identical.

Further programs are useful for identifying the differences between two files. The program in *UNEDIT can be used to produce a listing of the differences between two files and also to produce a set of edit commands to convert one to the other. This can be very useful where large edits are carried out or where source programs used in production are frequently updated. A second pair of programs, which will help with the latter task if the source code is stored as card decks, are *DOWNDATE and *UPDATE.

Descriptions of all these programs may be found in MTS Volume Two: "Public File Descriptions".

3.10 THE FILE ARCHIVE SYSTEM

There is only a limited amount of public disc space available to MTS . Periodically this becomes full and free space must be created again, either by users voluntarily removing some of their files, or by NUMAC staff enforcing the transfer to magnetic tape of files which have not been accessed for some time. In previous versions of MTS at NUMAC the latter process, known as <u>dredging</u>, was the usual method by which disc space was released. Dredged files could only be restored by NUMAC staff.

With the advent of Distribution 4.0 of MTS, dredging has been merged with a new archive system. This allows more flexibility in file movement; users are able to use the *ARCHIVE program to archive and restore their own files. The question of data security and the operational and administrative aspects of the archive system are dealt with in the "NUMAC Service Documents". All users should read the relevant part very carefully and ensure that they understand its implications for their own work. The remainder of this section describes the *ARCHIVE program and its use.

The program in public file *ARCHIVE allows users to control the movement of files between the public discs and the MTS archive system. Although archived files are stored on magnetic tapes, users need not concern thenselves with the tape handling involved as this is done automatically by the archive system. Any user, including the owner, with rename/destroy access to a file may archive it but only the owner may restore it to public disc space or destroy it in the archive. Movement of files between the archive and the public discs is not carried out immediately instructions are issued. Requests are batched up, collated and executed in a sequence designed to optimise the tape handling required and to achieve a reasonable compromise between response time and capacity. Normally all requests to archive a file will be completed within 24 hours. The time required to restore a file depends, on average, upon the time it has been in the archive; files newly entered should be recoverable within hours while files of longstanding residence may take several days.

Execution of the *ARCHIVE program is initiated by means of the RUN command:

$RUN *ARCHIVE

The user is then prompted for archive commands. These are entered one at a time and the program performs some checks on the validity of the command and the accessibility of the files named. When the end of a batch of commands is indicated, the whole batch is passed to the archive system management program for execution at some later date.

*ARCHIVE is self-documenting: once the program is running the command 'HELP' will print a brief description of the program and a list of available commands. 'HELP <command name>' will provide further details about a specific command. The commands available are:

CANCEL <cmnd. no>	cancels specified command in current batch.
DESTROY <filename>	deletes file from archive.
DISPLAY	displays names of all your archived files.
END	sends current command batch for execution.
HELP	documents *ARCHIVE.
MTS	suspends *ARCHIVE and returns to MTS. $RESTART will cause *ARCHIVE to resume.
RECAP	lists the commands in the current batch.
RESTORE <filename>	schedules file for restoration later.
SAVE <filename>	saves file for archiving later.
STATUS <filename>	supplies *ARCHIVE status information for specified file. '*ALL*' gives information for all your files.
STOP	as 'END' but also terminates archiving session.

An example of the use of *ARCHIVE is given in Section 3.11.

3.11 FILE MANAGEMENT

Having created files, stored information in them and modified this as necessary, we now need to consider a number of commands which can be used to manipulate or manage files as single entities. These commands are all described in NUMAC Document: "Introduction to MTS". They are revised here for convenience and as an opportunity to recommend methods of file management. As was indicated in the preface, this document is intended for users who are investing considerable effort in developing their own programs, quite often to process data that has been collected with great care and labour. It is users' responsibility to protect their work against accidental loss due either to system failure or, more frequently, their own mistakes. A small amount of effort made to ensure good file management can avoid delays and frustrations caused by running out of resources or forgetting where you are in your project. It will also be an insurance policy against the tragic loss of many months valuable work.

For the purposes of this demonstration let us consider a not untypical user who has planned(!) his project in a number of stages. In Stage I, the program in file "ANAL1.PSRC" is being developed to process the experimental data in file "RAWDATA" and various intermediate sets of results will be stored in files "RES1","RES2" and "RES3" etc. "COPY.ANAL" contains a copy of "ANAL1.PSRC", this is the version which is edited and then copied back to "ANAL1.PSRC" as an insurance against accidental loss of the only copy of the program. "COPY.RDATA" is an edited copy of the original data from file "RAWDATA": it has been corrected to remove incomplete sets of results and recording errors.

Sets of MTS commands to run a job may be stored in a file - see Section 4.5 : file "RUNANAL1" contains such a set. Such files can be made self-documenting by use of the COMMENT command, e.g. file "RUNANAL1" might start with the following lines

```
$COMMENT   THESE COMMANDS RUN THE SOURCE
$COMMENT   PROGRAM IN FILE "ANAL1.PSRC"
$COMMENT   USING THE DATA IN FILE RAWDATA.
```

COMMENT lines are printed in all listings of a file but otherwise ignored.

If the command FILESTATUS is used to survey the general situation:

```
$filestatus * summary size
```

might produce the following output:

```
ANAL1.PSRC          SIZE=18P
COPY.ANAL           SIZE=18P
COPY.RDATA          SIZE=12P
RAWDATA             SIZE=12P
RES1                SIZE=8P
RUNANAL1            SIZE=1P
   6 FILES          SIZE=69P
```

Hmm! Rather close to the limit of filespace!

The next step is to use FILESTATUS to examine the latest version of the program in file "COPY.ANAL" :

```
$filestatus copy.anal  total
COPY.ANAL SIZE=18P,MINSIZE=12P,TRUNC=18P,MAXSIZE=255P,
TYPE=LINE,ACCESS=U/N,RPM=28,IDLEDAYS=0,LINES=2081,
HOLES=209,AVLEN=32,MAXLEN=120,AVAIL=5984,MAXHOLE=255,
LASTREF=02/16/78,CREDATE=01/25/77,VOLUME=MTS001,OWNER=XXX1,
LOC=DISC,USECNT=363
```

The MINSIZE and HOLES values indicate that there is considerable unused space within the file. This can only be freed by copying it. "ANAL1.PSRC" can be emptied and used for the copy:

```
$empty anal1.psrc ok
DONE
$copy copy.anal anal1.psrc
$filestatus anal1.psrc total
ANAL1.PSRC SIZE=18P,MINSIZE=12P,TRUNC=12P,MAXSIZE=255P,
TYPE=LINE,ACCESS=U/N,RPM=60,IDLEDAYS=0,LINES=2081,
HOLES=10 AVLEN=32,MAXLEN=120,AVAIL=5984,MAXHOLE=255,
LASTREF=02/16/78 CREDATE=02/16/78,VOLUME=MTS001,OWNER=XXX1,
LOC=DISC,USECNT=2
```

The number of HOLES has decreased but TRUNCATE reveals that there is now considerable free space at the end of the file, which can be released.

```
$truncate anal1.psrc
```

"COPY.ANAL" is emptied, then "ANAL1.PSRC" is copied back so that the two copies are once more identical. Finally "COPY.ANAL" must be truncated and then this can be used for further editing.

In the same way FILESTATUS is used to examine "COPY.RDATA" and that too is tidied up, bringing its size down to 10 pages.

"COPY.RDATA" is copied into a new file, "DATA". The program *UNEDIT is used to obtain a reference listing of the editing that was carried out on RAWDATA:

```
$run *unedit 0=copy.rdata 1=rawdata spunch=-changes par=list
```

The file -CHANGES contains a set of the necessary edit commands to turn RAWDATA into COPY.RDATA As this is only required for reference just the listing is retained: the file is discarded.

The files "COPY.RDATA" and "RAWDATA" are now archived :

```
$run *archive
EXECUTION BEGINS
NUMAC ARCHIVE SYSTEM
FOR HELP, TYPE 'HELP'

1:   save rawdata
2:   save copy.rdata
3:   stop
CLXX:RAWDATA        SAVE SCHEDULED
CLXX:COPY.RDATA     SAVE SCHEDULED

END OF ARCHIVING SESSION
EXECUTION TERMINATED
$
```

The next day after the archive system has had chance to carry out the commands :

```
$filestatus * summary size
```

reveals that:

```
ANAL1.PSRC          SIZE=12P
COPY.ANAL           SIZE=12P
DATA                SIZE=10P
RES1                SIZE=8P
RUNANAL1            SIZE=1P
   5 FILES          SIZE=43P
```

Another useful facility when a file is being heavily edited is to renumber the lines. This will reduce the effort in typing fractional line numbers and ease insertion. However it is not always convenient to copy a file. There are two other methods of renumbering either part or all of a file.

Use the editor's RENUMBER command:

```
$edit data
:renumber
:stop
```

Use the MTS command RENUMBER:

```
$renumber data
```

One more point to mention at this stage is that the file "RES1" should be emptied before being reused in the next development run of the program. Do not rely on the system overwriting a file - this is sloppy workmanship, uses considerably more CPU time and could well lead to confusion.

At the end of Stage I, the work is tidied up before embarking on Stage II. First a reference listing of the file in "ANAL1.PSRC" is produced, together with the MTS commands for running it, which are in file "RUNANAL1". Lines 3-5 in the example below cause a page throw between the two listings for easy reading - see Section 4.7.1.

```
$control *print* hold
$list runanal1 *print*
$copy *source* *print*
1
$endfile
$list anal1.psrc *print*
$release *print* --- IMPORTANT!  Do not forget this line!
```

If the COMMENT command and comment statements have been fully used, these listings, together with the reference listing of the editing needed to process the raw data, should complete the documentation of Stage I. Then "ANAL1.PSRC", a <u>copy</u> of "RUNANAL1", and the files of intermediate results are archived:

```
$run *archive
EXECUTION BEGINS
NUMAC ARCHIVE SYSTEM
FOR HELP, TYPE 'HELP'


1:   save cop.res1
2:   save res2
3:   save res3
4:   save anal1.psrc
5:   save copy.runanal
6:   recap
1:   SAVE COP.RES1
2:   SAVE RES2
3:   SAVE RES3
4:   SAVE ANAL1.PSRC
5:   SAVE COPY.RUNANAL
6:   stop
CLXX:COP.RES1         SAVE SCHEDULED
CLXX:RES2             SAVE SCHEDULED
CLXX:RES3             SAVE SCHEDULED
CLXX:ANAL1.PSRC       SAVE SCHEDULED
CLXX:COPY.RUNANAL     SAVE SCHEDULED

END OF ARCHIVING SESSION
EXECUTION TERMINATED
```

Now preparation begins for Stage II. File "DATA" is no longer required and there is already an archive copy so it is destroyed

```
$destroy data ok
```

The file "RUNANAL1" will be edited to produce a set of commands to run the new, Stage II, program so it is retained but renamed to reflect its new purpose. (Do not forget to change the comments as well!)

```
$rename run.anal1 run.anal2 ok
```

A new file is created for the Stage II program and the first source statements read into it. File "COPY.ANAL" still contains the Stage I program. It is now emptied, the new program copied into it and the file truncated. FILESTATUS summarises the new situation:

```
$filestatus * summary size

ANAL2.PSRC          SIZE=5P
COPY.ANAL           SIZE=5P
RES1                SIZE=8P
RUNANAL2            SIZE=1P
   4 FILES          SIZE=19P
```

One final reminder:

The FILESTATUS command has many facilities for selecting subsets of files and for selectively printing items of file,access or catalogue information about files - see the command specification in Appendix A. For instance it can be used to list all temporary files attached to an MTS job as follows:

```
$filestatus -?
-CHANGES -CHANGE2
```

4 RUNNING WORK IN MTS

Now that we have described the basic units of the MTS system we can turn our attention to the burning question "How do I run my program?". This is answered in two parts. This section deals with an MTS job as a complete unit, i.e. a set of commands to identify the user, to describe information, to process it and finally to produce the results. For this purpose "program" includes both private ones written by users and those available in the public files such as *SPSS and *FTN - see Appendix B. After describing the way jobs are processed and how their progress may be monitored, the merits, demerits and facilities of batch and terminal modes are described. The following section of this document, Section 5, then concentrates on users' own programs and proceeds to describe in more detail how their execution may be controlled.

4.1 EXAMPLE MTS JOB DECKS

NUMAC Document: "Introduction to MTS" describes the basic construction of an MTS job. This information is revised here by means of three annotated examples of batch jobs which run the most heavily used MTS programs, namely FTN, ALGOLW and SPSS.

Example 1: FORTRAN

A Fortran program, together with the commands necessary to compile it using the FORTRAN compiler in public file *FTN, is stored in the file "PROG". The compiler is executed, being instructed to read the program from this file, and the data for the program from the following cards in the job.

```
Columns 1                 19                  40
        S8                QXXX1               DECK            (1)
        $SIGnon XXX1
        ALBERT                                                (2)
        $COMMENT FTN EXAMPLE PROGRAM FOR INCLUSION IN
        $COMMENT SECTION 4 OF THE MTS USERS GUIDE
        $CREATE PROG                                          (3)
        $COPY *SOURCE* PROG
              READ (5,100) X
        100   FORMAT(F10.5)
              U=SQRT(X)
              WRITE(6,200) X,U
        200   FORMAT(1H ,2F10.5)
              STOP
              END
        $ENDFILE
        $TRUNCATE PROG                                        (4)
        $RUN *FTN SCARDS=PROG                                 (5)
        $RUN -LOAD 5=*SOURCE*                                 (6)
        21.67035                                              (7)
        $ENDFILE                                              (8)
        $SIGNOFF                                              (9)
```

(1) This is a batch job so it must start with an jobname card followed by the $SIGNON card.
(2) The password should be punched on a separate card to prevent its being listed on the front page of the output.
(3) Create a file and then copy the program into it.
(4) The file may not be full so tidy up any free space at the end.
(5) Compile the program using *FTN which will leave the compiled program in file -LOAD.
(6) Load the compiled program and begin execution, reading data for channel 5 from the following lines of the job (*SOURCE*).
(7) Data on the following cards as the RUN command expects.
(8) End-of-data is signalled.
(9) Job finished.

Example 2: ALGOLW

An ALGOLW source program already stored in file "PROGAW" is to be compiled. By default the compiled program will be left in temporary file -AWLOAD. This will then be loaded together with some subroutines from a library file. The program is run using data from one file and storing the results in another.

```
COLUMNS  1                19                   40
         S8               QCXD9                DECK
         $SIGNON CXD9
         SESAME
         $CREATE LOADAW SIZE=15P
         $CREATE RESULTS SIZE=10P                                  (1)
         $EMPTY RESULTS
         $EMPTY LOADAW
         $RUN *AW SCARDS=PROGAW SPUNCH=LOADAW                      (2)
         $RUN LOADAW+*NAG SCARDS=DATA1 SPRINT=RESULTS              (3)
         $SIGNOFF
```

(1) The file RESULTS is created with a suitable size parameter. The file is then emptied. If the job is rerun either by the operators or the user, this precaution will ensure that the file contains output only from the latest run, thus avoiding confusion.

(2) The ALGOLW program is compiled. The compiled program will be left in file "LOADAW". If you do not wish to preserve the compiled program the SPUNCH assignment may be omitted. By default the compiled program will then be left in temporary file "-AWLOAD": this filename is then given with the next RUN command.

(3) The compiled program is loaded together with the desired routines from the NAG subroutine library. Execution begins.

Note that in this example the results are stored in a permanent file. This enables them to be examined from a terminal, where the editor can be used to correct the source statements in "PROGAW", the file "RES and the program rerun. On many occasions during program development, the program output is used once, then discarded (i.e. added to the barricade of paper between the programmer and the outside world!) : save paper and effort by not printing files unnecessarily. When it is necessary to print the output directly line (3) can be changed to:

```
$RUN -AWLOAD+*NAG SCARDS=DATA1 SPRINT=*PRINT*
```

However please remember that we have only limited lineprinter capacity at the central site in Newcastle so please do not assign output to *PRINT* when running programs interactively from a terminal - see note at end of Section 3.3.

Example 3: SPSS

This job runs the program SPSS using raw data from the second file of a magnetic tape whose tapename is CXX1D1 and whose volume serial number is BUR001.

```
Colums   1                        19                    40
         S8                       QXXX1                 DECK
         $HOLD  MAG.TAPE CXX1D1  READ ONLY                            (1)
         $SIGNON  CXX1   T=30S
         SURVEY
         $MOUNT  CXX1D1  *T*  VOLUME=BUR001  POSN=*2*                 (2)
         $RUN *SPSS  8=*T*                                            (3)
            .
            .
            .
         INPUT MEDIUM   TAPE                                          (4)
            .
            .
            .

         READ INPUT DATA

            .

            .

            .

         $ENDFILE                                                     (5)
         $SIGNOFF $                                                   (6)
```

(1) A $HOLD message must be included whenever a peripheral device requiring operator intervention is required - see Section 4.6.
(2) The MOUNT command loads the magnetic tape and assigns it a pseudo-device name by which it may be attached to the program. Further details of the MOUNT command are to be found in Section 4.6.2 and NUMAC Document: "Introduction to Magnetic Tapes".
(3) The tape is attached to the program as data just like any other input or output device via the RUN command. The SPSS control commands are expected to follow the RUN command in the job.
(4) The program SPSS is told that the data is on magnetic tape.
(5) This marks the end of the SPSS control commands.
(6) This form of the SIGNOFF command produces an abbreviated form of the tail-sheet accounting.

4.2 BATCH PROCESSING BY HASP

All batch processing of jobs in the NUMAC system is controlled by a program known as HASP (Houston Automatic Spooling and Priority System.) This was originally written for use with IBM's standard operating system (OS) and was later revised and adapted for use with MTS. At NUMAC two operating systems are used, MTS and OS, the latter on two different computers. Since both operating systems already used HASP for spooling, they have been made to use identical versions of HASP so that they may use the same spooling discs. Spooling stands for "Simultaneous Peripheral Operations On-Line" which is effectively what HASP does. It controls all the incoming and outgoing work, assigning it priority for execution and printing, and acting as an interface between the many remote job entry stations and the two operating systems.

A batch job in HASP is processed in five phases:

Input phase
Execution phase
Print phase
Punch phase
Purge phase

At any one instant, there are normally several batch jobs in each phase competing for the use of the various hardware and software facilities.

During the input phase, HASP reads in the job from the card reader (or a terminal - see Section 4.5) and stores it in an area on a special disc to await execution. After the job has been read in its entirety, an entry is made on a HASP execution queue and the job is assigned a number between 1 and 4999 by which it may be identified all the time it is in the system. This is referred to as the HASP job number. The job is now ready for execution.

At the beginning of the execution phase, HASP creates an MTS task specifying the input disc area containing the job and specifying the output disc areas for printed and punched output. A print disc area is always needed by a batch job, but a punch area is set up only if the job indicates that this is necessary. When the job execution is terminated, HASP is informed, the MTS task is destroyed, and the disc area containing the job input is released.

At this point execution is over but the job output is still on the output discs. The job is now placed in the print queue to initiate the print phase. During this the job's printed output is produced on a lineprinter under HASP control. If it also produced punched output, it is entered into a punch queue after completing the print phase. In this case the job enters the punch phase and output is produced on a card punch.

When all printing and punching is complete, the job enters the purge phase. Here the job is placed in the purge queue and finally removed from the system. During this phase, all remaining disc areas set up by HASP for this job are released.

HASP controls the scheduling of jobs in each of the five phases. Jobs are selected on the basis of priority. Each is assigned an execution priority when it is placed in the execution queue, and a print priority when it is placed in the print queue. These priorities are numbers between 0 and 15; the higher numbers being the higher priorities. The execution priority assigned to each job is based on the CPU time limit specified on the SIGNON command. (A table of these priorities and time limits is displayed in the Ground Floor Batch Station at Newcastle or may be obtained by listing file INFO:OP.PRIO) After execution has terminated, the job is assigned a print priority based on the actual number of pages printed. There are no priorities associated with punched output, jobs are produced in the order in which they finish printing.

Once a job is placed in the execution queue it will be scheduled automatically by HASP unless it requests peripherals which require operator intervention, such as magnetic tapes, private disc packs or paper tapes - see Section 4.6. In these cases a HOLD message giving details of the peripherals required must be included in the job immediately before the SIGNON command as shown in Section 4.1 Example 3. HASP does not release these jobs until instructed to do so by the operators. The HOLD message was also described in Appendix B of "Introduction to MTS".

Full details of operational arrangements for running MTS jobs are to be found in the "NUMAC Service Documents". The most important details are repeated also at frequent intervals on the inside covers of the NUMAC Newsletter, namely the operating timetables, the resource limits in MTS and the telephone numbers for dial-up interactive terminals.

4.3 BATCH VERSUS TERMINAL MODE

As described in Section 1.1, computing under MTS can be carried out in two modes: batch or conversational mode. MTS is oriented towards the use of terminals so a number of very useful facilities are available in conversational mode.

Programs under development can be run interactively from a terminal. In conversational mode the user can intervene when an error becomes apparent, correct it immediately and try again. In this way programs may be "debugged" very much more rapidly than in batch mode. Similarly, as much editing as possible should be carried out in conversational mode so that the effect of each edit command is at once apparent and mistakes can be rectified before irreparable harm is done.

Once a program has been developed the situation changes. There are a small number of jobs which are designed to run interactively with the user directing the course of work, e.g. some simulation programs. On the other hand the majority of jobs can be run in batch mode entirely satisfactorily. Indeed, in such cases as jobs which create large volumes of output, it is desirable or necessary that they do so. It is wasteful of scarce computing facilities and extremely selfish to run large programs from a terminal when no intervention is required. The amount of elapsed time is, at best, about one minute per c.p.u. second required, and at busy periods this ratio increases by a factor of at least three.

4.4 USE OF TERMINALS

This section deals with the aspects of MTS directly pertinent to its use from a terminal. Although many different kinds of terminal devices are supported by MTS, these fall into two main categories: printing or hard copy terminals and visual display unit (vdu) or television-like terminals. Printing terminals are noisier and often operate at slower speeds than vdu terminals. The latter however only show a limited amount of the terminal session while a printing terminal records the entire session for later reference.

The operational and administrative details concerning the use of terminals are liable to change from time to time so these are dealt with in the "NUMAC Service Documents". These are supplemented by items in the NUMAC Newsletter of which every user should receive a copy. In particular the telephone numbers for the dial-up service are repeated on the inside covers of at least every alternate issue. Finally full details of MTS terminal support are given in the NUMAC Document: "NUMAC Terminal Guide" due for publication in 1979.

Terminals enable users to interact with MTS in many ways: they are useful to all users for file editing and initial program development; MTS offers special facilities for interactive programming from most terminals and special terminals are available for displaying graphical output. Once connected to MTS, communication with the system is through the command language as in batch mode. To help users at a terminal the first character of each line - known as the prefix character - identifies which component of the system is sending messages or awaiting input. The following table shows the most commonly used characters: these are generally correct but may vary with some models of terminal.

Mode	HEX Code	Prefix Character	Octal Code
MTS Command Mode	5B	# or $	043
Copying,Listing	6E	>	076
Loading	4B	.	056
Prompting	6F	?	077
Edit Mode	7A	:	072
Fast Insertion	6F	?	077
Debug Mode (1)	4E	+	053
Command Insertion	6F	?	077
Systemstatus Mode (2)	60	-	055
Archive System (3)	60	:	072

(1) See Section 5.7.
(2) See Section 4.8.2.
(3) See Section 3.10.

4.4.1 Input Line Editing

Each terminal device has its own particular characters or combinations of characters which have special functions in controlling the manipulation of information read from the terminal. The five most commonly required special functions are:

End-of-line Character (ELC)
This ends an input line and transmits the line to MTS. An ELC is usually generated by the RETURN key.

Delete-Previous Character (DPC)
This deletes the last character entered. The key(s) for this vary but the two most commonly found are BACKSPACE or 'CONTROL and H' pressed simultaneously.

Delete-line Character (DLC)
This deletes the whole line and is commonly achieved by the keys 'CONTROL and X'.

End-of-File Character (EFC)
This character sends an end-of-file signal to the system. The keys 'CONTROL and C' or 'ETX' often perform this function which is equivalent to a $ENDFILE MTS command.

Device Command Character (DCC)
In addition to the MTS commands, there are device commands which control the behaviour of a terminal - see Section (4.4.3). These are all prefixed by the DCC, which is '%' (hex code '6C') by default.

4.4.2 Attention Interrupts

During a terminal session an operation or program may be interrupted by pressing the interrupt or attention key. Certain MTS operations such as creating, emptying and destroying files may not be interrupted, otherwise the integrity of the file system could not be maintained: other operations such as LIST or COPY may be. For instance when all the required section of a file has been listed, an interrupt may be generated and this will cause the listing operation to be abandoned, the session will return to MTS command mode, and ATTN! will be printed followed by a prompt for further MTS commands.

When a program is interrupted the effect is rather different. The system will record all the relevant information about the state of the program and revert to command mode after printing the message:

ATTENTION INTERRUPT AT xxxxxxxx

xxxxxxxx specifies the memory address of the next machine instruction which would have been executed. The user may then issue MTS commands (with the exception of RUN, LOAD, UNLOAD, RERUN and DEBUG) and later restart the program with the RESTART command - see Section 5.6.

4.4.3 Device Support Routines

Each type of terminal is interfaced to the MTS system by a set of subroutines known as Device Support Routines. These perform various tasks including the control of data input and output, error recovery, recognising attention interrupts and helping with signon/signoff procedures. There are a number of device commands which allow users to interact with these routines and thus alter the input and output line editing, case conversion, and control of the terminal. A command must begin with the device command character ('%' by default). If the command is accepted as valid the response will be %OK; if not the message "LINE DELETED: INVALID DEVICE COMMAND" will be displayed on most terminals. Not all device commands are effective on all terminals. A complete list of such commands is given in the NUMAC Document "NUMAC Terminal Guide", but a few of the most useful are given here for users' convenience.

%DONT

This command (don't break the telephone connection) retains the terminal connection with MTS after the SIGNOFF command has been issued. The SIGNOFF statistics are followed immediately by the MTS session header, port number and terminal identifier.

%HEX=ON %HEX=ON; %HEX=OFF

This enables or disables hexadecimal input editing and, in form 2, redefines the delimiter from the default, which is an apostrophe, to a semi-colon.

%K=LC %K=UC

The programs supporting the connection of some terminals to the system, especially those with no facilities for displaying lower case letters, force the translation of all characters into upper case. The first form of this command overrides such a translation and the second reverses this again. (This command works independently of the MTS @UC modifier.)

%LEN=<n> %LEN=OFF

This sets the truncation length for output lines where 0<n≤256. %LEN=OFF returns to the default value which depends on the terminal concerned. When n is greater than the width of the terminal's display line, the output line is continued on successive lines.

%RMAR=<n>

This resets the right margin position: 0<n≤256. The default value depends on the terminal concerned. %RMAR controls the width of the display on the terminal : when LEN>RMAR, RMAR is the point at which a line of output is continued onto the next line of the terminal's display.

%TABI=ON %TABI=ON;x,t,t,t--- %TABI=OFF

This enables or disables the recognition of x as an input tab character for tab stops t. There may be up to ten values of t where 0<t≤256. Once tabs have been set up using this command, they can be used by means of the tab character, the default for which is the TAB key or the keys CONTROL and I. Note the physical tab stops (if any) on a terminal bear no relation to the TABI command.

4.4.4 Trouble!

If you get into a mess during a terminal session and the system seems to be taking over, here are four suggestions for re-establishing control. Try them in the order they are given.

(1) type "STOP"
(2) type "$ENDFILE" or use the end-of-file character
(3) press the attn or interrupt key
(4) switch off the terminal.

Suggestion (4) should be regarded only as a last resort: it may take five minutes or more for MTS to terminate a session automatically if this is used. If you wish to leave the terminal in such circumstances please contact the operators quoting the terminal identifier and port number - see "NUMAC Service Documents".

4.4.5 Routing Output From A Terminal

From time to time you may wish to generate output as a lineprinter listing or on cards from a terminal. This is done via the LIST or COPY commands using the pseudo-devices *PRINT* or *PUNCH* as shown in Sections 3.5 and 3.6. However it is necessary to consider the routing of such output. The default routing is to the central site lineprinters and punch at Newcastle but this is not very convenient for users working elsewhere. Each remote job entry (RJE) station is denoted by a one to six character mnemonic, e.g.

```
Newcastle central site = CNTR
Durham                 = DURH
```

Remote users should enquire about their local RJE station mnemonic and then route output accordingly using the CONTROL or SET commands, e.g.

```
$SET ROUTE=DURH
```

This can be done semi-automatically via a sigfile - see Section 4.5.2.

Occasionally jobs produce output which you know in advance will be of no interest. The system can be asked to discard such output before it is printed or punched by routing it to "DUMMY", i.e.

```
$SET ROUTE=DUMMY
```

There are three routing parameters:

CROUTE controls the routing of punched card ouput
PROUTE controls the routing of lineprinter output
ROUTE controls the routing of both types of output

4.5 SUBMITTING BATCH JOBS FROM A TERMINAL

Jobs can be entered into the batch stream from a terminal by means of the pseudo-device *BATCH*. This simulates a card reader feeding jobs into the HASP execution batch stream.

As there are only a limited number of ports into *BATCH*, the pseudo-device should be kept open for the minimum length of time so that as many people as possible can have access to it. For this reason we recommend that the program *BATCH is used to enter jobs to *BATCH*. The program which can be used to enter either MTS or OS jobs has a number of other advantages especially for users of MTS:

Parameters on the SIGNON command are checked.

If the next line begins with a $ the user is prompted in case the password has been forgotten.

The line $ENDFILE can be entered as data.

Records already entered can be changed or deleted.

The program is called as follows:

```
$RUN *BATCH [SCARDS=filename] [PAR=jobname]
```

where "filename" is the name of a file containing the batch job(s) to be entered.

*BATCH supplies a jobname record automatically for each job. By default the jobname used is the current identifier prefaced by a 'Q'. This may be overridden by the PAR field for the duration of the RUN command only.

If SCARDS is not specified, the program will prompt the terminal for input. This should begin with a $SIGNON command and end with $SIGNOFF, followed by '/*' if further jobs are to be entered, or $ENDFILE (end-of-file). $ENDFILE is only interpreted literally by the program when it is preceded by a $SIGNOFF command, in all other contexts it is accepted as a data line.

(MEMO: Do not forget to type the MTS command prefix character for all MTS commands entered!)

If a line is wrongly entered, immediately following it with a null line, i.e. pressing RETURN a second time, will delete it. Entering n null lines deletes the previous n lines.

Finally if a complete job is kept in a file, the password need not be included. *BATCH will prompt for this when it checks the SCARDS file.

A job is typed in directly:

```
$run *batch
EXECUTION BEGINS
_$hold needs plotter
_$signon xcl1 t=10
ENTER THE JOB'S PASSWORD OR $ENDFILE
(display of the password will be prevented if possible)
?password
_(rest of MTS commands are entered)
_$signoff
_$endfile
_*BATCH* ASSIGNED HASP JOB NUMBER NNNN (xxxxxxxx)
_*BATCH* RELEASED
EXECUTION TERMINATED
```

('NNNN' is the HASP job number and 'xxxxxxxx' is the jobname assigned by *BATCH*. Either of these may be used with a LOCATE command see Section 4.8.1)

A job has been stored in a file, without the password. This is the recommended way to use *BATCH for all but very small jobs.

```
$run *batch scards=myjob
EXECUTION BEGINS
_ENTER THE JOB'S PASSWORD OR $ENDFILE
?password
_*BATCH* ASSIGNED HASP JOB NUMBER NNNN (xxxxxxxx)
_*BATCH* RELEASED
EXECUTION TERMINATED
```

OS jobs may be entered but the program does not check them:

```
$run *batch
EXECUTION BEGINS
_//qclb6 job ------
_(rest of OS job)
_$endfile
_*BATCH* ASSIGNED HASP JOB NUMBER NNNN (xxxxxxxx)
_*BATCH* RELEASED
EXECUTION TERMINATED
```

4.5.1 The SOURCE Command

Instead of storing a complete job in a file it is often more convenient to store just the commands to carry out a specific task, i.e. without the SIGNON and SIGNOFF commands. Then these can be incorporated into a job by using the SOURCE command.

$SOURCE filename

causes MTS commands to be read from "filename" instead of prompts being sent to the terminal. At the end of the file the terminal user will again be prompted for commands. If file DATANAL contains the commands to analyse a particular set of data, the following job might be submitted:

```
$run *batch
EXECUTION BEGINS
_$signon xcl1 t=10
_albert
_$create results1
_$source datanal
_$list results1
_$signoff
_$endfile
_*BATCH* ASSIGNED HASP JOB NUMBER NNNN (XXXXXXXX)
_*BATCH* RELEASED
EXECUTION TERMINATED
```

4.5.2 Sigfiles

Although the standard settings for switches and the default values for parameters are chosen to satisfy the majority of users, they cannot, of course, meet the needs of all users. Therefore it is possible to alter these switches and parameters. However, it becomes very tedious to have to do this everytime one signs on, so the concept of a sigfile or signon file was developed. A sigfile is an ordinary line file which the user specifies 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.

One of the commands which might be executed is

$FILESTATUS

to give a brief listing of all filenames as a reminder. Another might be to route output to a remote job entry station which is more convenient than the central site reception area in Newcastle:

$SET ROUTE=DURH

The following commands illustrate how a sigfile is set up:

```
$create sigf
$edit sigf
: i
?$filestatus
?$set route=durh
?$run *anymail
?
:stop
$set sigfile=sigf
```

Note: See Section 4.8 for details of *ANYMAIL.

As soon as the user signs off the name of the sigfile is saved in the accounting record and the sigfile becomes effective.

The contents of a sigfile may also include a program which carries out security checks against unauthorised access where a user feels that this is necessary. The section "Sigfiles and Security" in MTS Volume One deals with this topic. However beware of being too clever in this area, lest you lock yourself out of your own identifier !

4.6 BATCH JOBS WHICH REQUIRE OPERATOR INTERVENTION

Input and output peripheral devices which use media such as cards, tape or paper must be loaded and controlled by the operators. Such devices are:

- Card readers
- Card punches
- Line printers
- Magnetic tape drives
- Paper tape readers
- Paper tape punches
- Central Site Graph plotters

Jobs requiring such devices are dealt with in one of two ways. Access to the first three types is dealt with automatically under HASP control by means of standard system pseudo-device names:

```
*MSOURCE* or *SOURCE*
*PUNCH*
*PRINT* or *MSINK*
```

- see Section 3.3. The remaining devices require operator intervention at the time the job is running. Jobs requiring these devices must alert the operators by means of a special control message which is included as part of a HOLD command which causes the job to be held in special queues. The operators read the message and release the job manually when the required devices and media are ready. Since the HOLD command is acted upon by HASP, it must occur in the deck immediately before the first true MTS command, SIGNON. It takes the form

$HOLD message

where "message" describes briefly to the operators the devices required and the data items to be loaded. Examples of HOLD messages were given in Sections 4.1, Example 3 and 4.5, Example 1.

The MTS MOUNT command is then used to assign a pseudo-device name to a specific paper or magnetic tape, by which it may be referenced thereafter. For graph plotting there are a number of administrative programs to produce plots on the different devices.

4.6.1 Paper Tape

NUMAC's central site in Newcastle has one paper tape reader for 5, 7 and 8 channel tape and one paper tape punch for 8 channel tape. In addition, paper tapes may be read in via some terminals. At present the use of paper tape in MTS is described in a number of documents: the main one is

"The Paper Tape User's Guide"

while operational procedures are given in the "NUMAC Service Documents". These are supplemented by some leaflets describing special facilities:

"*NFHFILE(13)"
A program designed to read a sequence of several paper tapes, checking that no tape is read in twice.

"*NFHFILE(14)"
A program to process ASCII coded paper tapes.

"Reading Paper Tape in NUNET"
How to read paper tape in via a NUNET terminal.

Al these publications are currently available from the program librarian at Newcastle. They will be drawn together in the second edition of "The Paper Tape User's Guide" which we plan to publish during the 1978/79 session.

4.6.2 Magnetic Tape

There are four magnetic tape drives attached to the 370/168 computer and thus available to MTS programs. The most important fact to remember about magnetic tapes at NUMAC is that the drives can use only 9 track tapes recorded either at 1600 b.p.i., phase encoded or 800 b.p.i., NRZI. Full details of the physical and logical tape formats which can be handled and the operational procedures for dealing with them are given in the "NUMAC Service Documents". There are two further documents which describe the MTS commands and programs for handling magnetic tapes:

"Introduction to Magnetic Tapes" published June 1978

"Magnetic Tape User's Guide"

The second edition of the latter is due for publication in 1979.

4.6.3 Graphical Output

Graph plotting facilities are available at all three NUMAC Northumbrian sites. The NUMAC Document

"Graphical Facilities at NUMAC"

gives full details of the graphical display terminals and plotters available for public use; descriptions of program packages and subroutine libraries; operational procedures for using the plotters and four utility programs :

*PLOTSEE	to view plot descriptor files
*UNEPLOT	to obtain hard copy
*DURPLOT	plots at the three
POLY:PLOTFILE	Northumbrian sites.

4.7 LINEPRINTER LISTINGS

4.7.1 Carriage Control

Carriage control characters are a means of determining the vertical spacing of output. They are used mainly for output to a terminal or a printer but may also be used to specify control operations for magnetic or paper tapes - see NUMAC document: "Magnetic Tape Users Guide". If carriage control is being used, the first character of every record is interpreted as a carriage control character. It is not regarded as part of the text; instead it is stripped from the output record, used to position the printer and printing begins with the second character. If the first character is not one of those defined as control characters for the particular device being used, a single space is assumed, and the first character is printed as part of the output text. The carriage control character codes are independent of a program's source language.

Example:

```
$copy *source* *print*
12345
abcd
$endfile
```

will produce a listing with "2345" at the top of a new page and "ABCD" on the following line.

MTS supports two types of carriage control: logical and machine. Both are used in the same manner but differ in the legal characters and their effect. Logical carriage control is the more common and, in general, the user need not be concerned with machine carriage control. The latter is supported because some programs such as *ASMG and *TEXT360 produce it. In most cases in which carriage control is desired, logical carriage control is enabled by default. This section will describe only logical carriage control: details of machine carriage control are to be found in Appendix H of MTS Volume One: "The Michigan Terminal System". To select either machine carriage control or no carriage control, the appropriate modifier must be specified - see Section 3.8.

Example:

If the COPY command in the previous example had been :

$copy *source* *print*@¬cc

the listing produced would have had two lines, "12345" followed by "ABCD".

When text is formatted for printing with logical carriage control, it is assembled into logical pages of 60 lines. The logical page is divided into two halves, four quarters and six sixths. A logical carriage control character of 4 will position the printing at the start of the next quarter page even if this may be the top or middle of a page. The logical page is then mapped onto the physical page of the output device concerned. Where this is a lineprinter, normally the first and last two lines of a page are skipped automatically. This is termed the overflow. It means that the logical "top" of a page is physically two lines below the perforation but overflow can be suppressed if desired.

CHARACTERS	EFFECT BEFORE PRINTING	EXCEPTIONS PRINTER	TERMINAL
blank	single space (SS)		
0	double space		
-	triple space		
+	overprint previous line-print without spacing first		SS
&	suppress carriage return after printing	undefined (i.e. SS)	
9	single space and suppress overflow		
1	skip to top of the next page		skip 6 lines
2	skip to next 1/2 page		skip 6 lines
4	skip to next 1/4 page		skip 6 lines
6	skip to next 1/6 page		skip 6 lines
8	same as 6		skip 6 lines
;	skip to top of next physical page (i.e. line 1)		undefined
<	skip to bottom of next physical page (i.e. line 63)		undefined

4.7.2 Special Prints

The lineprinters at the central site are normally fitted with a PN character set which contains no lower case letters. Further, since output is printed at eight lines per inch (lpi), the character height is slightly smaller than usual. For special purposes one printer can be fitted with either a full size PN set or a TN set. The latter contains a greater range of characters including both upper and lower case letters. The standard listing is produced on 8ins. x 14.5ins. paper of 50gsm weight. Several other types of 11ins. x 14.5ins. paper and special stationery are available for use with both PN and TN (upper and lower case letters) chains, printing at 6 lines per inch. The table below sets out those available at the time of writing this document. The different types of stationery are referred to as forms and each is identified within the operating system by a number in the range 1-99.

FORMS	CHARACTER SET		
	Std PN	Small PN	Std TN
8" x 14.5" 50gsm at 8lpi (default)	-	1	-
11" x 14.5" 60gsm at 6lpi	10	-	5
11" x 14.5" 85gsm at 6lpi	6	-	17
Two part (one carbon)	2	-	4
Three part (two carbons)	3	-	7
Peelable labels, 3 across	11	-	8
Bandalith masters	-	-	16

This table will be repeated from time to time in the NUMAC Newsletter where the current version should always be checked. To use any special form number other than 1 or 10 you must have authorisation from the Operations Supervisor at the Central Site, before you submit each job. For details of special prints at other locations, please consult your local advisory service.

The following example demonstrates how the forms numbers are used. This batch job will produce an upper and lower case copy of the contents of file MYPAPER on 11ins. x 14.5ins. 60gsm paper printed at 6 lines per inch, which is forms number 5.

```
$SIGNON XXX1 PRINT=5
<password>
$COPY MYPAPER
$SIGNOFF $
```

or alternatively from a terminal:

```
$set print=5
$COPY MYPAPER *PRINT*
*print* assigned hasp job number nnnn (qxxx1 )
*PRINT* RELEASED, NN PAGES
```

The LOCATE command (see Section 4.8) should then show the job in the correct queue:

```
$locate qxxx1
?JOB NNNN QXXX1   (PRIO 4) POSN N PRINT Q (PTR1   ),FORMS=5
```

4.8 SOME USEFUL PUBLIC PROGRAMS

There are many programs available as public files under MTS. These are listed in Appendix B and in MTS Volume 2 : "Public File Descriptions". One or two of the most useful are described here.

4.8.1 Tracing Jobs

One of the most frequently used commands is LOCATE which enables users to trace the progress of individual jobs through the system. The command can take various parameters of which the two most useful are "jobname" or "HASP job number". For example

```
$LOCATE QCLD1
```

will search for all jobs whose jobname begins with QCLD1 and might produce the following reply

```
?JOB 3058 QCLD1 (PRIO 4)POSN 7 PRINT Q(PTR1)
?JOB 2846 QCLD1 (PRIO 5)POSN 31 EXECL Q
```

Alternatively if no jobs can be found with jobname QCLD1 the reply will be:

```
***JOB NOT FOUND OR DONE
```

A job may also be traced by its HASP job number:

```
$locate 2846
?JOB 2846 QCLD1 (PRIO 5)POSN 31 EXECL Q
```

4.8.2 Assessing System Workload

The command SYSTEMSTATUS allows the workload of the system to be assessed, which is particularly helpful when embarking on a terminal session. There are two parameters which are most useful. The first will give an estimate of the total workload and thus an indication of the response time to be expected:

```
$systemstatus users
- THERE ARE 72 TERMINAL USERS, 8 BATCH JOBS, 86 AVAILABLE
LINES,
-AND 33 NON-MTS JOBS USING 3811 VIRTUAL PAGES AND 1394 REAL
PAGES..
$
```

The second can be used to see if there is a magnetic tape deck available before issuing a MOUNT command. This is done by asking for tasks using 9 track tape (TYPE 9TP):

```
$systemstatus tasks type 9tp
-00078     MTS 02C4C0  16 READY GGN6 DGGO; TOCO TOC1 HSL1TT06
-00826     MTS 02F440  28 I/O ON TOC2 N101 JNBO QJNIOP; TOC2
```

The response lists all the MTS jobs using such devices. The last four items for each job are its MTS identifier, its MTS project identifier (terminated by a semi-colon), the tape drive(s) being used and the terminal from which it is being run. There are currently four tape devices available (TOC0-TOC3)

A word of explanation! If SYSTEMSTATUS indicates that some drives are free but the operator replies to your MOUNT command "NO FREE DRIVES" this is not a contradiction. The remaining tape drive(s) may be booked or have faults requiring the engineer's attention. If you know that you will need a tape drive during a terminal session you may book one in advance by telephoning the main machine room (Newcastle 29233 Ext 252).

4.8.3 *MAIL - A NUMAC Message Facility

This program allows messages to be sent and received by MTS users. Note that it provides a mail facility only - messages are not displayed automatically if the recipient identifier is signed on when the message is sent. *MAIL must be run both to send and print messages. Messages remain in the MAIL system after printing, and should be deleted using *MAIL when finished with. The system will remove very old messages, but good housekeeping will be appreciated.

The program is accessed by the RUN command:

$RUN *MAIL

and is self-documenting via its HELP or EXPLAIN commands. The commands available are:

* SEND	send a message
* PRINT	print messages to which you have access
* DELETE	delete messages to which you have access
* SUMMARY	summarise any pending messages
* LIST	list messages to which you have access
INSERT	add lines to the program message buffer
DISPLAY	display the program message buffer contents
EDIT	edit the program message buffer contents (um edit)
CLEAR	clear the program message buffer
* HELP	request program information
* EXPLAIN	is a synonym of HELP
* QUERY	find preferred mail identifier for a staff member
STOP	(or end-of-file) terminate execution

All commands may be abbreviated to their minimum unambiguous form. Commands whose names are marked with an asterisk can be given in the *MAIL PAR field, in which case the program will execute the supplied command and return to MTS command mode. For example:

```
$RUN *MAIL PAR=SEND CL47 "PLEASE PHONE ME ...  JOHN"
```

Lines starting with a dollar sign are passed to MTS for interpretation.

5 PROGRAM DEVELOPMENT

Before deciding to write your own programs, we earnestly advise you to enquire what programs and subroutines already exist in your chosen subject area. Appendices B and C of this document list the majority of those available under MTS. Further programs are implemented under IBM's standard OS operating system. NUMAC staff will always try to advise users on the choice of programs available : please consult the Duty Officer in the first instance.

5.1 LANGUAGE CHOICE

The first choice facing the user who decides to develop his own program is that of selecting a suitable language. There are many languages available in MTS, and the choice is seldom straightforward.

It will depend on the factors listed below.

The types of object to be manipulated by program.
Will you be handling character strings, numbers, or complicated structures such as trees.

The input and output facilities.
Are they suited to your needs.

Your need for portability.
Will you require to run your programs on other computers? If so, what languages (and which dialects of those languages) are available on the other machines.

The debugging facilities available.
As most programming effort is spent debugging, it is important to use compilers which facilitate this.

The efficiency of the code produced.
In general, languages which save programmer time tend to cost more in machine time. If your program will be run only a few times this will not matter. If you will be using it regularly on a 'production' basis, it may be an important factor.

The availability of expertise.
We have many languages available about which we can give only slight advice. If you are not very experienced, it is unwise to explore such areas alone.

Existence of subroutine libraries.
Large numbers of working subroutines are available in libraries such as the Numerical Algorithm Group (NAG) library. Use of these can reduce programming effort enormously, provided they can be called from the language you are using.

5.2 PROGRAM DESIGN AND WRITING

Having decided on a language it is important that the program be properly designed, before the actual coding starts. You must decide how it will be split into subroutines, and how these can be separately tested. As much of the development effort is consumed in debugging, programs should always be written with debugging in mind. It is well worth while including extra statements to trace program execution, and to verify implicit assumptions e.g. ($X > 0$ or $1 \leq I \leq 365$) Further, specific data should be carefully designed for use in testing a program in order exercise all its aspects.

5.3 COMPILING

The programmer's coding is often called the 'source' program. Before it may be run on the computer, it must be converted into a form called an 'object-module', by a program called a 'compiler'. By convention, most compilers read their source programs from unit SCARDS and write the corresponding object module on SPUNCH. We might therefore compile an AlgolW program in file AWPROG with the commands

```
$CREATE OBJPROG
$RUN *AW SCARDS=AWPROG SPUNCH=OBJPROG
```

The object module might now be run by

```
$RUN OBJPROG
```

Most compilers will also accept in the parameter list a sequence of words which control various compiler options such as the program listing, and cross references. Sometimes there are special debugging options which produce code to check for certain programming errors such as using unset variables or invalid array subscripts.

For a few languages, there are special debugging compilers which provide comprehensive error checking at some cost in efficiency. These are used until a program is thoroughly tested, when it can be recompiled with the normal compiler to increase running speed. Thus for FORTRAN, FTN (with SDS) and IF are debugging compilers, and FTNX is a production compiler.

5.4 THE RUN COMMAND

The RUN command causes MTS to acquire sufficient virtual memory in which to load the specified object module. If the loading is completed successfully, MTS prints the message

'EXECUTION BEGINS'

and transfers control to the program. If problems arise during loading, one of the following messages will be printed.

ATTEMPT TO LOAD A NULL PROGRAM

This means that the source for loading is empty, and does not contain an object module. It may be that a file title was mis-spelt, or that compilation errors caused the compiler not to generate an object module.

THERE ARE nn UNDEFINED SYMBOLS

This means that at the end of loading, a number of subroutines had been used but not defined. It may be that one has forgotten to write them, one has mis-spelt their names when calling them, or that object modules containing their definitions are contained in a file or library one has forgotten to load. In this latter case the second source file should be included in the RUN command e.g.

```
$RUN    PROGOBJ+MOREOBJ
```

In batch, this message is followed by a map of what has already been successfully loaded, but no attempt is made to execute the program. At a terminal, the user is invited to

```
.ENTER LOCN OF MORE LOADER INPUT,"CANCEL","IGNORE",
."USMSG"," UXREF" OR "MAP":
```

CANCEL	Causes the RUN command to be abandoned
USMSG	Prints the names of the undefined symbols
UXREF	Indicates which programs already loaded refer to the undefined symbols
IGNORE	Ignore the references to undefined symbols, and proceed to run the incomplete program
MAP	Prints a map showing what has already been loaded

ABNORMAL LOADER INPUT ORDERING
LOADING ERRORS ARE NON-RECOVERABLE

The file being loaded does not contain a standard-format object module. A common cause is accidentally attempting to load a source file. Forgetting to empty an object file before recompiling can also cause this, as the new object module may not completely overwrite the old one, leaving a mixture in the file.

5.5 RUN TIME ERRORS

After a program is successfully loaded, the computer will start executing it, and continue until either the program stops, or it attempts to do something invalid, such as dividing by zero, computing a number too large to be represented in the machine, or trying to access a storage location not belonging to the program. This causes an 'interrupt', and the program will be halted. It is possible for a program to make allowances for such unruly behaviour by 'trapping' the fault, and then printing its own diagnostics. This is done in all the main programming languages such as FORTRAN and AlgolW, and their documentation should be consulted for details of the error handling.

If interrupts have not been trapped by the program, MTS will print a map of the loaded program, preceded by the PSW or program status word. This is a pair of hexadecimal 8 digit numbers. The last digit of the first number specifies the type of interrupt and the rightmost six digits of the second word give the address where the interrupt occurred. As MTS also prints an explanation of the type of interrupt, generally it should not be necessary to resort to the IBM manuals. However a full list of interrupt types may be found in IBM Manual GA22-7000 :

"IBM System/370 Principles of Operation"

The address from the PSW may be used to decide in which subroutine the interrupt occurred. Look at the printed map, and find the subroutine at the highest address which is less than the PSW address. This is the subroutine causing the problem. By subtracting (in hexadecimal) the subroutine address from the PSW address, one can find the area of the subroutine at which the interrupt occurred. A detailed program listing can then be consulted, to find the construction at that offset.

5.6 ATTENTION INTERRUPTS

When being run at a terminal, a program may be halted by pressing the 'attention' or 'break' key. MTS replies by typing

ATTENTION INTERRUPT AT nnnnnnnn

and returning to MTS command mode. "nnnnnnnn" is the address of the program instruction being obeyed at the time of interruption. The user can now issue any MTS commands. It might be useful to list the output files used by the program, to see that everything is proceeding as planned.

If it is desired to resume execution of the program from the current point, a RESTART command may be given. I/O unit assignments may be given on the RESTART command, just as on the RUN command. It is therefore possible to start a program, interrupt it, and restart it with different unit assignments. If they are omitted those given on the last RUN command will be used, e.g.

```
$run  progobj  5=testdata
ATTN!
$restart    ( restarts the program with 5=testdata)
ATTN!
$restart  5=realdata ( I/O unit 5 is reassigned)
```

5.7 DEBUG

The DEBUG command has a syntax identical to the RUN command, but instead of causing the program to be executed, it merely loads it together with SDS - the Symbolic Debugging System. This provides a wide range of interactive facilities to allow carefully controlled execution and monitoring of a program being debugged. Selected program or data locations may be printed or modified, and the program may be run step by step, or until a certain label is reached. SDS is described in NUMAC Document: "Introduction to SDS", and full details about its use are given in MTS Volume 1. MTS Volume 6 : "MTS FORTRAN" describes its use for debugging FORTRAN programs.

5.8 MANAGEMENT OF OBJECT MODULES

5.8.1 The Object File Editor

The MTS object file editor provides facilities to replace, add, delete or correct an object module. In addition it can generate and edit files of object modules. The object file editor is available in file *OBJUTIL and is called via the RUN command:

```
$RUN *OBJUTIL [SCARDS=file of object modules or commands]
   [SPRINT=editor outputfile] [0=file to be edited]
```

Complete details of *OBJUTIL are to be found in MTS Volume 5 : "System Services".

5.8.2 Updating Files Of Object Modules

A typical example of the use of *OBJUTIL is a FORTRAN program in the file "PROG" consisting of several subroutines whose object modules are in file "OBJ". While debugging the program, an error is found in one of the subroutines. This is corrected in the source file by using the MTS editor ($EDIT), the subroutine is recompiled and the object file editor is used to replace the erroneous module in the object file. The last two stages would be as follows:

```
$RUN *FTN SCARDS=PROG(200,299) SPUNCH=-LOAD SPRINT=*PRINT*
$RUN *OBJUTIL SCARDS=-LOAD 0=OBJ
```

5.8.3 Creating Object Module Libraries

A common pattern of programming is to develop a large number of subroutines to perform frequently needed tasks in one's subject area, and then to write several main programs which make calls on the subroutines. If we keep the object modules of the subroutines all together in a file MYROUTS.O, we could run a program that needed some of them by

```
$RUN   PROG7.OBJ+MYROUTS.OBJ
```

However this has the disadvantage that all the subroutines will be loaded, even if PROG7 needs only a few.

This may be avoided by storing them in a subroutine library file which has a special format for storing a collection of object modules. If encountered during the loading process, not all the object modules are loaded, but only those which have been referred to but not yet defined. Thus just those routines needed are loaded, with consequent savings in memory and loading time.

The command language of the object file editor, OBJUTIL, provides facilities for creating and editing library files as shown in the following two examples:

```
$run *objutil
*comment create and generate a library file
*create@library sublib size=20p
  FILE "SUBLIB" HAS BEEN CREATED.
*sniff
  *** THERE ARE NO OBJECT MODULES IN "SUBLIB"
*add myrouts.obj allbut main
  ADDED:
  READIN   OUTPUT   PASS1   PASS2
*include sort.0+*lcs
   INCLUDED:
   SORT   SORT1   SORTEA   ENDJUNK
*list
   READIN      3.000    OUTPUT     10.000
   PASS1      14.000    PASS2      18.000
   SORT       22.000    ENDJUNK    43.000
*list endjunk
  ENDJUNK   LCS   LCSYMBOL
*sniff
  LINE  LIBRARY  FILE  "-SUBLIB" HAS 5 MODULES, 7 ENTRY POINTS
AND 43 LINES.
*stop
```

This second example shows how such subroutine libraries are maintained:

```
$run *objutil 0=sublib
*comment to replace a module in a library file
*replace from -load
  REPLACED:
  PASS1  PASS2
*comment to add further modules to the library
*add from test.0
  ADDED:
  TEST   NUMBER
*list
   READIN      3.000    OUTPUT     10.000
   PASS1      14.000    PASS2      18.000
   SORT       22.000    ENDJUNK    43.000
   TEST       62.000    NUMBER     75.000
```

APPENDIX A: MTS COMMAND DESCRIPTIONS

A1: INTRODUCTION

This appendix contains specifications of the MTS commands. Only the most generally used commands and their most useful parameters are included. However most users will find this sufficient for their needs.

In the specifications the following conventions are used:

Lower case	represents a generic type which must be replaced by a specific item.
Upper case	represents material to be reproduced verbatim.
Brackets []	represent optional material.
Braces {}	enclose alternative items from which exactly one must be chosen.

CANCEL

PURPOSE:

This command is used to cancel *PRINT*, *PUNCH* or *BATCH* jobs submitted from a terminal or regular batch jobs that the user has previously submitted.

HOW TO USE:

```
$CANCEL job identification [user identification]
```

"job identification"
comprises the pseudo device name (*PRINT*, *PUNCH*, or *BATCH*) and the HASP job number, e.g.

```
$CANCEL *PRINT* 4022
```

If the device has been held, the job has not been released to the system, so it is sufficient to give just the pseudo-device name. In all other cases the HASP job number must be specified and the device name is optional. If both are given and the job number belongs to a job in a different pseudo-device queue an error comment is printed.

[user identification]
This information is required only if cancelling a job submitted under a different user identifier. It takes the form

```
ID=xxxx PW=yyyyyy
```

If the password is omitted, the user is prompted for it. A job cannot be cancelled once the system has begun to process it. Thus a *PRINT* job can only be cancelled if it is still awaiting printing.

EXAMPLES:

```
$CANCEL *PRINT*
```

The current *PRINT* job is cancelled.

```
$CANCEL 4021
```

The job with HASP number 4021 is cancelled.

```
$CANCEL *BATCH* 4021
```

The *BATCH* job with HASP number 4021 is cancelled.

ABBREVIATIONS: CAN

COMMENT

PURPOSE:

This command allows the insertion of comments into listings of commands. Otherwise it does nothing at all.

HOW TO USE:

```
$COMMENT this is a comment
```

Users should expect to have no trouble using this command.

ABBREVIATIONS: COM

CONTINUE WITH

PURPOSE:

This allows files to be implicitly concatenated.

HOW TO USE:

```
$CONTINUE WITH name [RETURN]
```

"name"
is the name of the file or device from which further information is to be read.

[RETURN]
This parameter is optional. If it is supplied reading continues with the record after the CONTINUE WITH command; otherwise control is returned to MTS command level when an end-of-file condition is encountered.

If implicit concatenation is disabled by means of a global switch or an @¬IC FDname modifier the $CONTINUE WITH line is read as a data line.

EXAMPLE:

If line 10 of FILEA is :
$CONTINUE WITH FILEB
and if there are no other '$CONTINUE WITH' lines in FILEA, then the command :
$COPY FILEA TO *PRINT*
will give a listing comprising lines 1-9.999 of FILEA followed by all of FILEB.

If line 10 of FILEA is:
$CONTINUE WITH FILEB RETURN
the listing will comprise: lines 1-9.999 of FILEA, followed by all of FILEB, followed by lines 10.001 to 'last' of FILEA.

The command:
$COPY FILEA(FIRST)@¬IC *PRINT*
will give a listing of the whole of FILEA only, including the CONTINUE WITH line.

ABBREVIATIONS:

None. There must always be just one space between the words 'CONTINUE' and 'WITH'.

CONTROL

PURPOSE:

This command allows the execution of control operations on certain types of files and devices.

HOW TO USE:

$CONTROL name control-command

"name"
is the name of the file or device on which the control operation is to be performed.

"control-command"
is the operation to be performed. There are many options for this parameter, all of which are described in MTS Volume One. Some of the most useful are described here and in Section 4.7.2 by means of examples.

EXAMPLES:

1. To double the size of file ELEPHANT whose present SIZE is 15 pages.

```
$CONTROL ELEPHANT SIZE=30P
```

This command achieves the same result as that above but by specifying the amount of the increment rather than the final value

```
$CONTROL ELEPHANT SIZEINC=+15P
```

2. To write files to magnetic tape for reading elsewhere - see also NUMAC Document: "Introduction to Magnetic Tapes".

```
$MOUNT CXXXPR *TAPE* W              Mount the tape
$CONTROL *TAPE* FMT=FB(7920,132)    It is a print file
$COPY OUTPUT1 *TAPE*                Write first file
$CONTROL *TAPE* WTM                 Write a tape mark at end
$COPY OUTPUT2 *TAPE*                Write second file
$CONTROL *TAPE* WTM 2               Write 2 tape marks
                                    to signify end of tape
$CONTROL *TAPE* REW                 Rewind tape
$COPY *TAPE*(1,50) -A               List first 50 lines
$CONTROL *TAPE* POSN=*2*            of each file to check.
$COPY *TAPE*(1,50) -A(LAST+1)
$RELEASE *TAPE*                     Release tape
$LIST -A *PRINT*                    Print out check listing
```

ABBREVIATIONS: CON

COPY

PURPOSE:

This command is used to copy records from one file or device to another file or device.

HOW TO USE:

```
$COPY [FROM] fromfile [TO] tofile
```

"fromfile"
is the name of the file or device from which the records are to be copied.

"tofile"
is the name of the file or device to which the records are to be copied. If this parameter is missing the current sink (*SINK*) is assumed.

[FROM], [TO]
are optional "noise" words which may be used to improve readability and occasionally remove ambiguity. For example:

```
$COPY FROM HORSE TO COW
```

is equivalent to any one of the following three commands:

```
$COPY HORSE COW
$COPY HORSE TO COW
$COPY FROM HORSE COW
```

The records are read sequentially from "fromfile" and copied into "tofile" until an end-of-file (EOF) is sensed from "fromfile". The lines are renumbered as they are placed in the "tofile" unless the "@I" modifier is specified for it. In addition the lines are copied from the "fromfile" starting at line number 1. For example:

```
$COPY HORSE COW@I
```

will copy the lines starting at line 1 of the file HORSE to the file COW. Line numbering will be conserved so the lines in the file COW will be numbered exactly the same way they were in the file HORSE.

To ensure that all lines of file HORSE are copied the command should read:

```
$COPY HORSE(FIRST) COW@I
```

When copying is being done to or from a terminal the prefix character issued by the COPY command is the ">", hex "6E".

EXAMPLES:

This command as used in its simplest form should give little trouble. For example:

```
$COPY ABBEY CHURCH
```

will copy the contents of the file ABBEY to the file CHURCH. The following two commands:

```
$COPY ABBEY *SINK*
$COPY ABBEY
```

are equivalent. In the second case "tofile" has been omitted and will be assumed by MTS to be *SINK*. The following command will copy from the current source (normally the terminal in conversational mode or the card reader in batch mode) into the file INFILE:

```
$COPY *SOURCE* INFILE
```

This is the recommended way to place data in a file (see the example under the description of the ENDFILE command).

There exists one ambiguous case:

```
$COPY FROM TO SNARF
```

In the above case TO is taken as a noise word and FROM is the name of the file from which the records are to be copied.

ADVANCED CONSIDERATIONS

COPY will work for any type of file or device. Restrictions are applied where necessary according to the type of file being used. For example, for "fromfile", line numbers are simulated if it is a sequential file or a device (for example *MSOURCE*). Copying a line file to a sequential file will result in loss of the line numbers. COPY may be used to copy a file from magnetic disc to tape or from one tape to another as shown in the following example:

```
$COPY *IN* *OUT*
```

In the above example, the command will copy only one file each time it is executed. In copying from disc to tape, file names can be concatenated together, e.g.

```
$COPY FILE1+FILE2 *OUT*
```

The COPY command will not write a tape mark after writing a file out to magnetic tape. The user must do this himself. However, when the tape is released, either explicitly, or implicitly when a user signs off, the system will write an end-of-file label, which comprises two tapemarks, on the tape when closing it.

ABBREVIATIONS: C CO COP

CREATE

PURPOSE:

This command may be used to create private or temporary files.

HOW TO USE:

```
$CREATE name [SIZE=size] [TYPE=type]
```

"name"
Is the name of the file to be created. It must not be the same name as a file already belonging to the current identifier.

[SIZE=n] or
[SIZE=nP]
is an optional parameter which specifies the estimated size of the file. It may be specified in one of two forms:

SIZE=n - number of 50 character (byte) lines
SIZE=nP - number of 4096 byte pages.

In specifying the size of the file it is the total number of characters or bytes which is important. Hence 100 80 character lines are approximately equivalent to 200 40 character lines. The default size parameters are the following:

SIZE=1P (for permanent files)
SIZE=10P (for temporary files)

The SIZE parameter given in the CREATE command gives an approximation to the number of bytes or characters that can be stored in the file. The actual capacity of the file is affected by the type of the file, where it is located, and how long the records or lines are that are stored in it.

The maximum allowable size parameter for a line file is 2686975 or 32767P. 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.

[TYPE=LINE] or
[TYPE=SEQ]
The file type may be line or sequential. If omitted TYPE=LINE is assumed.

When this command is given, first MTS will check to make sure a file by the given name does not already exist. Second, it will check that the user has enough free file space left to allow creation of the file. Third, MTS will attempt to acquire the space for him. If all three steps are successful, MTS will inform the user of the successful creation of the file.

ERROR MESSAGES:

Most error messages are self explanatory: however there is one which may need explanation:

INSUFFICIENT SPACE AVAILABLE.

This indicates that there was not enough room to create a file of the size indicated. Normally this would be caused only by making a mistake and including too many figures in the size specification. If this is not the case, contact a member of NUMAC staff for advice about how to minimise your file space requirements.

EXAMPLES:

$CREATE CANARY

would result in the creation of a line file of size one page - large enough to contain approximately 80 50 character lines.

$CREATE -ELEPHANT SIZE=7500

would result in the creation of a temporary file big enough to contain approximately 7500 50 character lines.

$CREATE SIMPLE SIZE=2P TYPE=SEQ

would result in the creation of a two page (8192 byte) sequential file.

ABBREVIATIONS: CR CRE

DEBUG

PURPOSE:

This command is used to load a program and enter debug command mode without initiating execution.

HOW TO USE:

```
$DEBUG [name] [loadspecs] [I/O parameters] [limits]
       [program keys] [PAR=parfield]
```

The parameters are exactly the same as those described under the LOAD command. If the program in file "name" is loaded successfully, control is transferred to debug command mode. This enables the facilities of the symbolic debugging system (SDS) to be used to display or modify parts of the loaded program and to initiate execution. SDS then monitors the execution of the program. An introduction to the SDS system is to be found in NUMAC Document "Introduction to SDS" and a complete specification is given under the Section "Debug Mode" in MTS Volume One.

EXAMPLE:

```
$DEBUG OBJPROG 5=INPUT 6=OUTPUT
```

ABBREVIATIONS: DEB

DESTROY

PURPOSE:

This command is used to destroy private files, or temporary files.

HOW TO USE:

```
$DESTROY filename {ok}
```

"filename"
is the name of the file to be destroyed. For example:

```
$DESTROY SAM
```

is a command instructing MTS to destroy the file named SAM. If the user is in conversational mode and the file is a permanent one, MTS will ask for confirmation in the following way.

```
FILE filename IS TO BE DESTROYED. PLEASE CONFIRM.
```

And prompt the user for a reply. The correct response is "OK". If the file exists MTS confirms that the file has been destroyed with the response:

```
DONE
```

If the file does not exist, MTS replies:

```
FILE TO BE DESTROYED DOES NOT EXIST.
```

Remember, confirmation is not required if the user is in batch mode or if the file is a temporary file. Any response other than OK will result in cancellation of the command. If the file does not exist, this fact will not be made known to the user until after confirmation has been obtained.

All space the file occupied is released and any information in it is lost.

EXAMPLES:

The following examples illustrate terminal sessions using this command.

```
$destroy myfile
FILE "MYFILE" IS TO BE DESTROYED. PLEASE CONFIRM.
?ok
DONE
```

```
$destroy -temp
DONE

$destroy obsolete
FILE "OBSOLETE" IS TO BE DESTROYED.  PLEASE CONFIRM.
?ok
FILE TO BE DESTROYED DOES NOT EXIST.
```

ABBREVIATIONS: DES

DISPLAY

PURPOSE:

To display system information for the user's job.

Note: This command may also be used in an expanded format to display the contents of registers and memory locations - see "MTS Volume One".

HOW TO USE:

$DISPLAY [ON name] item

"name"
is the name of the file or device on which the information is to be displayed. The default is *SINK*.

"item"
is the information required. The most useful are:

VMSIZE
the current size of the user's virtual memory as a decimal number of pages.

COST or $
specifies the accumulated cost of the current job. It does not include charges for permanent file storage, mounted tapes or unreleased paper-tape output.

SIGFILE
specifies the current and new (if any) signon files.

PDNS
specifies all user mounted pseudo-device names that are active, i.e. magnetic tapes or paper tapes. The information displayed includes the pseudo-device name, the tape name, the device type and device name.

...
specifies all active *PRINT*, *PUNCH* and *BATCH* jobs. The items displayed include the HASP job number and the amount of output or input. To ask about one specific pseudo-device name, whether system or user specified, use *pseudo-device name*, e.g. *PRINT* or *TAPE*.

EXAMPLES:

```
$display vmsize
VMSIZE=   46  PAGES

$display cost
COST=    $.67,CHARGING RATE = UNIVERSITY TERMINAL
```

```
$display sigfile
SIGFILE: "SIGFILE"

$display pdns
NO USER-MOUNTED DEVICES ACTIVE
```

<u>ABBREVIATIONS</u>: D

EDIT

PURPOSE:

To use the University of Michigan file editor to make changes to a file.

HOW TO USE:

```
$EDIT filename [:edit-command]
```

"filename"
is the name of the file which is to be edited. This may be a line file or a sequential file.

[:edit-command]
is any single editor command, which must be preceded by a ':'. The editor command is executed as a single command and control is returned immediately to MTS command mode.

The Michigan editor commands are described in the following two NUMAC documents:

"An Introduction to the University of Michigan File Editor",

"The Michigan Editor".

EXAMPLES:

```
$EDIT DATAFILE
```

The file editor is invoked to edit the line file 'DATAFILE'

```
$EDIT DATAFILE :CHANGE 10 'A'B'
```

This command changes the first occurrence of the character "A" in line 10 of the file "DATAFILE" to the character "B" and then returns to MTS command mode.

ABBREVIATIONS: ED

EMPTY

PURPOSE:

This command will empty a file: the entire contents of the file will be removed. It may be used with either temporary or private files.

HOW TO USE:

```
$EMPTY filename {ok}
```

"filename"
is the name of the file to be emptied.

This command is used in the same way as the DESTROY command. If the user is in conversational mode and the file is a private one, confirmation will be requested in the following way:

```
FILE "name" IS TO BE EMPTIED.  PLEASE CONFIRM
?
```

The correct response is "OK". "NO" or "CANCEL" will result in the command being cancelled: any other response will result in a message prompting for a recognisable response. In batch mode, or if the file is a temporary file, confirmation is not required. Unlike the DESTROY command the EMPTY command will check for existence of the file before confirmation is requested.

RESTRICTIONS:

The file cannot be emptied if it is a "read only" file. Line number ranges should not be specified. A portion of a file cannot be emptied. <u>If a line number range is specified it will be ignored and the whole file will be emptied.</u>

EXAMPLES:

```
$empty -t
DONE

$empty snug
FILE "SNUG" IS TO BE EMPTIED.  PLEASE CONFIRM
?ok
DONE

$empty nofile
FILE "NOFILE" DOES NOT EXIST.
```

```
$empty wrongfile
FILE "WRONGFILE" IS TO BE EMPTIED.  PLEASE CONFIRM
?no
COMMAND CANCELLED

$empty *source*
BUT THAT'S NOT A FILE...
```

ABBREVIATIONS: EM EMP

ENDFILE

PURPOSE:

This is not strictly a command but is used in the source stream or in a file to give an end-of-file indication.

HOW TO USE:

```
$ENDFILE
```

This command is normally only recognised from *SOURCE* or *MSOURCE* and is commonly used:

1. After source decks
2. After object decks
3. After data decks
4. To terminate COPY commands from *SOURCE*.

For example to copy data from *SOURCE* into a file:

```
$COPY *SOURCE* MYFILE
.
. Data cards
.
$ENDFILE
```

This command may also be used to send an end-of-file indication from other than *SOURCE* or *MSOURCE* although this is not the normal mode of use. To do so, the SET command must be used to turn the ENDFILE flag on as shown in the following example:

```
$SET ENDFILE=ON
```

(see the SET command)

ABBREVIATIONS: NONE

FILESTATUS

PURPOSE:

To obtain information about a file. Use of the command does not affect any items associated with the file such as the use count.

HOW TO USE:

$FILESTATUS filename(s) [format] [information]

"filename(s)"
is the name of the file(s) about which information is being sought. This parameter can have various forms:

MYFILE	specifies the file whose name is "MYFILE"
*	all permanent files of the signon identifier
-?	All temporary files of the signon identifier
id:*	all permanent files of identifier "id" for which the user must have access
?.S	all files whose names end with ".S"
A?B	all files whose names begin with "A" and end with "B"
A?Q?B	all files whose names begin with "A", end with "B" and contain "Q"
(MYFILE,DATA1,RESULTS)	the three files whose names are given.

[format]
controls the way in which the information is presented and is specified as

(1) COLUMNS	fixed-width columns with headings
(2) KEYWORD	free-form output, presented as e.g. TYPE=LINE
(3) PACKED	as for KEYWORD with the "label=" portion omitted e.g. LINE

KEYWORD is the default at a terminal, COLUMNS for output to any other file or device.

[information]
specifies the type(s) of information required. There are five groups of items:

Catalog information

OWNER	Owner signon identifier
VOLUME	The disk pack on which the file is located
USECNT	Number of references since file was created
LASTREF	Date when file was last referenced
LASTCHG	Date when file was last changed
CREDATE	Date when file was created
TYPE	Type of file (e.g., LINE)
LOC	Type of storage device on which file is located
RPM	References per month since creation date
IDLEDAYS	The number of days since the last reference
PKEY	Program key
MYACCESS	Access to the file for the current signon identifier
ACCESS	If signon identifier has permit access this gives indication of access for owner and others
FULLACCESS	If signon identifier has permit access gives full details of permit status of file.

File information

SIZE	Current file size
TRUNC	Truncated file size
MAXSIZE	Maximum file size
MAXLEN	Maximum length of a line
LINES	Number of lines
AVLEN	Average length of a line
MINSIZE	Minimum file size
HOLES	Number of holes (emptied space) in a file
MAXHOLE	Size of maximum hole
HOLESIZE	Number of bytes of holes.

Information type FILE requests the first four items and FULLFILE requests all file information.

Name information

NAME	Name of file

Summary information

When SUMMARY is specified a single summary line is produced that contains a summary for each item requested for only those files requested.

Groups of items

Apart from individual types of information, groups of types may be requested.

```
CATALOG      All catalogue information.
FILE       ] File information - see description
FULLFILE ]     under group heading above.
ALL            SIZE, MINSIZE, TYPE, ACCESS, RPM, USECNT,
               IDLEDAYS,LINES,AVLEN,MAXLEN,HOLESIZE,MAXHOLE,
               LASTREF,LASTCHG,CREDATE,VOL,OWNER
TOTAL        all information items.
```

Where "filename" is "?" or "identifier:?", the default for "information" is NAME only. Where specific filenames are given CATALOG (except for PKEY) and ACCESS information is printed by default.

EXAMPLES:

When the FILESTATUS command is given without any parameters, a list of all filenames is given.

```
$filestatus
CATALOGUE     COMCO.CONFER CONFERENCE    CONFLABELS    COPMUG
DOCASSNOS     DOCCALENDAR  DOCPROGRESS   DOCPUBL       DOCSURVEY
FICHEFMT      HELP         INFOLABELS    IUCCSRC       LABELS
MEETDEX.CTRL MEETDEX.PRIN MEETDEX.SRC    MTS.US.G1     MTSV1A.FS
MUGSET        RES1         RUNCAT1F      RUNFCAT.JCL   RUNPCAT.JCL
SIGF          SIGFILE      TXTFRMTST
```

Where one particular filename is specified all CATALOG and ACCESS information is given by default except for the program key.

```
$filestatus iuccsrc
IUCCSRC   TYPE=LINE, IDLEDAYS=20, RPM=6, USECNT=50, OWNER=CL47,
LASTREF=SEP14/78,LASTCHG=SEP14/78,CREDATE=FEB16/78,VOLUME=MTS001,
ACCESS=U/N*
```

Specific items of information, e.g. type and size, may be requested for a given list of files.

```
$filestatus (mugset,copmug,mts.us.g1) lastref type size cols

FILE NAME         LASTREF  TYPE   SIZE
                  MM/DD/YY        PGS.

MUGSET             OCT03/78 LINE     1

COPMUG             OCT04/78 LINE    57

MTS.US.G1          OCT04/78 LINE    63
```

The names of all files resident on a particular disc may be requested - in this case public pack "MTS001"

```
$filestatus volume=mts001
CATALOGUE CONFERENCE DOCCALENDAR DOCSURVEY IUCCSRC
LABELS MEETDEX.PRIN RUNFCAT.JCL TXTFRMTST
```

To ascertain the names of files on private disc packs, the command takes the form:

```
$filestatus volume=private
```

If files are named systematically full advantage can be taken of the command's string searching facilities. To find all files containing MTS jobs one might search for all files with names ending in ".JCL":

```
$filestatus ?.jcl
RUNFCAT.JCL RUNPCAT.JCL
```

Alternatively to find all files whose names begin with "DOC" and end with "S":

```
$filestatus doc?s
DOCASSNOS DOCPROGRESS
```

ABBREVIATIONS: F

LIST

PURPOSE:

This command is used to list, with line numbers, the lines from one file or device on another file or device. It is normally used to list the contents of a file on a terminal or line printer.

HOW TO USE:

```
$LIST source [ON] destination
```

"source"
is the name of the file or device containing the lines to be listed.

"destination"
is the name of the file or device on which the lines are to be listed. If omitted, this will default to *SINK* (normally the terminal in conversational mode or the line printer in batch mode).

The lines are read sequentially from "source". Their line numbers are converted to 12 characters which are appended to the front of the line before it is written to "destination". Complaint will be made if either the source or destination does not exist.

The prefix character issued by the LIST command is the ">", hex '6E'.

EXAMPLES:

In its simplest form, this command is used to list the contents of a file on *SINK*.

```
$LIST FIT3STANZA7
$LIST MERRY.S
```

The command is often used to obtain a listing of a deck of cards in the following way:

```
$LIST *SOURCE*
.
.  Deck to be listed.
.
$ENDFILE
```

In all the preceding examples the destination has been allowed to default to *SINK*. Occasionally, users have access to another device from a terminal. For example:

```
$LIST MYFILE *PRINT*
```

will list the file MYFILE on the line printer.

Portions of files may be listed by specifying line number ranges. The default starting line number is always '1'. The following command will list all the lines of the file MYLIB.

```
$LIST MYLIB(FIRST)
```

ABBREVIATIONS: L LI LIS

LOAD

PURPOSE:

The LOAD command is used to load a program but not actually start execution of it. (This is achieved by a subsequent START command.) It is normally used by assembly language programmers so that parts of a program may be altered before execution. It is also possible to determine the size of the program through use of the $DISPLAY VMSIZE command before execution begins.

HOW TO USE:

```
$LOAD [name] [loadspecs] [I/O parameters] [limits]
     [program keys] [PAR=parfield]
```

This program is identical to the RUN command except that where execution of the program would begin, control is transferred back to the user in MTS command mode. The program may be started by using the START command. The parameters are exactly the same as those for the RUN command.

EXAMPLES:

Normal use of this facility simply implies loading the file and requesting a map and cross reference if wanted and also a parameter field to be passed to the program. All other parameters may be specified on the RESTART command.

The following example will load an object deck from *SOURCE* and print a storage map and cross reference map for the loaded program:

```
$LOAD XREF MAP=*SINK*
.
. Object deck
.
$ENDFILE
```

The following example will load the deck from the file -LOAD.

```
$LOAD -LOAD
```

The following command will load the program in the file TESTFILE, print a load map and record the parameter string 'LINE,TEST' so that it may be passed to the program when execution begins.

```
$LOAD TESTFILE MAP=*SINK* PAR=LINE,TEST
```

The following example will load the program in the file -DICK and set up MTS logical I/O units 5 and 7 so that they will refer to the files INPUT and -PRINT respectively.

```
$LOAD -DICK 5=INPUT 7=-PRINT
```

ABBREVIATIONS: LOA

LOCATE

PURPOSE:

The LOCATE command determines the status of any batch jobs (MTS or OS) currently in the HASP job queues.

HOW TO USE:

```
$LOCATE [job description]
```

"job description"

identifies the job or queue about which information is required. Those job descriptions most likely to be required by users are illustrated in the examples given below. The other descriptions are designed to help operators both at the central site and remote job entry stations. A complete specification of the program LOCATE upon which this command is based is to be found in MTS Volume 2.

EXAMPLES:

To trace all the jobs belonging to one identifier :

```
$locate qclc3
?JOB 2480 QCLC3 (PRIO 6) POSN  2  EXECL Q
?JOB 2523 QCLC3370 (PRIO 5) POSN  4  PRINT Q (PTR1)
```

To trace the job with HASP number 2654 and all jobs awaiting print or punch at remote batch stations a) with mnemonic "NEW1" and b) with remote number 99 :

```
$locate
?ENTER LOCATE PARAMETERS
?2654
?*** JOB NOT FOUND OR DONE
?new1
?JOB 2787 QULJ7A PRINTING      ,PRIO= 7, PR=NEW1    ,PU=PCH1
?rmt99
?*** EMPTY Q
?$endfile
$
```

The answer to HASP job number 2654 indicates that the job is no longer in the HASP queues; either it has finished execution and printing or it failed to enter the system for some reason.

ABBREVIATIONS: LOCA

MOUNT

PURPOSE:

To mount magnetic tapes or paper tapes.

HOW TO USE:

```
$MOUNT [request[;request]...]
```

[request[;request]...]
gives details of the item(s) to be mounted. If the requests are omitted from the MOUNT command, they must be entered as separate data lines following the command and terminated by an end-of-file.

The form of "request" for mounting magnetic tapes is:

```
tapename *pseudodevice name* [keywords]
```

For further details of magnetic tape handling see Section 4.6.2.

The form of request for mounting paper tapes is:

```
tapename PPRD *pseudo-device name* [keywords]
```

For further details see Section 4.6.

EXAMPLE:

```
$MOUNT  CLD1AA *T*
```

This will mount magnetic tape CLD1AA for reading only, with pseudo device name *T*

```
$MOUNT
CLD1AB  *TIN*  VOLUME=LONXXX
CLD1ZZ  *TOUT* RING=IN
```

This will mount the magnetic tape whose rack number is "CLD1AB", and whose volume serial number is "LONXXX", as pseudo-device *TIN* for reading only and the magnetic tape, whose rack number is CLD1ZZ, with a write permit ring so that it may be written to or read as pseudo-device *TOUT*

Note: It is only necessary to give the volume serial number where this differs from the rack number: a condition usually encountered when importing labelled tapes from another installation. For further guidance please consult NUMAC Document: "Introduction to Magnetic Tapes".

ABBREVIATIONS: MOU

PERMIT

PURPOSE:

To control access to files by both the owner's identifier and those of other users.

HOW TO USE:

```
PERMIT filename how whom

PERMIT filename LIKE filename2 [EXCEPT how whom]
```

"filename"
is the name of one or more files, separated by blanks and/or commas, each of which is to be permitted as specified.

"how"
specifies the type of access to be allowed. The category names, their abbreviations and meaning are as follows:

READ,R	Read (This is the default)
WRITE,WE,APPEND, AP,WA	Write to the end of the file only.
WRITCHG,WC,CHANGE, CH,EMPTY,E	Change a file's contents and empty it.
TRUNCATE,TRUNC,T RENUMBER,RNU	Truncate and renumber
DESTROY,DES,D RENAME,RNA	Destroy and rename
PERMIT,P	Change the file's access information.

There are also names for combinations of categories:

WRITE,W	- Write to the file in any manner and empty it.
RWCHG,RWC	- Read the file, alter its contents and empty it.
RWEXP,RWE	- Read the file and add to the end of its contents.
RW	- Read, write to file in any manner and empty it.
FULL	- Everything except permit.
UNLIM,UNL,U	- Everything.
DEFAULT,DEF	- Default access i.e. read only or read and permit if the owner.
NO,NONE	- No access.

"whom"
specifies the identifiers or projects being given access to the file.

ALL	All identifiers except the owner. All previous access information for the file is discarded.
ME	The identifier issuing the PERMIT command.
OTHERS	All identifiers not explicitly mentioned. This is the default.
OWNER	The file owner's identifier.
<xxxx>	A specific identifier.
PROJNO=xxxx	Specifies a project identifier and thus a group of user identifiers.
PKEY=key	Specifies a program key - see Section 3.2.1

"filename2"
is the name of a file or a list of names whose access information is to be copied to "filename". If "filename2" is a list the copying proceeds left to right.

EXAMPLES:

```
$PERMIT DATA
```

gives read access to OTHERS, i.e. all identifiers in this case.

```
$PERMIT DATA WRITE (CCXX,CCYY,CCZZ)
```

allows identifiers CCXX,CCYY,CCZZ to write to or alter file DATA.

```
$PERMIT (A,B,C) READ ALL
```

allows all identifiers to read files A,B and C.

```
$PERMIT X LIKE Y
```

copies the access information of file Y to file X.

ABBREVIATIONS: P PER

RELEASE

PURPOSE:

This command is used to release a device, i.e. make it available for other jobs to use. It is used most frequently to release a paper tape reader or a magnetic tape drive, in which case the tape is dismounted before the device is released.

HOW TO USE:

```
$RELEASE *pseudo-device name*
```

"pseudo-device name"
specifies the device which is to be released.

The command is equivalent to

```
$CONTROL *pseudo-device name* RELEASE
```

EXAMPLES:

```
$RELEASE *TAPE*
```

will release the device associated with name "*TAPE*", usually a magnetic tape drive.

ABBREVIATIONS: REL

RENAME

PURPOSE:

To change the name of a file.

HOW TO USE:

```
$RENAME filename1 [AS] filename2 {OK}
```

"filename1"
is the name of the file to be renamed.

"filename2"
is the new name for the file.

The user must have rename access to "filename1" and "filename2" must specify a file belonging to the same identifier as "filename1" or the current identifier. Confirmation is required if a permanent file is renamed from a terminal. Temporary files may be renamed as permanent ones and vice versa. Whenever a temporary file is renamed as a permanent file, it should also be truncated to free unused space at the end of the file.

EXAMPLE:

```
$RENAME  -TEMP AS MYPRINT
$TRUNCATE MYPRINT
```

The temporary file "-TEMP" is renamed as "MYPRINT"

ABBREVIATIONS: RENA

RENUMBER

PURPOSE:

To renumber all or part of a line file

HOW TO USE:

```
RENUMBER filename [fno,lno] [TO fnewno,incr]
```

"filename"
is the name of the file to be renumbered.

[fno,lno]
are the first and last line numbers in the range to be renumbered. Defaults are FIRST and LAST respectively.

[TO fnewno,incr]
this parameter specifies the new beginning line number and the increment to be used in the renumbering. The default is "1" in each case.

Renumbering is not attempted if the process would result in duplicate line numbers or line numbers out of sequence in the file.

EXAMPLES:

```
$RENUMBER FILE1
$RENUMBER FILE1 *F,*L
$RENUMBER FILE1 TO 1,1
$RENUMBER FILE1 ,LAST TO 1
$RENUMBER FILE1 FIRST TO ,1
```

Each of the above examples will renumber the whole of "FILE1" starting with line one in increments of one.

ABBREVIATIONS: RENU

RERUN

PURPOSE:

To reissue the previous $RUN command.

HOW TO USE:

```
$RERUN [parameters]
```

"parameters"
may be any parameters which are valid for the RUN command. If given, these will override the corresponding parameters on the previous RUN command.

EXAMPLES:

If the command

```
$RUN PROG SCARDS=FILE1 SPRINT=FILE2
```

is issued, then

```
$RERUN
```

reruns program PROG. Note that, if PROG has been using files, the rerun program will begin reading from, or writing to, the files at the position specified on the previous RUN command - in this example line '1'. However if the program is using magnetic or paper tapes, these will not be rewound automatically. Input and output will continue from the point at which the tape stopped unless the user specifically rewinds it before issuing the RERUN command. If

```
$RERUN SCARDS=FILE3 PAR=TEST
```

is issued this is equivalent to the command

```
$RUN PROG SCARDS=FILE3 SPRINT=FILE2 PAR=TEST
```

ABBREVIATIONS: RER

RESTART, START

PURPOSE:

This command may be used to restart a program after an interrupt or error has been made to MTS. It may also be used to start a program after it has been loaded - see the LOAD command. START and RESTART are synonyms.

HOW TO USE:

```
$RESTART [AT] [location] [loadspecs] [I/O parameters]
```

[AT]
is a noise word which may be used to improve readability

"location"
specifies the address at which execution is to begin:

RF=hhh or
RF=GRx
specifies a relocation factor which will be applied to the address at which the program is to be started. The first form specifies a hexadecimal value to be taken as the relocation factor. The second form specifies that the content of the specified general register is to be used as the relocation factor. If omitted, the relocation factor defaults to the global relocation factor maintained by MTS through the SET command.

<hhhh>
specifies the address where the program is to be started. If omitted (the normal case) the program will be restarted at the position where the last interrupt or abnormal termination occurred. This parameter must be a hexadecimal number. The relocation factor is added to it to compute the effective starting address. This parameter specifies the right hand 32 bits of the PSW which includes the instruction length code, condition code, and program masks.

All other parameters are identical with those described for the RUN command.

This command may be used to start execution of a program after loading, or restart a program after some abnormal termination (for example a user pressing the attention interrupt key). Any logical I/O unit assignments or limits specified on a previous LOAD or RESTART command may be reassigned on the RESTART command.

EXAMPLES:

The following example shows how the START command would be used to start the execution of a program after it has been loaded.

```
$LOAD -LOAD MAP=*SINK*
.
. Commands to change or examine the loaded program.
.
$START
```

The following example shows how the command may be used to reassign I/O units for an executing program. The user has started a program from a terminal and realizes after execution begins that the program will print a great deal of output on the terminal, output he would rather have placed in a file so it can be examined later. The user presses the attention key to interrupt the program and restarts it, reassigning the output unit as he does so.

```
$run -load
EXECUTION BEGINS
  ( program output is produced which the user does
  not want printed on the terminal ..).
$ATTENTION INTERRUPT AT 7050347E
$restart sprint=-output
EXECUTION TERMINATED
```

The default assignment for SPRINT from the original RUN command was SPRINT=*SINK*. The preceding example illustrates one of the more common uses of the attention key and RESTART command to alter the way that a program is executing. Note that, since the user did not specify where the program was to be restarted, MTS restarted the program at the point where the attention interrupt occurred. This again is normal use of this command.

ABBREVIATIONS: RES ST

RUN

PURPOSE:

To load and initiate execution of a program.

HOW TO USE:

```
$RUN name [loadspecs] [ioparameters] [limits]
                [PAR=parfield]
```

"name"
is the name of the file or device that contains the object deck(s) that are to be loaded. If omitted it defaults to *SOURCE*.

"loadspecs"
These parameters specify what information the loader is to print after loading has been completed. They may be any of the following parameters:

UXREF

print a cross-reference listing of undefined external symbols on the MAP output file or device.

MAP or MAP=name

print a load map of the loaded program on the file or device "name". This defaults to *SINK* if "name" is omitted.

NOMAP

suppress printing of the load map of the loaded program.

XREF

print a cross reference listing of the external symbols on the MAP output file or device.

The defaults assumed are no cross reference and NOMAP unless the MAP= parameter is specified in which case MAP is assumed.

"I/O parameters"
specifies the assignment of logical I/O (input/output) devices for the execution of the program. These logical I/O devices with their default assignments are:

SCARDS=*SOURCE*
SPRINT=*SINK*
SPUNCH=*PUNCH* (see note below)
SERCOM=*MSINK*
GUSER=*MSOURCE*
0=,1=,...,19= (no defaults)

Note the following exceptions to the defaults above.

In HASP batch mode SPUNCH will default to the device *PUNCH* if a global card limit has been specified on the SIGNON card (see the SIGNON command).

When FORTRAN programs are run, the assignments 5=*SOURCE* and 6=*SINK* are made by default.

When running *FTN the default assignment SPUNCH=-LOAD is made by the *FTN program, not by MTS.

"limits"
Local limits on computer resources can be specified for the execution of the program. The following table shows these limits, their meaning and acceptable abbreviations for the keywords:

Limits	Meaning	Abbr.
CARDS=n	number of cards punched	C CP
TIME=n	CPU time in seconds	T
TIME=nS	CPU time in seconds	T
TIME=nM	CPU time in minutes	T
PAGES=n	number of pages printed on t	P PP
PLOTTIME=n	plot time in secs	PT
PLOTTIME=nM	or minutes	

If assigned, MTS will terminate execution of the program whenever any of these limits are exceeded. If limits are not assigned, none will be placed on the running of the program other than the global limits specified with the SIGNON command. "n" is a decimal integer. For example TIME=5S places a local time limit of five seconds CPU time on the execution of the program. The number of pages limit has meaning only to programs which are printing on one of the line printers. It does not limit the amount of output that will be printed at a terminal.

If a program exceeds a local time limit during its execution the following message will be printed.

    LOCAL TIME LIMIT EXCEEDED AT hhhh

where "hhhh" gives the address of the next instruction that would have been executed in the program. 'TIME' will be replaced by 'PAGE' or 'CARD' for the other two limits. The program may be restarted using the RESTART command if this happens. The local limit exceeded will take the default value if it is not reassigned on the RESTART command. The other limits will remain unchanged if not reassigned on the RESTART command.

[PAR=parfield]

This parameter is used to pass information to the program which is to be run. All information after the PAR= to the end of the card is passed to the program.

This is the only positional parameter. It must appear last.

The parameters are inspected to make sure that the files and devices specified exist. If not, the run is set up so that the first time the program refers to the non-existent file or device, the user is given a chance to respecify the name (conversational mode) or execution is terminated (batch mode). If spelling errors are made in the parameters and the user is in conversational mode, MTS will ask for re-entry of the parameters. In batch mode, the command will be terminated with an error. After all parameters have been scanned, the loader is called to load the program into core.

In conversational mode, the MTS loader uses a period "." as a prefix character for any conversation which must be carried on with the user (note example under LIBRARY FACILITY).

<u>LIBRARY FACILITY</u>:

If after loading there are still unresolved external references, loading will continue from the system library. Only those subprograms necessary will be loaded. The search of the system library may be suppressed through use of the SET command as shown in the following:

    $SET LIBR=OFF

If there are still unresolved references, the loader will prompt the user for the location of more loading input (conversational mode) or terminate loading with an error comment (batch mode). For example, if a conversational user has forgotten to include the object deck for the subprogram SUBA the following message would be printed:

```
. SUBA IS AN UNDEFINED SYMBOL.
. ENTER LOCN OF MORE LOADER INPUT, "CANCEL","IGNORE"
. "USMSG","UXREF", OR "MAP":
?
```

If the missing subprogram was in the file MYDECKS, the user would reply:

```
?mydecks
```

and loading would continue. If the subprogram was not available, he would enter

```
?cancel
```

and loading would terminate. If he entered

```
?map
```

the loader would print a storage map of the program as it was currently loaded and repeat the question. This feature would probably be useful only to more advanced programmers.

EXAMPLES:

The first example shows the simplest use of the command. SPRINT is allowed to default to *SINK* and the command specifies only the name of the file to be loaded.

```
$run -load
EXECUTION BEGINS
.
. Program output.
.
EXECUTION TERMINATED
```

The next example specifies that the compiled code in the file -LOAD is to be loaded together with any routines required from the library file *NAG, a storage map is to be printed and that any input requested through SCARDS is to come from the file INPUTFILE.

```
$run -load+*nag map=*sink* scards=inputfile
.
. Storage map printed here
.
EXECUTION BEGINS
.
. Program output
.
EXECUTION TERMINATED
```

The limits specifications allow a user to place a local limit on the RUN command. For example, if it is known that a program should not use more than 5 seconds of computer time for execution, the user may specify a local time limit on the run command:

```
$run myprogram sprint=-p 5=inputfile time=5
EXECUTION BEGINS
EXECUTION TERMINATED
```

If the local time limit had been exceeded, MTS would have terminated the program abnormally with the comment:

```
LOCAL TIME LIMIT EXCEEDED AT hhhhh
```

ABBREVIATIONS: R

SDS

PURPOSE:

To enter or return to DEBUG mode.

HOW TO USE:

$SDS [debug command]

Without a parameter this command transfers control to debug mode so that the facilities of the symbolic debugging system may be used, see:

"Introduction to SDS"
"MTS Volume One", Section on Debug mode.

If a debug command is specified, this is executed and control is returned immediately to the caller - normally MTS command mode.

EXAMPLES:

$SDS

Control is transferred to debug mode.

$SDS SET ERRORDUMP=ON

This sets the automatic errordumping option of SDS on and returns control to MTS command mode.

ABBREVIATIONS: SD

SET

PURPOSE:

This command allows a user to set various global switches maintained by MTS and change the meaning of various characters which have special meaning to MTS.

HOW TO USE:

```
$SET keyword=value ...
```

"keyword"
is a keyword as described in the following list and ...

"value"
is a value to be assigned to it. More than one "keyword" may be assigned in one command; the only restriction is that the pairs should be separated by one or more blanks. In the following descriptions the keyword is followed by the alternative values it may take, separated by commas. The default value is underlined. No blanks may be embedded between the keyword, the equal sign "=", and its value.

CASE=LC, UC

If UC is in effect (the default), all data lines read in MTS command mode will have lower case letters converted to upper case. CASE=LC should be specified if it is desired to enter upper and lower case letters. Note that this command does not affect lines read from a program or lines copied from *SOURCE* to a file.

CONTCHAR=char

specifies what the continuation character is to be : normally this is the minus sign, "-". If it appears as the last character in the record, it indicates that the current MTS command is continued in the next record. The user may specify any character he wishes. Thus the command:

```
$SET CONTCHAR=/
```

will set the slash as the current continuation character.

COST=OFF, ON ; $=OFF, ON

If the COST option is ON, the approximate cost (subject to roundoff) since the last cost was printed and the current cost of the session are printed after every MTS command has been executed.

CROUTE=station

This specifies the default destination for *PUNCH* output. "station" should be the mnemonic or remote number of a remote job entry (RJE) station: the default is "CNTR" denoting the central site in Newcastle - see Section 4.4.5.

EBM=characters

This controls the format of the "EXECUTION BEGINS" message. "characters" may be selected from the following ones:

W means print the words "EXECUTION BEGINS."
H means print the time of day in the form HH:MM:SS

The defaults are:

EBM=W at a terminal
EBM=WH in batch mode.

ECHO=<u>ON</u>, OFF

If the *SOURCE* device or file differs from the *SINK* device or file (for example in batch mode *SOURCE* is the card reader and *SINK* is the line printer) the command lines read from *SOURCE* are normally echoed to *SINK*. A $SET ECHO=OFF command will turn off the echoing and a $SET ECHO=ON command will restore it. The following sequence of commands enables a user to change his password in batch mode and not have the new password printed:

```
$SET ECHO=OFF
$SET PW=SESAME
$SET ECHO=ON
```

ENDFILE=ON, <u>OFF</u>

If ON, an $ENDFILE line will be recognized as an end-of-file whenever it is read, not just from *SOURCE* or *MSOURCE* as is normally the case. Fortran programs will recognize a $ENDFILE as an end-of-file regardless of the setting of this switch. This is accomplished in the FORTRAN input/ output routines and not by MTS.

ETM=characters

This controls the format of the "EXECUTION TERMINATED" message. "characters" may be selected from the following ones:

W prints the words "EXECUITON TERMINATED" or "ERROR RETURN"
H prints the time of day in the form HH:MM:SS
T prints the cpu time used to execute the program.
R prints the return code produced by the program.
$ prints the cost of executing the program.

The defaults are:

ETM=WR at a terminal
ETM=WHTR$ in batch mode.

IC=OFF, ON

If ON, implicit concatenation is set on; if OFF, no check is made for $CONTINUE WITH lines and they are treated like any other lines. This may be overridden by using the IC modifier in I/O operations.

LIBR=OFF, ON

Normally the MTS loader will search the system library if there are any unresolved external references after loading a program. If the command $SET LIBR=OFF is given, this automatic search will not be made.

LIBSRCH=OFF, name(s)

This indicates that, if LIBR=ON, specific public or private libraries are to be searched for unresolved symbols in a loaded program. If LIBSRCH=OFF only the system library (*LIBRARY) and the low core symbol directory (LCSYMBOL) are searched; otherwise LIBSRCH specifies the libraries to be searched and the order in which this is done. To specify more than one library concatenate the names together.

PRINT={PN|TN|ANY}

This specifies the default printer character set to be used for *PRINT* output - see also Section 4.7.2.

PROUTE=station

This specifies the default destination for *PRINT* output. "station" should be the mnemonic or remote number of a remote job entry (RJE) station: the default is "CNTR" denoting the central site in Newcastle - see Section 4.4.5.

PW=characterstring

This resets the current identifier's password. "characterstring", which is any sequence of from one to twelve characters, none of which is blank, specifies the new password. The latter must be correctly given during subsequent signons.

ROUTE=station

This specifies the default destination for both *PRINT* and *PUNCH* output. "station" should be the mnemonic or remote number of a remote job entry (RJE) station: the default is "CNTR" denoting the central site in Newcastle - see Section 4.4.5.

SIGFILE=<u>OFF</u>, name

This specifies the name of a file which is to be used as an implicit $SOURCE file after the user signs on.

S8=characters

This option resets the default jobname ("S8" number) of a job from "Q<MTS identifier>" to the given string of from 5 to 8 characters. Care is needed when choosing jobnames. Characters 4 and 5 are used by the operators at the central site to sort and dispatch output. A jobname other than the default could result in output being sent to an unexpected batch station.

TDR=ON, <u>OFF</u>

If TDR=ON, MTS will print the elapsed CPU time in seconds and the number of drum reads which have occured since the last command was terminated. This parameter allows a user to obtain a timing estimate for the running of a program.

TERSE=ON, <u>OFF</u>

If this option is ON messages from MTS are suppressed or abbreviated.

TIME=t,tS,tM

This sets a default local call time limit for all RUN, RERUN, START, RESTART, LOAD or DEBUG commands which do not give an explicit limit.

TRIM=ON,OFF

If this option is ON, all but one of any trailing blanks are removed from lines read or written to files. The keyword can be overridden by the @TRIM modifier see Section 3.8.

VERBOSE=ON,OFF

If this option is OFF, many informative and error messages are suppressed or abbreviated. Otherwise all messages are given in full. BRIEF and TERSE are antonyms for VERBOSE.

*LIBRARY=ON,OFF

If this option is ON, the file *LIBRARY is searched for any unresolved external symbols after a program is loaded, provided that the the LIBR option is also ON.

EXAMPLES;

$SET PROUTE=DURH

The default destination for printed output is changed from the central site to the Durham remote printers.

$SET S8=QCL47EG1 COST=ON

The default jobname is changed from the current signon identifier preceded by a "Q" to "QCL47EG1" and running costs for the session will be printed after every MTS command has been executed.

ABBREVIATIONS: SE

SIGNOFF

PURPOSE:

This command notifies MTS that the user wishes to sign off from the system.

HOW TO USE:

$SIGNOFF [SHORT|$]

All storage, devices and files that the user may have been using are released. MTS then types a summary of the system resources used during the run and the approximate cost of the run. If SHORT is specified, the summary is abbreviated so that, if the user is at a conversational terminal, the typing will take a shorter time. If $ is specified, only the approximate cost of the session is given.

EXAMPLES:

```
$SIGNOFF
$SIGNOFF SHORT
$SIGNOFF $
```

The following example illustrates the type of output which will be printed by MTS when the SIGNOFF command is encountered.

```
$signoff
OFF AT 13:36:21
ELAPSED TIME       32.16   SEC.
CPU TIME USED        .071  SEC.
STORAGE USED         .27   PAGE-SEC.
DRUM READS          4
MAX VM SIZE        23 PAGES
APPROX COST OF THIS RUN    C$.04
FILE STORAGE     608 PG-HR.     $.20
```

Details of this output may differ slightly depending on what resources were used, and on whether the user was signed on in conversational mode or batch mode.

ABBREVIATIONS: SIG

SIGNON

PURPOSE:

This command is used to identify a user to MTS and to allow him to request system resources.

HOW TO USE:

$SIGONON id [limits] [haspspecs] ['comment']

When MTS reads a SIGNON command, it checks to see that the user has a valid ID, prompts for the password, checks this and then that there are sufficient funds remaining. If these checks are satisfactory, further commands are accepted from the source file or device.

"id"
specifies a four character user identifier. It must be the first parameter on the SIGNON command.

[limits]
These parameters are used to control the use of system resources. They should be specified as follows:

TIME=n (CPU time in seconds)
TIME=nS (CPU time in seconds)
TIME=nM (CPU time in minutes)
PAGES=n (total pages printed on line printer)
CARDS=n (total cards to be punched)
PLOTTIME=tM (estimated plotting time)

where "n" is a decimal integer representing the value of the limit. If CARDS= is not specified the logical device SPUNCH will not default to *PUNCH*. The specified limits are used to calculate the HASP priority; see Section 4.2.

[haspspecs]
specifies parameters for the HASP spooling system. These are only applicable in batch mode.

COPIES=n
specifies the number of copies of output wanted. If omitted, COPIES=1 is assumed. The page limit for the job must specify enough pages to cover the total number that will be printed for all copies.

PRINT=forms number
specifies that output is to be queued for printing in a special way. A table of the current special forms numbers is printed in Section 4.7.2 and this will be updated in the NUMAC Newsletter from time to time.

CROUTE=station (card output)
PROUTE=station (printed output)
ROUTE=station (all output)
These three options specify the location at which printed and punched output is to be produced. "station" is the remote job entry station's number or mnemonic. The default is the station from which the job is submitted.

['anything']
As the last parameter on the card, the user may specify a string of characters between single quote marks (') which will serve to identify the run or as a postal address.

EXAMPLES:

The following is an example of a signon procedure at a conversational terminal.

```
$signon ccx9 'demo run'
ENTER USER PASSWORD.
sesame
CHARGING RATE = UNIVERSITY, TERMINAL
**LAST SIGNON WAS: 10:51:02 08-15-78
USER "CCX9" SIGNED ON AT 13:35:49 ON TUES 15 AUG 78
```

The following is an example signon procedure for a batch job.

```
$SIGNON CCX9 TIME=5M PAGES=75 COPIES=2 ROUTE=DURH
MYPW
```

The job will be allowed to use up to five minutes of CPU time, print up to 75 pages of output, and two copies of the output will be printed at the remote batch station whose mnemonic identifier is "DURH".

ABBREVIATIONS: SIG

SINK

PURPOSE:

This command is used to change the assignment of the pseudo-device *SINK*. This is the default destination for output from a job: it may be changed to point to another file or device.

HOW TO USE:

```
$SINK name
$SINK PREVIOUS
```

"name"
is the name of the file or device that is to be the new sink. The appearance of this command causes any following output directed to *SINK* to be output to the file or device specified. The terminal, in conversational mode, or the line printer, in batch mode, always remains connected as the master sink (*MSINK*) on which error messages are printed.

The previous sink device in use is saved by MTS. If:

```
$SINK PREVIOUS
```

is given, the previous sink device or file is restored.

Initially, the pseudodevice *SINK* is the terminal in conversational mode, or the line printer in batch mode. At a conversational terminal, the meaning of *SINK* will be restored to the terminal if the attention interrupt key is pressed.

EXAMPLES:

This command is normally used to obtain a complete record of a run, including commands, in a file for later analysis, e.g. to place all output directed to *SINK* in the permanent file RECORD:

```
$SINK RECORD
```

If the previous assignment of *SINK* had been the default one of *MSINK*, the command:

```
$SINK PREVIOUS
```

would restore the original situation, otherwise

```
$SINK *MSINK*
```

must be used to achieve this.

ABBREVIATIONS: SIN

SOURCE

PURPOSE:

This command is used to change the meaning of the pseudo-device *SOURCE*. This is the source for all commands and data lines and may be changed to point to another file or device.

HOW TO USE:

```
$SOURCE name
$SOURCE PREVIOUS
```

"name"
is the name of the file or device which is to be taken as the new source. Execution of this command causes any following input lines to be taken from the file or device specified. In conversational mode, users will always remain connected as the master source *MSOURCE* from which attention interrupts occur. In batch mode MTS will restore *SOURCE* to *MSOURCE* (the card reader) if it encounters an error while reading commands or data lines.

The previous source file or device in use is always saved by MTS. The command:

```
$SOURCE PREVIOUS
```

will restore this previous file or device. Also, in conversational mode, pressing the attention interrupt key will always restore the pseudo-device *SOURCE* to *MSOURCE*, i.e the terminal. If this occurs, processing of commands from the previous source file cannot be restarted through use of the $SOURCE PREVIOUS command. If an end-of-file is encountered from the source file or device the input lines will again be taken from *MSOURCE*, that is the terminal in conversational mode or the card reader in batch mode.

Commands which are read from the new source file or device will be echoed to *SINK*. This action may be stopped through use of the command:

```
$SET ECHO=OFF
```

- see the SET command.

EXAMPLES:

This command is normally used when a user wishes to set up a series of commands in a file so that he can execute the full series with a single SOURCE command. The following series of commands will build a file of commands to compile a FORTRAN program, then execute the program it produces.

```
$CRE RUNPROG
$COPY *SOURCE* RUNPROG
$RUN *FTN SCARDS=MYPROG
$RUN -LOAD SCARDS=INPUTFILE 7=*DUMMY* TIME=5S
$SOURCE PREVIOUS
$ENDFILE
$SOURCE RUNPROG
```

The $SOURCE RUNPROG command will direct MTS to take the next command from the file RUNPROG. It will then execute the $RUN *FTN command and then the $RUN -LOAD command. The source will return to *MSOURCE* when the end-of-file is detected.

ABBREVIATIONS: SO

SYSTEMSTATUS

PURPOSE:

To provide information about the current workload of the system.

HOW TO USE:

$SYSTEMSTATUS [systemstatus command]

[systemstatus command]
is any single systemstatus command. If given, this is executed and an immediate return is made to MTS command mode. Otherwise systemstatus mode is entered with prefix character ".", and the input lines are treated as systemstatus commands until one of the following is encountered: "end-of-file", MTS, MCMD, RETURN or STOP.

There are a number of systemstatus commands but those given in the following examples are most likely to be of general use to NUMAC users. The others are given in MTS Volume One.

EXAMPLES:

```
$systemstatus tasks type 9tp
-00078 MTS 02C4C0 16 READY GGN6 DGG0; TOC0 TOC1 HSL1TT06
-00124 MTS 02F440 28 I/O ON TOC2 N101 JNB0; TOC2
```

This command displays the number of jobs currently using 9-track tape decks (maximum 4) and thus whether there may be any free ones. "TASKS" may be abbreviated to "T".

Note: Apparently free decks may in fact be booked - see "NUMAC Service Documents" for details of tape deck booking scheme.

```
$systemstatus
-users
-THERE ARE 72 TERMINAL USERS, 8 BATCH JOBS, 86 AVAILABLE LINES,
-AND 33 NON-MTS JOBS USING 3811 VIRTUAL PAGES AND 394 REAL PAGES
-stop
```

This displays details of the current MTS system workload. "USERS" may be abbreviated to "U".

ABBREVIATIONS: SY

TRUNCATE

PURPOSE:

To release unused space at the end of a file.

HOW TO USE:

```
$TRUNCATE filename
```

"filename"
is the name of the file to be truncated.

EXAMPLE:

```
$TRUNCATE LONGFILE
```

ABBREVIATIONS: T

UNLOAD

PURPOSE:

This command is used to release core storage and devices being retained by MTS from the last LOAD command or because the last program run did not terminate normally.

HOW TO USE:

$UNLOAD

All core storage and devices currently being held by MTS are released.

Whenever a program terminates normally, MTS automatically releases the core storage and devices the program was using except when the program was loaded via the DEBUG command. If, however, the program terminates abnormally (normal termination is via the MTS subroutine SYSTEM or through a normal return to MTS evidenced by the printing of EXECUTION TERMINATED) MTS will not release the core storage and devices as the user may want to RESTART the program. The user is, however, charged for the core storage and if it is not intended to restart the program it is wise to issue an UNLOAD command, especially if the program is large.

ABBREVIATIONS: UNLO

APPENDIX B: PROGRAMS AVAILABLE IN PUBLIC FILES

In an effort to to help people find programs that may be useful in their work, a number of categories of activity have been defined. Under each category are listed the names of the public files containing programs that might be of use in that activity. The file names are listed alphabetically within each category. Detailed descriptions of these programs are to be found in MTS Volume 2 : "Public File Descriptions".

ASSEMBLERS

*ASMG	System 360/370 G Assembler
*ASMT	System 360/370 TSS Assembler
*ASMTIDY	Tidies Assembler source listings
*IDASS	Cambridge Interdata/70 assembler
*MADSAM	Motorola M6800 Advanced Symbolic Assembler - a cross-assembler
*MAL	PDP-11 assembler - Merit Assembly Language
*NOVA	Data General Corporation Nova/Supernova assembler
*OSMAC	macro library of IBM's Operating System
*11PAL	PDP-11 cross-assembler and simulator
*1130ASM	assembler for IBM 1130 and 1800 source code
*11ASR	PDP-11 assembler
*8IASS	Iowa University PDP-8 assembler
*8LINK	PDP-8 link-editor/loader

FORTRAN

*DZERO	converts F-type and E-type constants to D-type
*FORTEDIT	converts free-format statements to fixed-format ones
*FORTRAN	IBM FORTRAN G compiler OS Release 21.8
*FORTRANG	FORTRAN G callable as a subroutine
*FORTRANH	IBM FORTRAN H compiler OS Release 21
*FORT5LIB	library to facilitate use of UNIVAC FORTRAN V programs
*FTN	interface program to FORTRAN G compiler
*IF	University of British Columbia Interactive FORTRAN
*TIDY	renumbers and edits FORTRAN source programs

OTHER LANGUAGES

*ALGOL	IBM OS/360 Algol compiler
*ALGOLW	Stanford AlgolW compiler
*STRAW	Medusa project AlgolW input/output interface
*APL	IBM APL-SV interpreter
*BASIC	Michigan BASIC system
*COBOL	OS Release 19 COBOL F compiler - modified
*COBOLU	IBM ANSI standard COBOL compiler
*LISP	LISP/360 interpreter for Stanford LISP 1.5
*ML1	general purpose stream oriented macroprocessor
*PLC	version 7.1 of the Cornell compiler for PL/I
*PL1	MTS version of the F-level PL/I compiler

*PL1TIDY	edits MTS PL/I source programs into an easily readable form
*PL360	PL360 compiler
*REDUCE2	language for general algebraic computations of interest to physicists and engineers
*SIMULA	Norwegian Computer Centre general purpose and discrete event simulation language
*SNOBOL4	version 3 of the SNOBOL4 interpreter without BLOCKS
*SNOBOL4B	version 3 of the SNOBOL4 interpreter with BLOCKS
*STAGE2	Culham Laboratory general purpose record oriented macro processor
*UMIST	interpreter for UMIST interactive text processing language
*WATBOL	University of Waterloo's ANSI COBOL compiler
*WATFIV	University of Waterloo's FORTRAN IV compiler
*ALGOLW	AlgolW compile-and-go compiler/monitor

CONVERSION

*BCDEBCD	BCD to EBCD code conversion
*CONVSNOBOL	reads and executes SNOBLO4 statements
*EBCDBCD	EBCDIC to 7090 Augmented BCD code
*EBCDTO1900	EBCDIC to ICL 1900 card code
*FTNTOPL1	FORTRAN to PL/1 conversion
*TRANSNOBOL	SNOBOL3 to SNOBOL4 translator
*1900TOEBCD	ICL 1900 to IBM EBCDIC card code

GRAPHICS

*AWPLOT	subroutines for Algolw programs which plot at Durham
*DURPLOT	*PLOTSYS post processor for Durham plotter output
*GINOF	graphical subroutines from CAD, Cambridge
*GINOGRAF	library using more primitive subroutines from *GINOF
*GPCP	Calcomp General Purpose Contouring Program
*IG	FORTRAN-callable subroutines for drawing and manipulating pictures
*ORTEP	Oak Ridge Thermal Ellipsoids Plot program
*OVERPLOT	superimposes plotter pictures stored in plot data files
*PLOT	produces plots on lineprinters and terminals
*PLOTLIB	generates pictorial output for a drum plotter
*PLOTSEE	previews plots on any IG supported graphics device
*PLOTSYS	University of Michigan Plot Description System subroutine library
*PL1PLOT	allows PL/1 source programs to call PDS subroutines of *PLOTSYS
*UNEPLOT	plots *PLOTSYS output on Newcastle plotter

SIMULATION

*CSMP	IBM's Continuous System Modelling Program
*GPSS	IBM's GPSS/360 version 1, modification level 4
*SIMULA	Norwegian Computer Centre general purpose and discrete event simulation language

STATISTICAL APPLICATIONS

*ANOVAR	University of British Columbia - ANalysis Of VARiance
*APL	IBM's APL-SV interpreter
*BMD	University of California - Biomedical Computer Programs
*BMDP	P series of BMD programs
*BMDPT	special purpose version of BMDP
*CLUSTAN	CLUSTAN-1A package
*MIDAS	Michigan Interactive Data Analysis System
*OSIRIS	Michigan's OSIRIS III, version 1 : management and analysis of data
*SPACES	interactive non-metric multidimensional scaling of data
*SPIRES	Stanford Public Information REtrieval System - data base management
*SPSS	Statistical Package for the Social Sciences - version 7
*VALIDATA	data validation program

NUMERICAL APPLICATIONS

*ALGWLIB	mathematical routines for AlgolW
*DOUBLE	generates double-precision floating point representation of a given decimal number
*HARWELL	subroutines for mathematics and numerical analysis
*LINPG	IBM LINear ProGramming subroutines
*GLIM	Generalised LInear Modelling program (version 2)
*NAG	mathematical and numerical analysis subroutines
*REDUCE2	language for general algebraic computations of interest to physicists and engineers

SUBROUTINE LIBRARIES

*ALGWLIB	mathematical subroutines for AlgolW
*COBLIB	COBOL subroutine library
*HARWELL	subroutines for mathematics and numerical analysis
*LIBRARY	MTS subroutine library - see Appendix C and MTS Volume 3
*NAG	subroutines for mathematics and numerical analysis

OTHER APPLICATIONS PACKAGES

*COCOA	text concordance and vocabulary statistics - Atlas Lab., Chilton
*FAKEOS	OS/360 environment simulation program
*FMT	University of British Columbia text processing program
*LINPG	IBM LINear ProGramming subroutines
*PAFEC	Programs for Automatic Finite Element Control engineering calculations - Nottingham
*PMS	IBM's Project Management System version II
*WIREWRAP	general purpose logic design program

MAGNETIC TAPES

*DOWNDATE	compares two files and produces *UPDATE commands
*UPDATE	copies, edits and reblocks tapes or files of card images
*FS	to save and restore magnetic tape files
*IEBUPDAT	IBM utility to copy tapes of card images
*LABELSNIFF	prints IBM and/or ANSI tape labels
*LABEL	labels magnetic tapes
*TAPECOPY	copies magnetic tapes
*TAPEDUMP	dumps magnetic tapes or files in character or binary format
*TAPEFIXER	recovers data from a damaged magnetic tape
*TAPESNIFF	examines the contents of a magnetic tape

FILE MANIPULATION

*AMENDS	convert two line files and generate commands to convert one to the other
*DOWNDATE	compares two files and produces *UPDATE commands
*FILEDUMP	prints hexadecimal dump of contents of file or device
*FILEMOVE	copies files from one disk to another
*FILEPUNCH	copies files to cards
*FILESCAN	locates specified lines in a file and prints them
*TAPEDUMP	dumps magnetic tape or file in character or binary format
*UNEDIT	generates commands to edit the original version of a file to match the current one
*UPDATE	copies,edits and reblocks tapes or files of card images
*VALIDATEFILE	checks internal consistency of MTS line files

OBJECT CODE MANIPULATION

*ENDJUNK	prevents *LIBRARY search for assembler programs
*LCS	reduces loading time for object modules if there are no references to library subroutines
*LINKEDIT	link edits files of object modules
*OBJCONV	converts OS FORTRAN H object decks to MTS ones
*OBJLIST	lists object modules
*OBJSCAN	reports on contents of object file
*UNLINKER	converts OS load modules into MTS ones

UTILITIES

*ASA	converts ASA printer control characters into MCC ones
*BATCH	aids the submission of batch jobs
*DISKSAVE	saves a disk pack's files to magnetic tape
*DOUBLE	generates double-precision floating point representation of a given decimal number
*FULLPAGE	prints output as one continuous sheet across the physical printer pages
*GETDISK	attachs private disks to a user's task
*HEXLIST	lists object cards in hexadecimal

*IEHMOVE	loads unloaded datasets from an input tape (IBM)
*LISTVTOC	prints information from the Volume Table Of Contents of an MTS disk
*MACGEN	macro library generator
*MVC	rearranges records
*SCRAMBLE	scrambles and unscrambles information in an MTS line file
*SORT	sorts, merges, blocks and deblocks data
*STATUS	prints user's accounting information
*TABEDIT	tab editing program
*TALLY	gathers run time statistics for MTS programs
*TIME	prints current date and time
*TIMETALLY	monitors program execution and prints histogram of distribution of cpu activity
*TOTPAGES	computes number of unused pages on an MTS disk
*UPDATE	copies,edits and reblocks tapes or files of card images
*USERS	shows how many people are using MTS
*VALIDATA	data validation program

APPENDIX C: SUBROUTINES AVAILABLE IN MTS STANDARD LIBRARY

In an effort to help users find subroutines that may be useful in their work, a number of subject categories have been defined. Under each category is listed the name of the appropriate subroutine description, the purpose of the subroutine, and whether the subroutine is callable from assembly language and/or FORTRAN. Detailed descriptions of these subroutines are to be found in MTS Volume 3 : Subroutines.

CHARACTER AND NUMERIC CONVERSION

ASCEBC	USASCII to EBCDIC translation	Assembly
BINEBCD	binary input to EBCDIC translation	Assembly
BINEBCD2	binary input to EBCDIC translation	Assembly
CASECONV	lowercase to uppercase conversion	Assembly
CVTOMR	OMR card image to EBCDIC translation	Assembly, FORTRAN
EBCASC	EBCDIC to USASCII translation	Assembly
E7090,D7090,E7090P,D7090P	7090 to 360 floating-point conversion	Assembly, FORTRAN
IOH	numeric input/output conversion	Assembly
SIOC	numeric input/output conversion	Assembly, FORTRAN
SIOCP	numeric input/output conversion	Assembly, FORTRAN

DATE AND TIME CONVERSION

GRGJULDT	gregorian to julian date and time	Assembly
GRGJULTM	gregorian to julian time	Assembly
GRJLDT	gregorian to julian date and time	FORTRAN
GRJLSEC	gregorian to julian time	Assembly
GRJLTM	gregorian to julian time	FORTRAN
GROSDT	gregorian to OS date	Assembly, FORTRAN
GTDJMS	gregorian to julian date and time	FORTRAN
GTDJMSR	gregorian to julian time	Assembly
JLGRDT	julian to gregorian date and time	FORTRAN
JLGRSEC	julian to gregorian time	Assembly
JLGRTM	julian to gregorian time	FORTRAN
JMSGTD	julian to gregorian date and time	FORTRAN
JMSGTDR	julian to gregorian date and time	Assembly
JTUGTDR	julian to gregorian date and time	Assembly
JULGRGDT	julian to gregorian date and time	Assembly
JULGRGTM	julian to gregorian time	Assembly
OSGRDT	OS to gregorian date	Assembly, FORTRAN
TIME	get time of day, cpu and elapsed time	Assembly, FORTRAN

FILE AND DEVICE USAGE

CFDUB	compare fdub-pointers	Assembly, FORTRAN
CHGFSZ	change file size	Assembly, FORTRAN
CHGMBC	change number of file buffers	Assembly, FORTRAN
CHGXF	change file expansion factor	Assembly, FORTRAN
CHKACC	check access to file	Assembly, FORTRAN
CHKFDUB	get a fdub-pointer for a file	Assembly, FORTRAN
CHKFILE	determine existence of a file	Assembly, FORTRAN
CLOSEFIL	close a file	Assembly, FORTRAN
CNTLNR	count number of lines in a file	Assembly, FORTRAN
CREATE	create a file	Assembly, FORTRAN
DESTROY	destroy a file	Assembly, FORTRAN
EDIT	edit a file	Assembly, FORTRAN
EMPTY	empty a file	Assembly, FORTRAN
EMPTYF	empty a file	Assembly, FORTRAN
FNAMETRT	check for legal file name	Assembly
FREEFD	free a file or device	Assembly, FORTRAN
FSIZE	determine size required for a file	Assembly, FORTRAN
FSRF,BSRF	forward and backspace records in a file	Assembly, FORTRAN
GDINF	get file information	FORTRAN
GDINFO	get file or device information	Assembly
GDINFO2	get file or device information	Assembly
GDINFO3	get file or device information	Assembly
GETFD	get a file or device	Assembly, FORTRAN
GETFST,GETLST	get first and last line numbers of a line file	Assembly, FORTRAN
GFINFO	get file and catalog information	Assembly, FORTRAN
LETGO	periodically unlock and lock a file	Assembly, FORTRAN
LOCK	lock a file	Assembly, FORTRAN
NOTE	remember sequential file pointers	Assembly, FORTRAN
PERMIT	permit a file	Assembly, FORTRAN
POINT	change sequential file pointers	Assembly, FORTRAN
RENAME	rename a file	Assembly, FORTRAN
RENUMB	renumber a file	Assembly, FORTRAN
RETLNR	return line numbers of a file	Assembly, FORTRAN
REWIND	rewind a logical I/O unit	FORTRAN
REWIND#	rewind a file or magnetic tape	Assembly
SETFPRIV	make a file privileged	Assembly, FORTRAN
SETFSAVE	to enable or disable file saving	Assembly, FORTRAN
SETKEY	set program key for a file	Assembly, FORTRAN
SETLNR	set line numbers of a file	Assembly, FORTRAN
TRUNC	truncate a file	Assembly, FORTRAN
UNLK	unlock a file	Assembly, FORTRAN
WRITEBUF	write file buffers	Assembly, FORTRAN

FORTRAN USAGE

ADROF	get address of a FORTRAN variable	FORTRAN
ARRAY MANAGEMENT ROUTINES		
	array processing for FORTRAN	FORTRAN
ATNTRP	attention interrupt processing	FORTRAN
BITWISE LOGICAL FUNCTIONS		
	FORTRAN bitwise logical functions	FORTRAN
CHARACTER MANIPULATION ROUTINES		
	character processing for FORTRAN	FORTRAN
DUMP,PDUMP	dump storage	FORTRAN
FREAD	free format input	FORTRAN
FTNCMD	execute FORTRAN I/O library command	FORTRAN
GDINF	get file information	FORTRAN
GRJLDT	gregorian to julian date and time	FORTRAN
GRJLTM	gregorian to julian time	FORTRAN
GTDJMS	gregorian to julian date and time	FORTRAN
JLGRDT	julian to gregorian date and time	FORTRAN
JLGRTM	julian to gregorian time	FORTRAN
JMSGTD	julian to gregorian date and time	FORTRAN
LINKF	dynamic loading	FORTRAN
LOADF	dynamic loading	FORTRAN
LOGICAL OPERATORS		
	FORTRAN logical machine operations	FORTRAN
RCALL	R-type call from FORTRAN	FORTRAN
REWIND	rewind a logical I/O unit	FORTRAN
SIOERR	I/O error processing	FORTRAN
STARTF	dynamic loading	FORTRAN
TICALL	timer interrupt processing	FORTRAN
UNLDF	dynamic unloading	FORTRAN

INPUT/OUTPUT ROUTINES

BLOCKED I/O ROUTINES		
	read and write blocked records	Assembly, FORTRAN
FREAD	free format input	Assembly
GUSER	read from logical I/O unit GUSER	Assembly, FORTRAN
LIOUNITS	table of valid logical I/O units	Assembly
READ	read a record	Assembly, FORTRAN
READBFR	read without knowing length	Assembly
REWIND	rewind a logical I/O unit	FORTRAN
REWIND#	rewind a magnetic tape or file	Assembly
SCARDS	read from logical I/O unit SCARDS	Assembly, FORTRAN
SERCOM	write on logical I/O unit SERCOM	Assembly, FORTRAN
SETIOERR	I/O error processing	Assembly
SETLIO	set logical I/O unit	Assembly, FORTRAN
SIOERR	I/O error processing	FORTRAN
SPRINT	write on logical I/O unit SPRINT	Assembly, FORTRAN
SPUNCH	write on logical I/O unit SPUNCH	Assembly, FORTRAN
WRITE	write a record	Assembly, FORTRAN

INTERRUPT PROCESSING

ATNTRP	attention interrupt processing	FORTRAN
ATTNTRP	attention interrupt processing	Assembly
GETIME	timer interrupt processing	Assembly
PGNTTRP	program interrupt processing	Assembly
RSTIME	timer interrupt processing	Assembly
SETIME	timer interrupt processing	Assembly
SETLCL	to set a local time limit	Assembly, FORTRAN
SPIE	program interrupt processing	Assembly
TICALL	timer interrupt processing	FORTRAN
TIMNTRP	timer interrupt processing	Assembly
TRACER	elementary function library error processing	Assembly, FORTRAN
TWAIT	timer interrupt processing	Assembly

PL/I USAGE

ATTACH	attach PL/I files	PL/I
BATCH	terminal or batch status	PL/I
CNTL	execute a device support operation	PL/I
CPUTIME	get cpu time	PL/I
ELAPSED	get elapsed time	PL/I
FINFO	get file or device information	PL/I
IHEATTN	attention interrupt processing	PL/I
IHENOTE	remember sequential pointers	PL/I
IHEPNT	change sequential pointers	PL/I
IHEREAD	read from PL/I	PL/I
IHERITE	write from PL/I	PL/I
NEXTKEY	find key of next PL/I record	PL/I
PLCALL	S-type call from PL/I	PL/I
PL1ADR	get address of a PL/I variable	PL/I
PL1RC	determine return code from subroutine	PL/I
RAND	uniform random numbers	PL/I
SIGNOFF	signoff the user	PL/I
USERID	get user ccid	PL/I

STATUS OF USER AND SYSTEM

CANREPLY	terminal or batch status	Assembly, FORTRAN
CNFGINFO	get system configuration information	Assembly
COST	get cost of current signon	Assembly, FORTRAN
CUINFO	change user status information	Assembly
GUINFO	get user status information	Assembly
GUINFUPD	update user status information	Assembly
GUSERID	get user ccid	Assembly, FORTRAN
LOADINFO	get symbol or address information	Assembly

SYSTEM UTILITIES

BLOKLETR	produce block letters	Assembly
CALC	call $CALC routines	Assembly, FORTRAN
CHARGE	computes charges for resources	Assembly, FORTRAN
CMD	execute an MTS command	Assembly, FORTRAN
CMDNOE	execute MTS command without echoing	Assembly, FORTRAN
CONTROL	execute a device support operation	Assembly, FORTRAN
DISMOUNT	dismount a tape	Assembly, FORTRAN
ERROR	terminate execution with error	Assembly, FORTRAN
GRAND	normally distributed random number	Assembly, FORTRAN
KEYWRD	keyword processing	Assembly
KWSCAN	keyword processing	Assembly
MOUNT	mount a tape	Assembly, FORTRAN
MTS	return to MTS command mode	Assembly, FORTRAN
MTSCMD	return to MTS and execute a command	Assembly, FORTRAN
PRINTER PLOT ROUTINES		
	produce plots	Assembly, FORTRAN
QUIT	signoff user at next MTS command	Assembly, FORTRAN
SETLIO	assign logical I/O units	assembly, FORTRAN
SETPFX	set prefix character	Assembly, FORTRAN
SKIP	space a magnetic tape	Assembly, FORTRAN
SORT	sort and merge records	Assembly, FORTRAN
SORT2	sort vectors	Assembly, FORTRAN
SORT3	sort vectors	Assembly, FORTRAN
SPELLCHK	spelling check	Assembly, FORTRAN
SYSTEM	terminate execution	Assembly, FORTRAN
URAND	uniformly distributed random number	Assembly, FORTRAN

VIRTUAL MEMORY MANAGEMENT

DUMP,PDUMP	dump storage	FORTRAN
FREESPAC	release storage	Assembly, FORTRAN
GETSPACE	acquire storage	Assembly, FORTRAN
GPSECT, FPSECT, QPSECT		
	PSECT storage management	Assembly
LINK	dynamic loading	Assembly
LINKF	dynamic loading	FORTRAN
LOAD	dynamic loading	Assembly
LOADF	dynamic loading	FORTRAN
LOADINFO	get loader table information	Assembly, FORTRAN
LODMAP	produce loader map	Assembly, FORTRAN
SCANSTOR	scan storage blocks	Assembly
SDUMP	dump storage and registers	Assembly
STARTF	dynamic loading	FORTRAN
STDDMP	dump storage	Assembly
UNLDF	dynamic unloading	FORTRAN
UNLOAD	dynamic unloading	Assembly
XCTL	dynamic loading	Assembly
XCTLF	dynamic loading	FORTRAN

INDEX

MTS USERS GUIDE

COMMENT SHEET

We are anxious to achieve a high standard of documentation for users of the NUMAC system, but we cannot do this without your help. If you have noticed any errors or omissions in this document, please use this form to tell us about them. We also welcome any suggestions for ways of improving later editions.

1. Do you find this document adequate?

2. Can you suggest any improvements?

3. Please note any specific errors found, with page and paragraph reference.

4. General Comments

FROM Name Dept.

University

Date:

Please return this form to the Information Officer, Computing Laboratory, Claremont Tower, Claremont Road, Newcastle upon Tyne.

Thank you.