UNIVERSITY OF EDINBURGH

Edinburgh Regional Computing Centre

EMAS User's Guide Update 1

Originator		Issue date
Roderick McLeod		January 1978

Replace CONTENTS line 28 by:

Replace page 2 line 20 by:
14, and 0 in the main store (see figure 4).

Delete page 9 lines 19-22 inclusive (Interactive graphics)

Replace page 14 last two lines by:
The size of an existing file can be altered by its owner, to a value between 1 and 1024 pages. This function is requested by the Subsystem, as required.

Replace page 45 lines 21,22 by:
of the file, and the second parameter, which is optional, is an output file or output device. By default output goes to the interactive terminal.

Add to foot of page 48:
* Unlike LIST, SEND does not start the listing by giving the name of the file.

Replace page 61 line 23 by:
* As optional output files from commands; for example, FILES(,A,OUT) produces a

Replace page 62 line 17 by:
* as files read in BINARY mode from paper tape

Replace page 69 line 29 by:
preceded by a minus sign. A single asterisk ('*') can be used to mean 'very large positive integer'.

Replace page 97 line 5 by:

```
                    Files        1:132
```

Replace page 100 line 9 by:
* Compile this using the compile time option DYNAMIC, together with any other

Replace page 102 line 16 by:

```
Output Card Punch (.CP)                          Data File    Data File
```

Replace page 111 line 6 by:
* Any command that would write to the file; for example, FILES(',,OUT')

Replace page 128 line 6 by:
DETACH, HELP(.LP) or LIST.

Renumber page 137 as page 141

Remove	Insert	
Preface	Preface	
pp 7,8	pp 7,8	
pp 11,12	pp 11,12	
p 15	pp 15,16	
pp 17-21	pp 17-20	Note p 21 no longer exists.
pp 25,26	pp 25,26	
pp 27-34	pp 27-34	
pp 35-43	pp 35-42	Note p 43 no longer exists.
pp 67,68	pp 67,68	
pp 73-77	pp 73-77	
p 117	pp 117,118	
pp 119-121	pp 119,121	
pp 123-126	pp 123-126	
No pages	pp 137-139	
Index	Index	

PREFACE

This manual describes the Edinburgh Multi-Access System (EMAS), and in particular the user interface to the System - the Standard Subsystem. The first five chapters describe those parts of the whole System that support the Subsystem; the remaining chapters describe the Subsystem itself and the way in which it can be used. Appendices to the manual include a list of standard error messages, character code translation tables and a glossary of words and abbreviations used in EMAS user documentation.

It has been the policy of those responsible for running the System that its appearance should be developed to reflect the needs of its users. This process is continuing and as a result this manual will become out of date. It is unlikely that any future changes will invalidate information in the manual, but new facilities will be added and extensions made to the existing ones. The EMAS HELP command provides an up-to-date description of the facilities currently available.

One of the characteristics of EMAS is the ease with which users can add their own features to the standard facilities. Further they can readily make their additions available to their colleagues and to the wider user community. Apart from the facilities described in this manual there is a large amount of user-contributed material including commands, routines and packages. Information about what material is available can be sought from the Advisory Service, in the first instance.

The manual was typed by Mrs Anne Tweeddale and printed by the ERCC Reprographics Department. It is dedicated to all the members of the EMAS user community who by their suggestions, formal or otherwise, have created the present appearance of the System.

Roderick McLeod
December 1976

Preface to First Update

During the period since the publication of this manual a number of changes have taken place. The main one is the addition of partitioned files, which are described in appendix 4.

Roderick McLeod
January 1978

System processors

Apart from virtual processors occupied by users there are a number of system functions that run in their own virtual processors. In many ways they are similar to user processors but they have some special attributes:

* they are privileged - that is, they can access information and obtain services from the resident supervisor which are not available to user processors

* they normally run without any interactive terminal

The two most important system processors are the demons processor and the volumes processor. The demons processor is responsible for

* handling input files from slow devices, e.g. card readers (see chapter 4)

* handling the output of files to line printers etc. (see chapter 4)

* scheduling batch jobs (see chapter 19)

* the verification of names and passwords at log on (see chapter 5)

* controlling communications to remote terminals (see chapter 4)

The volumes processor is mainly concerned with the organisation of the archive store (see chapter 3).

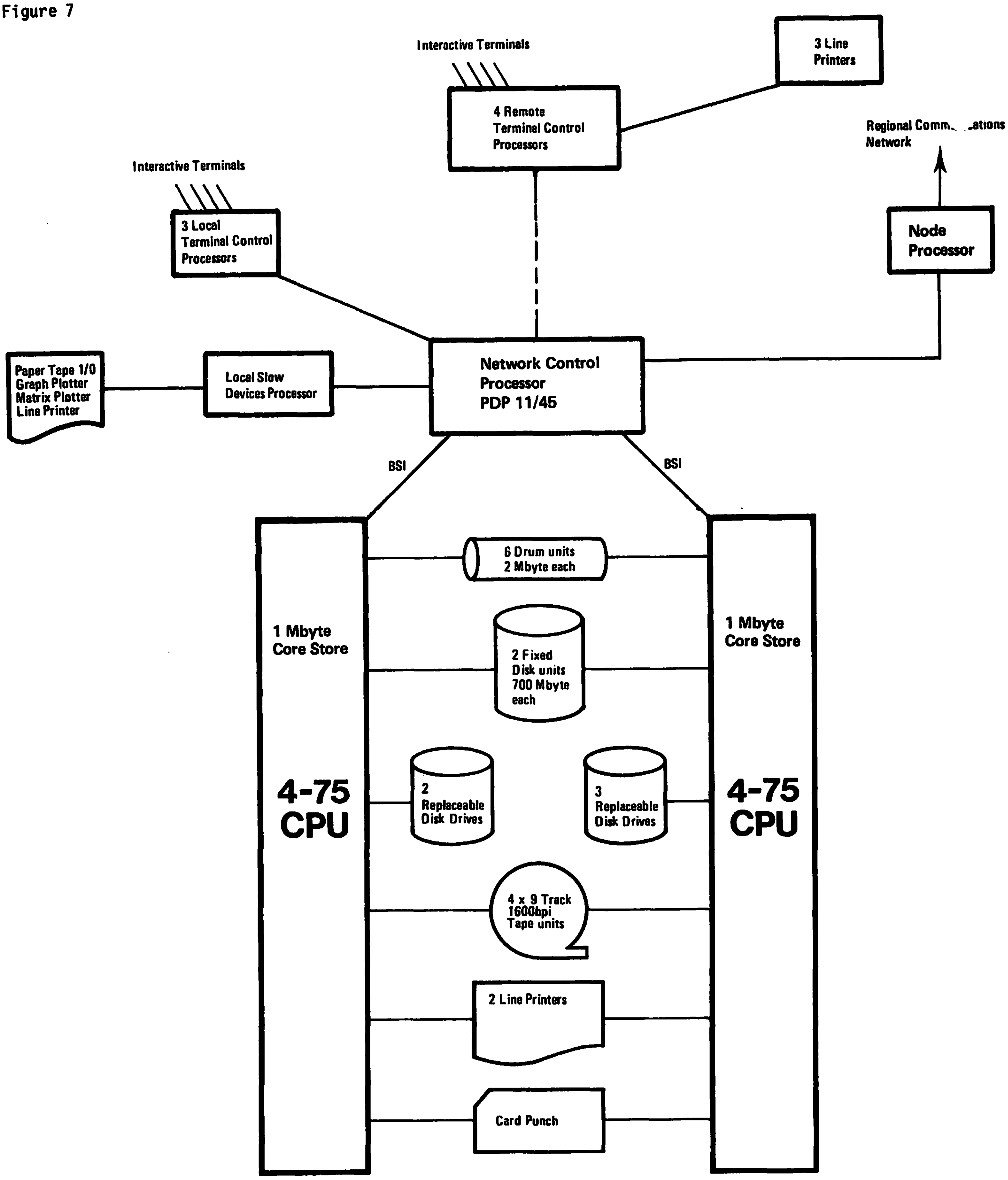
Scheduler

Part of the supervisor, known as the scheduler, is concerned with sharing available main store and central processor time between the various processors competing for these resources. Since EMAS is primarily intended as a multi-access system for interactive computing, the scheduler is designed to give priority to processes which make comparatively small demands for store and processor time. Put another way, the scheduler gives a lower priority to jobs which require large amounts of either resource. Thus processes which display characteristics of batch jobs, or are in fact started as batch jobs, only get resources when other processes are not waiting for them; see also reference 4.

HARDWARE CONFIGURATION

This manual does not contain a detailed description of the hardware configuration, or of any part of the hardware. The overall structure is summarised in figure 7.

Figure 7



Interactive Terminals
3 Line Printers
4 Remote Terminal Control Processors
Regional Comm... ...ations Network
Interactive Terminals
3 Local Terminal Control Processors
Node Processor
Paper Tape I/O Graph Plotter Matrix Plotter Line Printer
Local Slow Devices Processor
Network Control Processor PDP 11/45
BSI
BSI
6 Drum units 2 Mbyte each
1 Mbyte Core Store
2 Fixed Disk units 700 Mbyte each
1 Mbyte Core Store
2 Replaceable Disk Drives
3 Replaceable Disk Drives
4-75 CPU
4-75 CPU
4 x 9 Track 1600bpi Tape units
2 Line Printers
Card Punch

Figure 7 Simplified Diagram of EMAS Hardware

CHAPTER 3
THE FILE SYSTEM

This chapter describes the EMAS file system. Files are used in EMAS for a wide variety of purposes. They can contain programs and data as in conventional systems but also they are used to hold temporary information such as the variables and arrays for a running program. The file system can be divided into two parts: immediate files, which are held on the discfile and are available for use whenever the user logs on to the system; and archive files, which are held on magnetic tape and which have to be transferred to immediate storage before they can be used.

All files are made up of one or more pages of information, the size of each page being 4096 bytes. As explained in chapter 1, when they are being accessed these pages are mapped onto pages in the user's virtual memory, and access to any particular part of the file is then achieved by addressing the appropriate part of his virtual memory. One way in which EMAS differs from some other systems is that it frees the user from any concern about the physical layout of his file. The system controls the way in which a file is stored on disc or on tape.

IMMEDIATE FILE STORE

The immediate file store is organised by the director. The director is only concerned with files as sequences of pages; it is not concerned with the internal structure or contents of files. The effect of this is that a number of Subsystem functions - those which are passed straight on to the director - are not concerned with file types; for example those functions initiated by the commands DESTROY and RENAME.

Naming files

Each file in immediate storage has a unique name which comprises two parts - the name of its owner (the 'ownername') and a file name given by the owner to distinguish it from his other files (the 'filename'). The two fields are separated by a full stop. Files created by users should have filenames of between 1 and 8 characters which should all be upper case letters or numerals, the first character being a letter. Examples of valid full filenames are:

```
ERCC06.FILEABC1
ERCC99.N
LMPT01.FIRE
```

The filenames of files created by the Subsystem usually start with the characters 'SS#' to avoid conflict with files created by the user. For example:

```
ERCC06.SS#STK
ERCC99.SS#LIST
```

Note that when referring to his own files a user does not normally have to prefix the filename with his ownername. Thus user LMPT01 would refer to the file in the example above as FIRE. When referring to files belonging to other users, however, he must always use their full names.

Security of files

The information contained in a file can only be read or altered if the file is connected in the user's virtual memory. The Subsystem can only cause a file to be connected by making a call on the director, and the director will only connect a file if appropriate access permission to the file has been given by its owner (see below). There is thus no way in which a fault in the Subsystem or in a user's program can enable him to access or corrupt information not intended for him. This arrangement makes it possible to store confidential information in EMAS without risk of corruption or illegal access.

Access permission

Each file has associated with it access permission information. Four different types of access are possible:

READ (unshared)
WRITE (unshared)
READ SHARED
WRITE SHARED

Any or all or none of these modes can be given to the file in respect of

* the owner
* all other users
* a specific user
* a group of users

When created, a file has all modes of access permitted to its owner and none to anyone else. Thus, unless a user explicitly permits a file to someone else, it is only available to himself.

The Subsystem command PERMITFILE is used to set access permission information for a file. Its use is described in chapter 7. When determining what access permission is allowed to a user who wishes to connect a file, the director follows these rules:

* If connecting a file belonging to self then use the access permission explicitly for self.

* If connecting a file belonging to another user then use the first of the following which applies:

  1. If specific permission has been granted to this user then use it.
  2. If group permission has been granted to a group that contains this user then use it.
  3. Use the 'everybody else' permission.

Note that if this user is included in more than one group of users the resulting permission is undefined.

Connect modes

A file can only be connected in a particular mode in a user's virtual memory if all of the following conditions are satisfied:

* It is permitted to the user in the required mode.

* It is not connected in another virtual memory in a conflicting mode.

* If the connect mode is WRITE or WRITE SHARED it is on the same machine as the requesting user (see chapter 2).

The following connect modes can be used:

* READ (unshared) - the file can only be connected in this mode if it is not connected in any other virtual memory. Whilst so connected it cannot be connected in any other virtual memory, even if permitted. Whilst connected in this mode it cannot be written to.

* WRITE (unshared) - all the attributes of READ (unshared) apply except that it can be written to or modified.

* READ SHARED - the file can be connected in a virtual memory in this mode if it is connected in one or more other virtual memories in the same mode or if it is not connected in any other virtual memory. Whilst so connected it can be connected in further virtual memories but only in READ SHARED mode. It cannot be written to or modified.

Transfer of ownership of a file

The Subsystem commands OFFER and ACCEPT result in a file being transferred from one user to another. The information relating to the name, the size, the location on the discfile and the CHERISH status is transferred from one file index to another. Note that all other information normally held in the file index (usage information and access permissions) is not transferred. Note also that the OFFER/ACCEPT mechanism can be used between users on the same or different machines.

Back-up of immediate file store

In order to protect users' files from hardware and system faults, they are copied onto magnetic tape once each day ('backed up'), subject to the following rules:

* A file must be marked for this purpose by use of the CHERISH command. Note that files created from card or paper tape input are automatically CHERISHed but otherwise files are created without the CHERISH marker being set. Note also that un-CHERISHed files will be DESTROYed by the system after a period (currently 4 weeks) of non-use.

* A file will only be backed up if it has been altered since the last time it was backed up. (Strictly, it is sufficient to have connected the file in WRITE or WRITE SHARED mode to cause back-up to take place.)

If, because of a failure, it is necessary to recover users' files from back-up tapes, the files are replaced on the discfile with the CHERISH status and any access permissions in force at the time they were backed up. Note the following points:

* Any alterations made to the file between the time of the latest back-up and the system failure will be lost.

* Files which have recently been DESTROYed may be copied back to the immediate file store. It will thus be necessary for the user to DESTROY them again.

Automatic re-prime

If a user's file is corrupted as a result of a system failure or hardware fault, it is deleted automatically at the next IPL. At the same time a message is sent to the user and it will be printed when he next logs on for a foreground session, e.g.

```
**TESTPROG DELETED
```

If the file was CHERISHed an automatic re-prime is initiated to transfer the latest copy in the back-up store to immediate store. If this succeeds a message of the following form is printed:

```
**TESTPROG REPRIMED
```

THE ARCHIVE FILE STORE

The Archive File Store is held on magnetic tape, quite separately from the back-up tapes. Files are moved from the immediate store to archive store in the following circumstances:

* as a direct consequence of using the command ARCHIVE in respect of the file

* indirectly as a result of CHERISHing a file but not accessing it for a period (currently 4 weeks)

The command ARCHIVE and related commands for restoring archived files to immediate store are described in chapter 7. Unused files are moved onto archive storage in order to free the limited space on immediate store for material which is being actively used.

There is currently no limit to the number of files a user can have on archive store, nor to the length of time they are left there. On the other hand users are encouraged to tidy up their archive file list periodically.

Unlike files on immediate store it is possible to have two files on archive with the same name. In such cases the date of archiving is used to indicate which file is to be restored.

CHAPTER 4
INPUT AND OUTPUT OF FILES

The Input and Output of files to and from EMAS is controlled by the DEMONS systems process (see chapter 2). It is not necessary for users to access directly file input and output devices, e.g. card readers or line printers. Instead they instruct the DEMONS process to carry out operations on these devices with reference to particular files.

INPUT

It is possible to read files into immediate storage from cards or paper tape. This is usually achieved by submitting the file, with appropriate Job Control Language (JCL) statements, to be read on a card reader or paper tape reader attached to EMAS via the NCP (see chapter 2). Alternatively, input can be sent from terminals connected to the medium speed network; files can also be transferred from other processors connected to the network. The details of these operations are subject to changes as the network is developed; the user is referred to the current HELP information.

Card reader input

Cards can only be read in a mode involving translation from IBM 029 card code to ISO internal code. The file produced is a standard EMAS character file (see chapter 11). By default the following rules apply:

* All 80 columns are read.
* Translation is carried out according to the table in Appendix 2.
* Trailing spaces are deleted; that is, all the spaces following the last non-space character on a card are excluded from the file.
* A NEWLINE character (ISO 10) is inserted in the file to signify the end of each card.

As explained below, some of these actions can be altered by including appropriate keywords on one of the JCL cards.

Paper tape input

Paper tape can be read in one of two modes:

* normal mode, for reading ISO-coded 8-track even-parity tape
* binary mode, for reading any punchings on 8-track tape

When reading in normal mode the following rules apply:

* all characters with odd parity are converted to the SUB (ISO 26) character
* the parity bit is set to 0 on input, whatever its value on the paper tape
* null (no holes punched) and delete (all holes punched) are ignored
* CARRIAGE RETURN (ISO 13) characters are ignored when they are adjacent to NEWLINE (ISO 10) characters

* trailing spaces (SPACE characters immediately before a NEWLINE character) are ignored

As explained below some of these actions can be altered by including keywords in one of the JCL statements.

When reading in binary mode all characters, including run-out (0), are read into the file, which is created as a standard data file (chapter 11) with a fixed record length of 80 bytes.

Job control for file input

The following JCL statements are needed for file input:

```
    //username   FILE   (PASS=back)
    //filename   DD   *
        <contents of file>
    //
```

where	username	is the name of an accredited EMAS user
	back	is the user's background password (see chapter 20)
	filename	is the name to be given to the file (see chapter 3)

Notes

* The format of the statements is fixed. Each statement must start on a new line; this means that for paper tape input there must be a line feed character before the first statement and between statements. One or more spaces must be inserted where indicated, but nowhere else.

* The name chosen for a file should not be the same as the name of an existing file. If it is, the existing file will be left and the one being read in will be ignored.

* Files are CHERISHed automatically on input.

* More than one file can be input, the form then being:

```
    //username   FILE   (PASS=back)
    //file1   DD   *
        <contents of file1>
    //file2   DD   *
        <contents of file2>
        .
        .
        etc.
        .
    //
```

* When reading in normal mode the following options can be used. The options selected should follow the asterisk on the DD card, separated from it and each other by commas:

    * NOIDENT - ignore the contents of columns 73-80: this is useful for removing sequence numbers

    * TRAIL - do not delete trailing spaces

    Example

    ```
    //TEXT DD *,NOIDENT,TRAIL
    ```

When reading paper tape in binary mode, the following form is used:

```
    //username   FILE   (PASS=Back)
    //filename   DD   BINARY
        contents of file
```

Job control statements and binary data should be on separate paper tapes and must be clearly marked as to the required mode of input. The operator will supply an 'end of file' making a terminator tape unnecessary.

OUTPUT

Information can be sent to output devices by two methods:

* By an explicit Subsystem command such as LIST or SEND (see chapter 8).

* Indirectly by using the DEFINE command to link a logical output channel to a particular device. The effect of this is to put the output in a temporary file which is sent to the output queue automatically when the file is closed (see chapter 11).

In neither case is the device accessed directly by the user. His output is held in an output queue until the required device is available; it is then listed. This may take place minutes or even hours after the user has requested the action. The command QUEUES is available to tell the user whether any files of his are waiting in output queues (see chapter 21).

Output device mnemonics

Output devices are referred to by mnemonics, for example .LP for the line printer. The dot is used to distinguish the device from a file name, since in many situations a device mnemonic <u>or</u> a file name can be used for a particular parameter in a command. If the output device is not connected directly to EMAS then its mnemonic is followed by the number of the remote terminal to which it is connected; e.g. .LP15 is the line printer connected to terminal 15. Note that since the terminal numbers are liable to change a list is not included in this manual. It can be found in the current HELP information.

Table of output devices

The following table gives the names and mnemonics of available output devices, with file types and record formats required (where relevant). File types and record formats are described in chapter 11. The final column gives the maximum size of file that can be sent to the device, in pages (each of 4096 bytes).

Device	Mnemonic	File type	Record format required	Max file size (in pages)
Line Printer	.LP	Character or Data		255
Line Printer with upper and lower case	.LLP	Character or Data		63
Money Line Printer	.MLP	Character or Data		255
Card Punch	.CP	Character or Data		63
Paper Tape Punch	.PP	Character or Data		63
Binary Paper Tape Punch	.BPP	Data	F80	63
Graph Plotter	.GP	Data	F80	63
Graph Plotter for liquid ink jobs	.SGP	Data	F80	63
Matrix Plotter	.MP	Character or Data		255

Notes

* For remote devices there is a limit of 79 pages on the size of output files. It should be noted that because of limitations in the communications mechanisms binary files are expanded before being sent, so the effective limit is half of this.

* The record format is described fully in chapter 11. In this context it is the format that should be used when writing files to be sent to the specified device. Additionally it is the record format implied by default when using DEFINE for these devices. For example, if the command

  ```
  DEFINE(SQ27,.PP)
  ```

  is used then each record written on sequential file 27 must be 80 bytes long.

* The significance of '.MLP' is explained below.

CHARACTERISTICS OF INDIVIDUAL OUTPUT DEVICES

Line Printer

* If the device is defined as .LP or .MLP then lower case letters are converted to upper case if the device does not print lower case letters.

* Lines longer than 132 characters are split and continued on the following line.

* CR (carriage return) is ignored if it is adjacent to line feed. It can be used to achieve over-printing if it appears within text. Note however that not all line printers accessible from EMAS are able to do this.

* If the device is defined as .MLP then the ISO character 33 is printed as pound sterling £ , instead of hash #.

Card Punch

* If the device is defined as .CP then a translation is performed from internal code to IBM 029 card code (see Appendix 2).

* If the device is defined as .CP then any lines longer than 80 characters are split and continued on the next card.

Paper Tape Punch

* When the punch is defined as .PP, paper tape is punched using the characters sent, made up to even parity where necessary by punching in the 8th hole. Characters with values greater than 127 are converted to the SUB character (ISO 26).

* When the punch is defined as .BPP all eight bits of each byte in the file are punched on the tape.

Graph Plotter

The graph plotter should be accessed via the graphics routines provided; these are described fully in reference 8. The device .SGP is used to indicate that 'special' facilities are required, such as liquid ink.

Matrix Plotter

The software for operating this device is currently being written. Further information should be obtained from the Advisory Service.

Reply	Effect
U	Upper Mode - All lower case letters are translated on input by the TCP to upper case. (default)
L	Lower Mode - No lower case translation is carried out on the input.
G	Graphic Mode - No lower case translation is carried out, and, with respect to terminal output, all format controls are disabled, thus allowing any character [values in the range 0-255] to be sent to the terminal.
Pn	Pads - n should be a number in the range 0-9. Used to specify number of pad characters to be inserted at start of each output line. Normally 0. It is only needed for a few special terminals which require a delay to allow the carriage to return to the beginning of the line.
N	Normal carriage - insert newline if attempt is made to output more than 72 characters on a line - (default).
W	Wide carriage - insert newline only if an attempt is made to output more than 132 characters on a line.

Type ahead

An important characteristic of the terminal support mechanism on EMAS is the ability for a user to type ahead. This means that it is not necessary to await the completion of one operation before typing the next command. There are a number of points to bear in mind:

* When typing ahead it must be appreciated that mistakes in typing earlier commands can have disastrous results later on. It is suggested that until users are conversant with the system they await the outcome of each command before typing the next.

* Because the prompt is not printed (see below) and because output and input will be interleaved, it is not always easy to decipher a listing produced on a hard copy device, such as a teletype, when extensive type-ahead is used. The RECALL facility (see chapter 12) avoids this problem.

Lost Input

When the system is heavily loaded it is occasionally unable to accept all input that has been typed ahead. If this occurs the message

```
**INPUT LOST
```

is typed on the interactive terminal. The user should wait until all the available input has been read and a prompt appears to determine how many lines of input have been lost.

Prompt mechanism

Whenever input is requested by the system or by a running program a prompt is output on the interactive terminal. This acts as a reminder to the user that input is required, and can also serve to indicate what sort of input is required. The prompt text is set by

* the Subsystem

* a Subsystem facility, e.g. EDIT

* a user program calling the routine PROMPT or FPRMPT (see chapters 15 and 16)

Note that the prompt is not output if the required input has already been typed ahead.

Logging in

Before a user can log on to the system from an interactive terminal he has to obtain an accredited username (see chapter 20). The username has associated with it two passwords (see chapter 20). The first of these is used for interactive access. In order to log on the following steps should be taken:

* Switch on the terminal and select Duplex mode, and, if using a dial-up line, also the modem or acoustic coupler.

* If using a dial-up line dial the correct number (031-667 1071) and if a data tone is heard (high pitched tone) switch to data. (The exact method depends on the terminal and MODEM or coupler being used.)

* Press the space bar. If there is no response press the CR key.

* To the prompt 'HOST:' reply EMAS.

* To the prompt 'USER:' reply with the username.

* To the prompt 'PASS:' reply with the foreground password.

All three replies should be terminated by CR.

In response the TCP and NCP will attempt to log the terminal on to the appropriate EMAS main computer. If this is successful a 'PROCESS STARTED' message will be printed, sometimes followed by a message of the day, and eventually a prompt COMMAND: will appear.

Alternatively one of the following messages, or some other explanatory message, will be printed:

SYSTEM FULL	- try later (or use restricted access - see below).
CANNOT START PROCESS	- possibly because the process is just stopping - try again, and if the problem persists contact the Advisory Service.
PROCESS RUNNING	- a background job is running which you DETACHed earlier (see chapter 19), or another person who shares the username is currently logged on. Disconnect the terminal by typing CNTRL+A followed by CNTRL+D and then try later.
NO USER SERVICE	- this indicates an attempt to log on outwith the service period, or a machine fault. Ring the answering service (031-667 7491) for information.
INVALID USER	- this may be caused by mis-typing the username or because one of the EMAS computers is unavailable (see above).
INVALID PASSWORD	- this means that the password was typed incorrectly.

Restricted access mode

When the reply SYSTEM FULL is printed it may still be possible to log on in restricted access mode. The method is to log in as above, except that the reply to the prompt 'HOST:' should be 'EMAS-L'. If there is a slot available the user will be logged on and can use the system subject to the following restrictions:-

* The default and maximum values for CPULIMIT are each 5 seconds (as against 30 seconds and something between 30 seconds and 10 minutes respectively). See also chapter 21.

* A strict rate check is applied so that any attempt to run any significantly cpu-bound job will result in a "RATE EXCEEDED" message, and subsequently an automatic log-off (see chapter 21).

Despite these limitations restricted access still allows for the use of many commands such as EDIT, DETACH, FILES, RESTORE etc.

CHAPTER 6
INTRODUCTION TO THE SUBSYSTEM

This chapter provides an introduction to the standard EMAS Subsystem. It explains the Subsystem's command language, describes its interactive terminal interrupts, introduces the file types provided, gives sources of further information, summarises its functions in logical groups and indicates how these are covered by the rest of this manual.

The standard Subsystem

The phrase 'standard Subsystem' is used to emphasize that the Subsystem described is the one provided as a standard part of EMAS. It is not, however, the only Subsystem and it should be appreciated that it is possible to use subsystems which differ slightly or even fundamentally from the standard one, without interfering with other users, and without having to make changes to other components of the system. Chapter 18 shows how it is possible to add commands to the standard Subsystem. More fundamental changes require information outwith the scope of this manual.

The Subsystem command language

The Subsystem command language is used to communicate with the Subsystem, both from interactive terminals and from background jobs (see chapter 19). Commands are typed according to the following rules:

- Each command must start on a new line.
- If the command requires one or more parameters, these should be typed after the command, normally enclosed in parentheses.
- As an alternative form of input it is possible to omit the parentheses, in which case the command name must be typed with no embedded spaces, and must be separated from the first parameter by one or more spaces. See OPTION (chapter 21). For clarity all examples in this manual use parentheses.
- Parameters should be separated by commas.
- To indicate that a parameter has been omitted an extra comma should be inserted, if more parameters follow.
- Spaces and newlines within parameters are ignored.
- After removing spaces, newlines and parentheses the total length of the parameters should be not greater than 63 characters.

Examples of commands

```
ALERT
LIST(ABC)
LIST(ABC,.LP)
FILES(,IA)
DESTROY(ABC,DEF,GHI)
```

In the rest of this manual the following format is used for commands:

```
COMMANDNAME(PARAMETER1[,PARAMETER2])
```

Note that the square brackets are used to indicate optional parameters - they are not typed when the command is used.

The full details of the parameters required for each command are given with the description of the command. There are, however, a number of common features:

* List - this is used for commands that can operate on a number of items of the same type. The list can consist of one item, or more than one, in which case all but the last are followed by a comma. For example, DESTROY can be used with a list of filenames:

  ```
  DESTROY(NABC)
  DESTROY(NABC,TYPE,FILE27)
  ```

* Output device - this is used for commands that generate output. In almost every case the default output device is the interactive terminal. Output devices are described in more detail in chapter 4. The name of a device consists of a full stop followed by a mnemonic; for example, .LP means line printer:

  ```
  LIST(TEST27,.LP)
  ```

* Output file - this is often an alternative to an output device. It is sometimes useful to direct the output from a command into a file, for subsequent examination using the editor or a user program. For example, the command FILEANAL can be used to obtain information about a file. If a second parameter, not an output device, is given then the information is put in the file named by the second parameter:

  ```
  FILEANAL(FILEABN,OUT)
  ```

  Other parameters are described in the context of particular commands.

Messages output by the Subsystem

In general, simple commands do not produce any output if they work successfully. A few, indicated in the table at the end of this chapter, produce confirmatory messages. Even these may be suppressed if preferred by use of the OPTION facility - see chapter 21. All commands produce failure messages if they do not work correctly. Subsystem failure messages are described in two places:

* Messages specific to a particular command are described with that command

* General error messages are described in Appendix 1

Operator messages

Apart from messages generated by the Subsystem there are messages sent to interactive terminals by the EMAS operators, or on their behalf. These messages are of the form:

```
**OPER hh.mm message
```

It is hoped that these messages will be self explanatory, but since they are restricted to 19 characters in length they are likely to be somewhat terse.

CONSOLE INTERRUPTS

Apart from normal interactive terminal input and output there is a mechanism whereby any operation can be interrupted. The method, described in chapter 5, allows the user to input a message of up to 15 characters. Single character input messages are used to control the Subsystem in the following manner:

Interrupt	Effect
A	Abort current command or program and return to read the next command.
C	As for A except that additionally any input that has been typed ahead is lost.
H	Use during printing of diagnostics to return to calling program. This is only of use when diagnostics have been printed as a result of a call of %MONITOR from IMP or DIAG in FORTRAN.
M	Make disc consistent; i.e. ensure that all copies of files in immediate store include all changes made to date. It can be used at any time without otherwise affecting the command that is currently being obeyed.
Q	Abort current command, print diagnostics and return to command level.
T	Print out the time and number of page turns since the start of this command and the number of users logged on, without affecting command being executed.

SUBSYSTEM FILE TYPES

| There are six types of file recognised by the Subsystem. The type of a file is determined from information held at the beginning of the first page of the file. The file types are listed below with an indication of the main chapter describing their use.

Type	Use	Chapter
CHARACTER	Contains characters - e.g. program source.	11
DATA	Contains binary data in discrete records.	11
OBJECT	Contains compiled programs and routines.	9
LIBRARY INDEX	Contains information to associate routine entry names with OBJECT files - used by the loader.	10
STORE MAP	Unstructured file used for direct mapping of data.	13
PARTITIONED	Contains members, each of which is for some purposes equivalent to a complete file.	Appendix 4

SUBSYSTEM INFORMATION

Apart from this manual the primary sources of information about the Subsystem are:

* the commands HELP and ALERT

* the EMAS Information Card

* the ERCC Advisory Service

The command HELP

This command provides on-line information for EMAS users. If the command is typed with no parameter then the output is a list of current commands and a brief description of their purposes. If a parameter is given which is the name of one of these commands then fuller information about the chosen command is typed. For example

```
HELP(FILEANAL)
```

would give information about the command FILEANAL.

Apart from commands, there are a number of general headings about which information is available. Currently these are:

```
ADVISORY
GRAPHICS
INTERRUPT
LIBRARIES
NEWS
OPERATIONS
PACKAGES
REMOTES
SCHEDULE
TERMINAL
```

Finally, HELP can be used with a parameter '.LP', '.LLPnn' or '.LPnn' (see chapter 4), in which case the whole of the current HELP text is printed on the local line printer or remote line printer specified.

The command ALERT

This command provides information about recent changes to the system and any serious faults that have been reported or corrected. If the command is typed with no parameter then the output is given on the interactive terminal. Otherwise it can be directed to a local or remote line printer; for example:

```
ALERT
ALERT(.LP)
```

EMAS Information Card

This quick reference information card provides a list of the currently available commands and their parameters and also details of remote terminals. It is intended to reprint it at least once each year, so it should reflect changes more quickly than this manual. Copies are available from the ERCC Library.

ERCC Advisory Service

The Advisory Service is available to users of EMAS, and Advisors will endeavour to answer questions about the Subsystem and the main programming languages.

Full details of the Advisory Service are contained in the current edition of the ERCC Advisory Guide.

Subsystem facilities

The facilities and commands provided by the Subsystem are divided in this manual into the following groups:

* General File Utility commands - these commands operate in respect of files as units. They operate on any type of file.
* Type Specific File Utility commands - these are used to carry out functions such as copying and listing files.
* Compilers and associated commands.
* Commands associated with program loading.
* Commands related to manipulating user data.
* File editing commands.
* Commands concerned with running work in background mode.
* Commands concerned with accounts and usage.
* Information and other commands that do not conveniently fit into other categories.

The table following gives a list of the commands in each group and for each the following information:

* A brief description of the purpose of the command.
* Whether the command produces any output (other than a failure message).
* A page number in this manual of the main description of the command.

TABLE OF COMMANDS

Group	Command	Purpose	Output	Page
General File Utilities	ACCEPT	Transfer file from another user		37
	ARCHIVE	Mark file(s) for transfer to archive store		40
	CHERISH	Mark file(s) for backing-up		39
	DESTROY	Destroy file(s) in immediate store		36
	DISCARD	Destroy file(s) in archive store		41
	DISCONNECT	Remove file from virtual memory		36
	FILES	Obtain complete or partial list of files in immediate and archive stores.		35
	HAZARD	Un-CHERISH file		39
	OFFER	Mark file for transfer to another user		37
	PERMITFILE	Allow other users access to a file		37
	RENAME	Change the name of a file		36
	RESTORE	Copy a file from archive to immediate store	*	40
Type Specific File Utilities	CONCAT	Join two or more character or data file(s)	*	46
	COPYFILE	Copy a file	*	46
	FILEANAL	Obtain details of type, contents and access permission of a file	*	45
	LIBANAL	Obtain details of contents of library index file	*	59
	LIST	List file on output device	*	47
	NEWPDFILE	Create new, empty, partitioned file		137
	SEND	List file on output device and destroy it	*	48
Compilers and associated commands	ALGOL	Compile ALGOL 60 source file	*	107
	FORTE	Compile FORTRAN IV source file	*	99
	IMP	Compile IMP source file	*	89
	LINK	Join two or more OBJECT files	*	52
	PARM	Set compiler options		51
Manipulating Data	CLEAR	Break link set up by DEFINE or DEFINEMT		67
	DDLIST	Print list of current logical channel definitions		68

Group	Command	Purpose	Output	Page
Manipulating Data contd.	DEFINE	Set up link between logical channel and particular file or output device		63
	DEFINEMT	Set up link between logical channel and user magnetic tape file		86
	NEWSMFILE	Create new file to be accessed via store mapping facilities		79
File Editing	EDIT	Edit character file	*	69
	LOOK	Examine contents of character file	*	76
Program Loading and Execution	APPENDLIB	Nominate additional library index for searching during program loading		56
	INSERTFILE	Insert details of object file in current library index		55
	PERMITLIB	Allow access to a library index and inserted files		59
	REMOVEFILE	Remove reference to object file from current library index		56
	REMOVELIB	Remove library index from current search list		56
	RUN	Execute program		53
	USERLIB	Nominate new library index	*	55
Background Mode	DELETEJOB	Remove job from background job queue	*	117
	DETACH	Put job into background job queue	*	115
	FINDJOB	Find information about jobs in background job queue	*	117
Commands associated with accounting	METER	Print usage information for current session	*	121
	PASSWORD	Change foreground and/or background password		119
	PROJECT	Set project code		120
	USERS	Print number of currently active users	*	121
Information and other commands	ALERT	Obtain information on state of System	*	30
	CPULIMIT	Set time limit for each command	*	123
	DELIVER	Set text for heading of line printer output, etc.	*	124
	HELP	Get advice on using Subsystem	*	30
	OBEYFILE	Execute a sequence of commands	*	124

Group	Command	Purpose	Output	Page
Information and other commands contd.	OPTION	Set user options		125
	RECALL	Examine file containing record of interactive terminal I/O	*	76
	QUEUES	Print information about files waiting in output queues	*	126
	STOP	Terminate foreground session	*	126
	SUGGESTION	Send suggestion to System Manager		126

CHAPTER 7
GENERAL FILE UTILITY COMMANDS

In chapter 3 the concept of a file was introduced and the conventions relating to EMAS files were described.

As explained there, the basic file handling facilities are provided by the director. These facilities act on files as sequences of pages, the contents of which are not significant. Thus, for example, to the director there is no distinction between a file containing character data and a compiled object file. This chapter describes the file manipulation commands provided by the Subsystem which make calls on the director, and which act on all types of file. The different types of file provided by the EMAS Subsystem are described in the chapters following.

The command FILES

FILES takes up to three parameters: FILES([mask][,group][,out])

mask - Each filename in the group specified in the second parameter is compared with the mask and only those filenames which match are listed. The mask consists of up to three fields, where a field is either a set of explicit characters, or the symbol '*' representing any set of characters, e.g.

ABC	selects file ABC (one field)	
ABC*	selects all files beginning with ABC	(two fields)
ABC	selects all files containing ABC	(three fields)
*XY	selects all files ending with XY	(two fields)

If mask is omitted, all filenames in the group are selected.

group - The group of files is determined as follows:

I	files in the immediate file store (excluding SS# files); the default
C	CHERISHed files in the immediate file store
H	HAZARDed (uncherished) files in the immediate file store
A	ARCHIVEd files; this code may be combined with any of the above.

Two further codes are available which may be combined with the above to modify the printing of the names of files in the immediate store:

S	single spacing (print file names down the page); print across the page is the default
E	print extra information; causes a header to be printed and SS# files to be included. If both E and S are coded then additional information is given on each file selected.

Any combination of the letters given above can be specified, in any order. File names are preceded by '*' if the file is CHERISHed and by '**' if it has been nominated for archiving.

out - This may be either output device code (.TT by default) or output file name. If a file of the same name already exists it will be overwritten.

CREATING, RENAMING AND DESTROYING FILES

Files can be created, renamed or destroyed only by their owners. There is no general command used to create a file. Files are created as a result of using certain commands or facilities. For example the command EDIT can be used to create a new character file. If this file is compiled using the command IMP an object file may be created. If the output from FILEANAL is directed to a file and a file of that name does not exist, one will be created. In general the following rule applies in relation to commands that create files:

If a file of the requested name exists already, it is overwritten - destroying any information it currently contains. If not, then a new file is created with the requested name.

The command RENAME

A file can be renamed using the command RENAME. This takes two parameters:

```
          RENAME(oldname,newname)
```

oldname is the current name of the file
newname is the name to be given to the file

RENAME will fail if a file with the name 'newname' already exists, or if the file being renamed does not exist or is connected in another user's virtual memory or is on OFFER (see below). Note that access permissions and the cherish status of the file are not affected by renaming.

The command DESTROY

One or more files can be destroyed by a call of DESTROY. It takes the name of one or more files as its parameter(s):

```
      DESTROY(ABC)
      DESTROY(TEMP,COBJ,BACL3)
```

The command will fail if the file being destroyed is connected in another user's virtual memory or is on OFFER (see below). Also if a file is permitted to its owner with an access permission of 0 (i.e. no access at all) it cannot be destroyed. This fact could be used to protect a file from inadvertent destruction - but see also CHERISH below.

CONNECTING AND DISCONNECTING A FILE

Before any use can be made of the contents of a file, e.g. before a character file can be edited or an object file can be executed, it must be connected in the user's virtual memory. This operation is described in chapter 3. There is no general command for this purpose - connection occurs as a result of the use of a wide variety of commands or facilities. For example, a file is connected when:

* it is analysed by FILEANAL
* it is listed on the interactive terminal using LIST
* it is edited
* it is read from by a FORTRAN program

Normally, once a file has been connected, it remains connected for the rest of the session - i.e. until the user logs off. There are however a number of commands which cause disconnection, and there is also an explicit DISCONNECT command. The following commands disconnect the file on which they are operating if it is connected at the time:

```
      DESTROY
      RENAME
      OFFER
      PERMITFILE
      PERMITLIB
      SEND
      COPYFILE (disconnects the input file if it belongs to another user)
```

The command DISCONNECT

This command can be used to disconnect one or more files from the user's virtual memory. It takes as its parameter the name of one or more files that are currently connected. There are several situations in which it is useful:

* To protect the file. By disconnecting it the user can be sure that its copy on immediate store (the discfile) is up to date (this can also be achieved by INT:M - see chapter 6). Also, since it is no longer connected in the virtual memory it cannot be corrupted by programs being run by this user that might be faulty. In fact this form of corruption is unlikely, since user files are normally left connected in READ mode (and are therefore protected) when they are not currently being used for output.
* To free the file for use by another user. For example, after a user has executed an object file belonging to another user, the file will remain connected in his virtual memory. If the owner attempts to alter the file (by re-compiling it) a failure will occur, because it is not possible to write to a file connected in READ mode in another virtual memory. If the user who has run the program DISCONNECTs the file, the recompilation will then be possible.
* To free space in the virtual memory. Although large (13 Mbytes) the virtual memory can be filled during a session. To free some space the user should use DISCONNECT to disconnect some of the files that are no longer being used.

TRANSFERRING OWNERSHIP OF A FILE

The two commands OFFER and ACCEPT can be used to transfer a file from one user to another. The owner of the file should use the command OFFER, which takes two parameters: the name of the file to be transferred, and the name of the user to whom it is to be transferred.

```
OFFER(ABC,ERCC98)
```

would offer the file 'ABC' to user 'ERCC98'.

Note that once a file is on offer it cannot be connected in any virtual memory, regardless of access permissions.

An OFFER can be revoked, if necessary, by using the command OFFER with only one parameter - the name of the file.

A file can be offered to any user on either machine.

Accepting the file

The user to whom the file is offered can accept it at any time by using the command ACCEPT. This takes as its first parameter the full file name of the file to be accepted. For example, if the user OFFERing the file in the example above was ERCC38 then the user ERCC98 would type

```
ACCEPT(ERCC38.ABC)
```

The effect of this would be to transfer the file 'ABC' from user ERCC38 to user ERCC98, giving it the new name ERCC98.ABC. This command will fail if user ERCC98 already has a file 'ABC'. However this problem can be overcome by typing a second, optional, parameter to ACCEPT, which is the new name to be given to the file. For example,

```
ACCEPT(ERCC38.ABC,NEWABC)
```

In this case the file will be transferred and given the new name ERCC98.NEWABC.

SETTING ACCESS PERMISSIONS ON FILES

In chapter 3 there is a description of the access permission mechanism. This section describes the use of the command PERMITFILE. This command, which can only be used in respect of one's own files, takes three parameters:

```
PERMITFILE(file,user,mode)
```

file	is the name of a file belonging to this user

user is one of the following

null	meaning give access to all other users
a username	meaning give access to a particular user (can be the owner)
a user-group	meaning give access to a group of users. The given parameter may contain up to 5 '?' characters. For example, EGNP?? means give access to any user with a username containing 'EGNP' as its first four characters.

Mode is one of the following modes:

RS or null	READ SHARED
R	READ
WS	WRITE SHARED
W	WRITE
NONE	no access.
CANCEL	used to cancel an access permission given previously to an individual user (other than the owner) or a user group.
ALL	all modes (see below).

In some situations it is necessary to combine more than one mode. This is done by using a single hexadecimal digit for the mode. This consists of a four-bit field, where the bits have the following meanings:

2^0	WRITE
2^1	READ
2^2	WRITE SHARED
2^3	READ SHARED

The table below shows the complete range of possible combinations:

MODE	UNSHARED		SHARED	
	WRITE	READ	WRITE	READ
0 (NONE)				
1 (W)	*			
2 (R)		*		
3	*	*		
4 (WS)			*	
5	*		*	
6		*	*	
7	*	*	*	
8 (RS)				*
9	*			*
A		*		*
B	*	*		*
C			*	*
D	*		*	*
E		*	*	*
F (ALL)	*	*	*	*

Notes

* When a file is created it has default access permissions of all modes to its owner and no access to anyone else.

* There is no overhead associated with access permissions to self and everyone else. Permissions to individuals and groups, however, require space in the file index (see chapter 3).

Examples

```
PERMITFILE(ABC)
```

This permits the file to everyone else with the default access permission READ SHARED.

```
PERMITFILE(DOUBLE,ERCC23,WS)
```

This permits the file DOUBLE to user ERCC23 with WRITE SHARED access permission.

Multiple permissions

It is possible to use PERMITFILE more than once in respect of a file. For example in the following sequence a file is permitted to all users with READ SHARED access permission, but access is withdrawn from users with user numbers starting with 'Y'. Finally access in all modes is granted to ERCC28.

```
PERMITFILE(PERTEST)
PERMITFILE(PERTEST,Y?????,NONE)
PERMITFILE(PERTEST,ERCC28,ALL)
```

The command PERMITLIB (see chapter 10) can be used to control the access permissions of library index files and the object files to which they refer.

Write permissions

The following restrictions should be noted in respect of write permission being given to other users:

* Even if write permission exists it is not possible to connect a file in write mode if its owner is accredited on the 'other' machine (see Chapter 2)

* Write permission does not allow another user to alter the size of a file. This restriction means that it is only possible to write to Direct Access files, or to files connected using SMADDR, if they belong to another user.

COMMANDS RELATED TO BACKUP

As explained in chapter 3 some files are copied onto a back-up store. All files which are likely to be difficult to reconstruct in the event of file system corruption should be marked by use of the CHERISH command. This command can be used to mark one or more files:

```
CHERISH(SNAP)
CHERISH(ABC,MINE,COBJECT)
```

The command HAZARD can be used to remove the CHERISH status.

Notes

* When first created, files are not normally CHERISHed. It is the user's responsibility to CHERISH his important files.

* The CHERISH status of a file also affects its disposal when it is left unused for a significant period - see below.

COMMANDS RELATED TO ARCHIVE STORAGE

The archive store is held on magnetic tape, quite separately from the backup store. It contains files that have been moved there from immediate storage for one of the following reasons:

* Because the owner has indicated that he wishes the file to be moved by using the ARCHIVE command.

* Because the file has not been used for a significant period (currently about four weeks) and it has been moved by the system in order to free space in the immediate store. Note that this only applies to files that are CHERISHed: un-CHERISHed files are destroyed if they remain unused for a significant period (currently about four weeks).

Once in the archive store there is no distinction between files moved in for different reasons.

The command ARCHIVE

This command, which takes one or more filenames as a parameter, is used to mark files which the user wants to move from the immediate store to the archive store. Note that this command does not take effect immediately: there may be a delay of up to a week before the file is moved. There are a number of reasons for using this command:

* to clear space in the file index

* to dispose of files that are not currently required but may be needed at some later date

* to reduce the charge for keeping files on the system (see chapter 20)

Obtaining a list of files in the Archive store

The command FILES described at the beginning of this chapter can be used to obtain a list of some or all of a user's files in the archive store.

Moving files from Archive store to Immediate store

The command RESTORE is used to copy a file from archive store to immediate store. Note that the copy in the archive store is not altered by this command. The command takes three parameters. The first is the name of the file being restored, and the second, which is optional, is the date of archiving. This should be typed exactly as it appears in the FILES output. By default the most recent copy of a file is restored. The date is only needed when an earlier copy is required. The optional third parameter may be used to set access permissions to the file for all other users or alternatively to restore with the permissions set as they were when the file was archived. To set permissions for all other users, use the permission codes defined in PERMITFILE. To restore with the original permissions use the code 'WP'; all the permissions including own permissions are re-established. The default is to restore with SELF ALL and OTHERS NONE set.

Example
```
        RESTORE(KERN27S)
        RESTORE(IMP907A,23/12/75)
        RESTORE(DATA27,01/07/74,WP)
```

RESTORE will fail immediately if:

* A file of the same name already exists in the user's immediate store file index. To avoid this it is necessary to rename the existing copy before restoring the old one.

* The date is typed in incorrect format.

* There is no file in the archive store of the requested name, or, if a date is used,

no file of the requested name is held for that date.

If the RESTORE command is successfully interpreted then a request is sent to the VOLUMES process to carry out the operation. The user can then proceed to give other commands to EMAS. The file should be recovered from the archive store within 15 minutes.

The RESTORE operation can fail when the VOLUMES process attempts to copy the file to the immediate store. This will occur if:

* There is insufficient room in the user's file index.
* There is a file of the same name in the user's file index. This would only occur if the user had created a file of the same name after typing the RESTORE command.

An operator message will be typed on the interactive terminal, indicating the successful restoration of the file or a reason for failure. This message will be typed when the user next logs on if he logs off before the file is restored.

Files restored in this way are un-CHERISHed and have default access permissions unless a third parameter were used.

Destroying files in the Archive store.

The command DISCARD

The command DISCARD is used to delete files in the archive store. The command takes effect immediately: this means that following a call of DISCARD a call of FILES(,A) will confirm the deletion of the specified files.

DISCARD can be used with no parameter to delete small numbers of files. After typing DISCARD the user will be requested to supply the name and date of each file to be deleted. To terminate the command reply '.END'. The following example should make this clear:

```
COMMAND:DISCARD
FILE DATE:ACCW370 27/02/77
FILE DATE:IROUT21X 22/02/76
FILE DATE:.END
COMMAND:
```

Notes

* the date must be typed exactly as printed by FILES
* anything following date on the line will be ignored
* each file is deleted from the archive store as soon as its name has been read and checked.

DISCARD can alternatively take as its parameter the name of a file containing names and dates of files to be deleted. The file should contain the names of files to be deleted in the format specified above, i.e. name and date on each line. Note that FILES(,A,filename) can be used as a convenient method of generating such a file. Using either the mask facility in FILES or the editor it is possible to produce a selective list of archive files to be destroyed. An advantage of this method is that it is possible to check the list to ensure that it contains only files which are really unwanted before calling DISCARD. The following examples should help to explain this:

To destroy all files ending in 'Y' in the archive store:

```
COMMAND:FILES(*Y,A,CFILE)
COMMAND:DISCARD(CFILE)
```

To destroy all copies of 'MASTER' in the archive store:

```
COMMAND:FILES(MASTER,A,CFILE)
COMMAND:DISCARD(CFILE)
```

To destroy all files archived before 1976

```
COMMAND:FILES(,A,CFILE)
COMMAND:EDIT(CFILE)
```

Use the editor to delete the names of all files with dates more recent than 31/12/75

```
EDIT:E
COMMAND:DISCARD(CFILE)
```

Notes

* since the command DISCARD ignores any information following the date there is no need to remove the number of pages from each line.
* when convenient, DISCARD should be called with a control file as parameter; this form has efficiency advantages over the parameterless DISCARD.
* when using DISCARD with a file of file names the names should preferably be ordered with the most recent at the beginning of the file. This is the order produced by the command FILES, and will result in the quickest response to the command DISCARD.

Notes

* The default record format for files is V1024. This can be used for almost all applications. It is rarely necessary to specify this parameter at all.

* Certain output devices impose other formats automatically - see the table in chapter 4.

  If files are written for listing on one of these devices at a later stage, the correct record format and length must be specified. For example, if creating a file for sending at a later stage to the binary paper tape punch, it should be defined as, say

```
    DEFINE(SQ27,PPOUT,,F80)
```

  However if it is going straight to the paper tape punch it is not necessary to use the fourth parameter:

```
    DEFINE(SQ27,.BPP)
```

Summary of DEFINE parameters

Parameter	Position	Default	Contents	Examples
ddname	1	None	I/O type and channel no	STREAM3
file/dev	2	None	filename	ERCC27.HELP15 ABCTE
			device	.CP .GP29
			concatenated file	ABC+TEST37+END
			temporary file	.TEMP
			dummy file	.NULL
size	3	255	file size in Kbytes	500
record format and length	4	V1024	record format code and record length	F80 VA137

The command CLEAR

This command is used to clear one or more file definitions that have been established using DEFINE. It can be used in one of three ways:

* With no parameter, in which case all current definitions are cleared.

* With a list of ddnames, in which case the selected definitions are cleared.

* With a list of group names from the following list, in which case all definitions in the selected groups are cleared:

  'STREAMS', 'SQFILES', 'DAFILES', 'FTFILES', 'SMFILES'.

Examples:

```
CLEAR
CLEAR(ST1,STREAM27,DAFILE42)
CLEAR(SQFILES,DAFILES)
```

Notes

* All definitions are cleared automatically at the end of a foreground session.
* If a DEFINE is used for a logical channel for which there is already a definition, the earlier definition is automatically cleared.

The command DDLIST

This command is used to provide a list of current links between ddnames and files or devices. It can be used without a parameter, in which case output goes to the interactive terminal, or with a parameter to specify an output device or file:

```
DDLIST
DDLIST(.LP)
DDLIST(DDFILE)
```

Typical output from DDLIST:

```
SQFILE01    TESTSQ
DAFILE07    TRYPACK
FT14        .LP
```

Q is used as an exit when for some reason the editing done during a session is not required. The effect is to leave the file or files being edited in the state that they were in before use of the EDIT command. In order to reduce the risk of a user inadvertently pressing Q and losing his editing unintentionally this command does not cause an immediate exit but causes the prompt 'QUIT': to appear, to which the user should reply 'Q' or 'Y' if he really does want to exit. Any other reply will result in the edit session continuing.

Preserving editing done so far

The command W can be used at any time during editing to 'Write' all the editing done so far to the output file. This is a useful way to avoid the risk of losing all one's editing if a system failure occurs before one completes an editing session. Note the following:

* The editing done so far is consolidated into the output file

* If Q is used the output file will be as it was left by the W, not as it was before editing commenced

* The cursor is left at the top of the file *T* by this command.

MORE ADVANCED FACILITIES IN THE EDITOR

The commands described so far provide most of the commonly required functions of a context editor. There are three further facilities described here which may be of interest to some users:

* repeated commands

* moving a section of text within a file

* extracting part of a file and putting it into another file

Command repetition

A single editor command or group of commands can be obeyed repeatedly a specified number of times. The commands are enclosed in parentheses followed by an integer repetition factor. For example, if it is required to remove all occurrences of the text 'REAL' in a file and to replace them with the text 'INTEGER' one could type

```
(R/REAL/I/INTEGER/)1000
```

This assumes that there are not more than 1000 occurrences of the text 'REAL'. Another use of this facility might be to find out the names of the next 10 subroutines in a Fortran program. The command sequence to do this would be:

```
(M/SUBROUTINE/P1M1)10
```

Note that strictly this example would print the next 10 lines that contained the text 'SUBROUTINE'. Since this word might appear in a comment the command sequence might not achieve the required effect. Note also the 'M1' in the command sequence. If this were not included the effect would be to print the next line containing 'SUBROUTINE' ten times. This example illustrates the need to consider carefully the effect of repetitive editing commands.

Bracketed commands may be nested.

The normal rules concerning failures within commands are followed. If a failure occurs the whole sequence of commands is aborted.

The separator *S*

The editor command 'S' is used to set a separator before the present position of the cursor. This separator is used to indicate the destination of text being moved (see below). Additionally it has the effect of a separator in the file. Three commands can be used to move the cursor past the separator:

* T. This moves the cursor to the top of the file, if necessary passing the separator

* B. This moves the cursor to the bottom of the file, if necessary passing the separator

* O (Over). This command, which takes no parameter, moves the cursor to immediately before the separator. This is a more efficient operation than, for example, TM1000, which would be the form needed if the cursor were positioned after the separator.

All other commands which move the cursor can only move it as far as the separator. This fact can be utilised when it is required to search only part of a file for text strings. Before starting, the user positions the separator at the end of the text to be searched. The position of the separator is indicated by the text

```
*S*
```

This text does not actually exist in the file.

The separator can be moved from the file by use of the command 'K'. This command takes no parameter and leaves the cursor at the point previously occupied by the separator.

Moving a section of text within a file

The process of moving part of a file from one place to another within the file involves

* setting a separator at the destination of the file, using the command 'S'

* moving the cursor to the top (start) of the text to be moved

* using the command 'U' to move the text

* removing the separator with the command 'K'

The commands 'S' and 'K' are described above. The command 'U' takes either an integer parameter or a text string parameter. When used with an integer the value of the parameter specifies the number of lines to be moved, counted from the current line. When a text string is used the text moved extends from the current position of the cursor up to and including the first occurrence of the specified text. In the following example the routine B is to be moved to before routine A.

Initial state of file:

```
*T*
%BEGIN
%ROUTINE A
    TEXT OF ROUTINE A
%END
%ROUTINE B
    TEXT OF ROUTINE B
%END
    TEXT OF PROGRAM
%ENDOFPROGRAM
*B*
```

```
EDIT:M/%ROUTINE A/S           Set separator
*S*
↑%ROUTINE A
EDIT:M/%ROUTINE B/U/%END      Move text
/:/
↑   TEXT OF PROGRAM
EDIT:K                        clear separator
↑%ROUTINE A
```

Final state of file:

```
%BEGIN
%ROUTINE B
    TEXT OF ROUTINE B
%END
%ROUTINE A
    TEXT OF ROUTINE A
%END
    TEXT OF PROGRAM
%ENDOFPROGRAM
```

Extracting part of a file

The command 'F' is used to extract part of a file and to put it into another file or send it to an output device. The parameter must be of the form

```
      <filename>
or    <output device>
```

The abbreviations used for output devices are given in chapter 4. The text which is extracted is that which lies between the present position of the cursor and the 'S' separator, if it is positioned lower in the file than the cursor; otherwise, between the cursor and the bottom of the file. The cursor is not moved by this command and the text in the file being edited is not altered. If the parameter specifies a file that already exists, the file will be over-written; otherwise a new file will be created. This command can be used, for example, for extracting one routine from a file for use in another program, or for listing a part of a long file on the line printer. In the previous example the %ROUTINE B could be listed on the line printer using the following sequence of commands:

```
EDIT:M/%ROUTINE B/M/%END/M1S      Set separator at bottom of routine
*S*
↑%ROUTINE A
EDIT:OM-/%ROUTINE B/F<.LP>        List routine on line printer
↑%ROUTINE B
```

THE OPERATION OF THE EDITOR

Although it is possible to use the editor with no knowledge of its internal workings some users might appreciate a brief description. The editor makes use of the virtual memory and handles its files by directly addressing them (see chapter 1). When editing one file to another it first connects the file in the virtual memory and sets up pointers to the top and bottom. Any text that is inserted is stored in a work area in the virtual memory and each section of text has pointers to the top and bottom. Any operation which divides a section of text - for example removing a character from the middle of it, results in additional pointers being set up pointing to the beginning and end of the 'hole'. All these pointers are linked together in the logical order in which the sections they point to appear in the file. Note that this order may bear no resemblance to the order in which the sections are laid out in the store. When the edit command 'E' is reached, an output file is created, if necessary, and the sections of text are moved into it in the correct order, as determined from the linked list of pointers.

Note:

* Since the output file is not constructed until the E command is executed all editing is lost if a system failure occurs during editing.

* Since there are pointers to the top and the bottom, moving the cursor to these points with T and B is efficient.

  When relevant O is also an efficient command.

* Searching backwards for text using either M-/TEXT/ or M-n is considerably slower than searching forwards.

The command LOOK

The command LOOK is used to activate the editor for the purpose of examining, rather than altering a file. It takes one parameter - the name of the file to be examined - with a default of 'SS#LIST', the default compiler listing file (see chapter 9).

A similar effect can be achieved by typing

```
EDIT (filename,.NULL)
```

There are three differences between using LOOK and EDIT:

* the editor commands I, R, D, U, and G are not allowed since they alter the file

* the prompt at editor command level is 'LOOK:'

* the parameter for LOOK has a default of 'SS#LIST'

Otherwise the facilities available are identical.

The command RECALL

This command is used to interrogate the file containing a copy of all interactive terminal Input/Output operations for this user; see also chapter 21. It takes no parameter. There are three differences between RECALL and EDIT:

* the editor commands I, R, D, U and G are not allowed since they alter the file

* on entry the cursor is at the bottom of the file - i.e. pointing at the end of the most recent information

* the prompt at editor command level is 'RECALL:'

The table below shows the available commands and the parameter types that can be used with each. The final column shows which commands can be used with LOOK and RECALL.

Valid Parameters								
Command	Uproot	None	Integer	Text string	- Text string	Quote	Filename	Allowed in LOOK and RECALL
A	After		*	*	*	*		*
B	Bottom	*						*
C	Cancel		*					*
D	Delete		*	*		*		
E	Exit	*						*
F	File						*	*
G	Go to		*					
H	Hold	*						*
I	Insert			*		*	*	
K	Kill	*						*
M	Move		*	*	*	*		*
O	Over	*						*
P	Print		*	*		*		*
Q	Quit	*						*
R	Remove		*	*		*		
S	Separate	*						*
T	Top	*						*
U	Use		*	*		*		
W	Write	*						

The command FINDJOB

This command is used to obtain information about a job that has been put into the background job queue by a call of DETACH.

The command can be used in two ways:

* to obtain information about a specific job. In this case the name of the job is given as a parameter; for example

```
FINDJOB(ERCC0603)
```

* to obtain information about all jobs in the batch queue for this user, in which case no parameter is used:

```
FINDJOB
```

The resulting messages include the name of each job, the time limit and the name of the command file(s) DETACHed to make each job.

The command DELETEJOB

This command is used to delete a job that has previously been put into the background job queue by a call of DETACH. The parameter should be one jobname, or a list of jobnames.

Example:

```
DELETEJOB(ERCC0600,ERCC0607)
DELETEJOB(ERCC0601)
```

Detaching jobs to other computers

An alternative use of DETACH is for sending jobs to other computers. Currently, for most users, it can only be used for sending jobs to NUMAC. The file (or concatenated files) being sent should contain one job, including any job control statements, program source or data if required, in exactly the form that would be used if the job were submitted on cards. The details of the services available at NUMAC are described in reference 10. The second parameter to DETACH is used to nominate the processor to which the job is to be sent. Currently NUMAC is described by the abbreviation .P37, though this may change. Hence:

```
DETACH(JOB370,.P37)
DETACH(JCL+PROG+DATA23,.P37)
```

Note that, as with detaching jobs to the local job queue, a copy is made of the file to be sent, so the original files can be reused or destroyed as soon as the DETACH command has been accepted. The command QUEUES (chapter 21) can be used to determine whether the job is still waiting to be dispatched to NUMAC.

Controlling Background Jobs

The preferred method of providing sophisticated control of jobs in EMAS is to write a program, or command, in IMP or FORTRAN (see chapter 18).

The routine SET RETURN CODE

This routine must be specified:

```
%EXTERNALROUTINESPEC SET RETURN CODE (%INTEGER N)
```

The routine is used to set a return code before leaving a program. This return code, which must be in the range 0-4095, can be interrogated by the function RETURN CODE.

The function RETURN CODE

This function must be specified:

```
       %EXTERNALINTEGERFNSPEC RETURN CODE
```

This returns as its result the value last set for the return code. The return code is set in the following way:

* by a call of the routine SET RETURN CODE

* by the IMP, FORTRAN and ALGOL compilers - which set a return code of zero to indicate successful compilation, and non-zero otherwise.

* by use of STOP n in FORTRAN. Note that if n is omitted the value of zero is assumed.

Examples

The following example is a command TESTIMP which compiles an IMP program and, if the compilation fails, sends the listing file to the line printer:

```
       %EXTERNALINTEGERFNSPEC RETURN CODE
       %EXTERNALROUTINESPEC IMP(%STRING(63)S)
       %EXTERNALROUTINESPEC SEND(%STRING(63)S)

       %EXTERNALROUTINE TESTIMP(%STRING(63)S)
       !THE PARAMETER S SHOULD CONTAIN ONLY THE
       !NAMES OF THE SOURCE AND OBJECT FILES
       IMP(S); !  CALL THE COMPILER WITH SUPPLIED PARAMETERS
       %IF RETURNCODE # 0 %THEN SEND('SS#LIST')
       %END; !  OF TESTIMP
```

In the next example a program is compiled with all checks on and run with a small set of data. If it ends in order (i.e. calls SET RETURN CODE (0)) then it is re-compiled with PARM(OPT) and run again with a full set of data.

```
       %EXTERNALROUTINESPEC DEFINE(%STRING(63)S)
       %EXTERNALROUTINESPEC IMP(%STRING(63)S)
       %EXTERNALROUTINESPEC PARM(%STRING(63)S)
       %EXTERNALROUTINESPEC RUN(%STRING(63)S)
       %EXTERNALROUTINESPEC SEND(%STRING(63)S)
       %EXTERNALINTEGERFNSPEC RETURN CODE

       %EXTERNALROUTINE GO(%STRING(63)S)
       !THE PARAMETER S IS NOT USED IN THIS EXAMPLE
       PARM (''); !  SET DEFAULT PARMS
       IMP('PROG,PROGY')
       %IF RETURNCODE # 0 %THEN SEND ('SS#LIST') %AND %RETURN
       DEFINE ('ST1,TESTDATA'); !  USE THE TEST DATA
       RUN('PROGY'); !  THE TEST RUN
       %IF RETURNCODE # 0 %THEN %RETURN; !  THE PROGRAM FAILED
       PARM('OPT')
       IMP('PROG,PROGY')
       DEFINE('ST1,FULLDATA')
       RUN('PROGY')
       %END; !  OF GO
```

Notes:

* That to be effective the program should set a non-zero return code early in its execution so that if a program failure - e.g. UNASSIGNED VARIABLE occurs then this will indicate a fault when the return code is interrogated.

* Apart from testing return codes from the compilers and user programs, it is also possible to detect faults in EMAS commands (see chapter 18).

CHAPTER 20
ACCOUNTS AND USAGE INFORMATION

This chapter describes the method of charging users for their use of EMAS resources, the procedure to follow to obtain access to the system, and the related command PASSWORD. Finally there is a description of two information commands METER and USER.

GAINING ACCESS TO THE SYSTEM

Before using EMAS it is necessary to obtain authorisation. This is in two parts:

* Obtaining authority to use the services of the Edinburgh Regional Computing Centre. This is dealt with by the ERCC User Support Group.

* Obtaining accredited EMAS user status. This is dealt with by the EMAS Operations Manager.

The end product is a 6-character user name and two four-character passwords. The first password is used for logging into an interactive terminal for foreground access to the system; the second is used for card and paper tape input (see chapter 4). Either or both passwords can be changed by use of the PASSWORD command.

The command PASSWORD

This command can be used to change either or both passwords. Passwords should consist of any four printable characters other than comma. The command take two parameters, the foreground and background passwords respectively:

```
PASSWORD(FORE,BACK)
```

If it is only required to change one of them then the other can be omitted:

```
PASSWORD(7777)
PASSWORD(,CARD)
```

CHARGING FOR USE OF RESOURCES

Resources are charged for, and invoices are sent to the appropriate funding body. The User Support Group can provide further details of the accounting mechanism outwith EMAS itself. There are two types of charge:

* charges for file space

* charges for computing resources used

File space charging

There are three rates for charging for file space. Files held in the immediate store (on disc) are charged at two rates. The charge for files which are CHERISHed is higher than for the rest. This is in order to recover the cost of backing up the files and replacing them on disc in the event of a serious system or hardware failure. Files held in the archive store (on magnetic tape) are charged for at a lower rate. This reflects the lower cost of keeping material on magnetic tape rather than on the disc file, and the fact that the archive store is more easily extended than the immediate store. The current (August 1977) charges are:

Type	Charge (pence) per page per day
Immediate Store (CHERISHed)	0.23K
Immediate Store (un-CHERISHed)	0.14K
Archive Store	0.014K

where K is a constant. Currently it is 3.85 for most users (7.70 for commercial users).

Charges for computing

There are four elements to the charge made for computing:

* Central processor time (T) - this is the time in seconds during which a user's process is actually executing instructions, plus an allowance to represent the share his process makes of facilities provided by the resident supervisor.

* Page turns (pt) - this is a count of the number of pages brought into main store or written back to the disc or drum for a user process. See Chapter 1 for a description of paging.

* Connect time (ct) - this is the time, in seconds, during which an interactive terminal is connected to the process.

* I/O unit records (U) - this is a count of the number of unit records handled; for example, lines printed or cards punched.

The other elements in the calculation are

* Priority (P) - this is normally 1 but is reduced to 1/3 for jobs that are detached at 'LOW' priority (see chapter 19).

* A constant K - this depends on the class of user. Currently (August 1977) this is 3.85 for most users (7.70 for commercial users).

The charge is calculated using the following formula:

$$\text{charge (pence)} = K\left[P\left(T + \frac{pt}{250} + \frac{ct}{60}\right) + \frac{U}{300}\right]$$

Project codes

It is possible for one user to divide his computing charges among various projects. This is done by using the command PROJECT, which accepts as its parameter a two-character project code. The characters should be upper case letters or digits. For example:

```
PROJECT(A9)
```

All work done after typing this command will be charged to code A9 until either PROJECT is used again, or the user logs off. The User Support Group should be consulted about the way in which project codes are printed in user accounts.

The form PROJECT(?) can be used to determine the current project code.

The command METER

This command is used to obtain usage information and an indication of the amount charged thus far in the current session (i.e. since log-on). The command takes no parameter. Output is of this form:

```
        21/09/77  12.44.23  CPU= 6.15 SECS  CT=21 MINS  PT=4282  CH=168P
```

The information given is as follows:

* Current date and time
* CPU time, in seconds
* Interactive terminal connect time, in minutes
* Page Turns
* Approximate charge (on the assumption that this is a user charged at the standard rate). No allowance is made in this charge for unit record output.

The command USERS

This command, which takes no parameter, is used to print out the number of active processes on the user's System 4. This normally includes three system processes (see chapter 2). It provides an indication of the loading on the system and the response that can be expected. Currently each machine can run with up to about 45 users.

CHAPTER 21
ANCILLARY COMMANDS

The table below lists the ancillary commands available in the Subsystem - commands which do not readily fit into any of the categories covered by earlier chapters, or, as in the case of OPTION for example, commands which relate to several of the categories.

Command	Purpose
CPULIMIT	Used to set time limit for subsequent commands
DELIVER	Used to specify delivery information, to be printed on output files
OBEYFILE	Used to execute a sequence of commands
OPTION	Used to set a number of optional characteristics of the Subsystem
QUEUES	Used to print information about files awaiting output
STOP	Used to terminate foreground session
SUGGESTION	Used to send suggestion to the System Manager

The command CPULIMIT

This command is used to set the amount of central processor unit (CPU) time allowed for each subsequent command. It takes one obligatory parameter, the time in minutes, and optionally a second parameter to specify time in seconds. For example:

Command	Time set
CPULIMIT (3)	3 minutes
CPULIMIT (,10)	10 seconds
CPULIMIT (1,30)	1 minute and 30 seconds

Notes

* In order to provide good response for users of interactive programs there is a low limit imposed during busy periods on the maximum CPU time that can be set using CPULIMIT.

* Currently the maximum values for foreground use varies between 30 seconds and 10 minutes depending on the number of users logged on. For background jobs it is 120 minutes.

* The default setting for foreground access is 30 seconds and for background jobs it is 2 minutes.

* The command takes effect for the following and subsequent commands, and remains in effect until CPULIMIT is used again or the user logs off. If the number of users logged on increases such that the user's current setting is higher than the maximum allowed, a warning message will be printed on his interactive terminal, and the limit will be reduced to the maximum allowed. This will occur at the start of the first command after the limit has been exceeded.

* This command can be useful for testing programs that contain faults which result in infinite loops. By using CPULIMIT with a low value - perhaps 5 seconds - the elapsed time taken to reach the TIME EXCEEDED failure is considerably reduced.

* CPULIMIT has no effect on the OBEYFILE command (see below) but affects any commands called by OBEYFILE.

Control of Rate of usage

Apart from limiting the maximum value for CPULIMIT a mechanism exists to check the rate at which cpu time is used by a user in relation to elapsed time. If the rate of use over a period is unreasonably high, in the context of the number of users logged on, the user will, when he starts the next command, get a message on his interactive terminal "CPU RATE EXCEEDED". He will be allocated a small amount of cpu time to tidy up and he should log off as soon as possible. If he fails to do so he will be logged off automatically.

Notes

* This mechanism has been introduced to attempt to penalise "anti-social" use of the system i.e. running heavily cpu-bound work during interactive sessions. Such work should be DETACHed to run in background mode (chapter 19), where the rate check is not applied.

* The precise characteristics of the rate check are controlled by the system manager. They may be varied in the light of user experience and comments. It does not operate when the system is lightly loaded.

The command DELIVER

This command is used to specify the text to be printed at the start and finish of output files to assist Job Reception staff in distributing output. The parameter should be suitable text with a maximum length of 19 characters. No spaces should be used. The underline character is a suitable substitute. For example:

```
DELIVER(ALISON.HOUSE)
DELIVER(CHEMISTRY_K.B.)
```

Notes

* The registered name of the owner of the process is always printed on the output as well as the delivery information.

* The command takes effect immediately and remains in effect until another use of the DELIVER command.

* The form DELIVER(?) can be used to determine the current delivery information. It is printed in reply on the user's interactive terminal.

The command OBEYFILE

This command is used to execute a sequence of commands. The required commands and any data they would normally read from the interactive terminal should be put in a file, using the Editor or some other means. The format should be identical to that which would be used when typing commands on the interactive terminal. OBEYFILE takes one obligatory parameter, the name of the file containing the commands to be obeyed. Additionally a second parameter can be given to specify a file or device to be used for output. By default the output goes to the interactive terminal. For example:

```
OBEYFILE(NE26)
OBEYFILE(NRJOB,.LP15)
```

Among the commands included in the file to be obeyed can be further calls of OBEYFILE for other files. This process of nesting calls of OBEYFILE can continue to four levels. Note that for the second and subsequent levels the optional second parameter is ignored. This parameter is also ignored if OBEYFILE is called in background mode (see chapter 19).

The command OPTION

This command is used to set a number of optional characteristics for this user. The command takes one or more keyword parameters, which are listed below. The standard settings are underlined.

The call only affects the options that are specified, all others in force being left alone (cf. PARM). The options specified do not take effect until after logging off and logging on again; they remain in effect until the end of the session in which the command OPTION is used again.

QUICKSEARCH	When searching for standard Subsystem commands, do not search the user's own library index or any library indexes appended (see chapter 10).
FULLSEARCH	When searching for commands, always search current library index and appended library indexes first.
FULLMESSAGES	Print confirmatory messages from commands (see the table in chapter 6).
NOMESSAGES	Suppress confirmatory messages.
NORECALL	Do not store interactive terminal input/output messages for use by RECALL (chapter 12).
TEMPRECALL	Store interactive terminal I/O for use by RECALL for the duration of the current session.
PERMRECALL	Store up to 16 pages (64K bytes) of the most recent interactive terminal I/O for use by RECALL. This is kept between sessions.
STACK=n	n can take an integer value in the range 2-8. This specifies the size (in segments of 64K bytes) to be used for the stack file created by the Subsystem, with the name 'SS#STK'. The default is 2 segments (128K bytes), which will be large enough for the majority of programs. See chapter 10 and reference 6 for the use of the stack.
BRACKETS	Parameters to commands typed on an interactive terminal or included in an OBEYFILE file or a DETACHed command file, should be enclosed in parentheses. All the examples in this manual assume that this option is in force.
NOBRACKETS	Parameters to commands do not need to be enclosed in brackets. The command name must not include any embedded spaces, and must be separated from any parameters by one or more spaces. The commands paired together below have the same effect:

BRACKETS	NOBRACKETS
COMMAND:LIST(A,.LP)	COMMAND:LIST A,.LP
COMMAND:INSERT FILE(OBJ)	COMMAND:INSERTFILE OBJ
COMMAND:METER	COMMAND:METER

STARTFILE	This option can be used to nominate a file to be executed by OBEYFILE automatically at the start of an interactive session. Example: OPTION(STARTFILE=STARTUP). The file can contain calls on any commands and might be used, for example, to set a PROJECT code or a non-standard setting for PARM. Note that it is not possible to nominate an output file for the OBEYFILE. This option precludes the use of the option STARTCOMMAND.
STARTCOMMAND	This option can be used to nominate a single command to be called automatically at the start of an interactive session. A suitable standard command would be FILES, but any command could be selected including a user written command. Note that it is not possible to specify a parameter to be passed to the command. This option precludes the use of the option STARTFILE.
NOSTART	Do not use the automatic command execution facility.

The parameter ? can also be used, to cause the currently effective options to be printed.

Examples:

```
        OPTION(FULLSEARCH,STACK=4)
        OPTION(FULLMESSAGES)
        OPTION(?)
```

The command QUEUES

This command is used to determine the number of files belonging to this user which are held in EMAS queues awaiting output. It takes no parameter and output is of the form:

```
        FILES QUEUED
        LOCAL DEVICES LP CP   REMOTE NO: 37
        NO OF FILES   3  1             2
```

In this case there are three files waiting to be printed on the line printer, one on the card punch and two waiting to be dispatched to remote terminal 37.

Notes

* Normally files are output on a particular device in order of submission.

* In the case of remote terminals the output from QUEUES makes no distinction between the devices available. Hence files for .CP37 and .LP37 and .P37 all appear as files for Remote number 37.

The command STOP

This command is used to terminate a foreground session. It takes no parameters. It has the following effects:

* Prints out usage information as for METER (see chapter 20).

* Destroys all temporary files created during this session and the default compiler listing file SS#LIST if it exists.

* Disconnects interactive terminal - making it available for another user.

* Stops the user's virtual processor - freeing a slot for another user to log on.

* Indicates to the demons process that any NOW jobs waiting for this user can be started as soon as machine time is available.

The command SUGGESTION

This command is provided to make it easy for users to send suggestions for changes or improvements to EMAS to the System Manager. Its use is intended primarily for minor items which crop up during an interactive session which do not merit the formality of a letter. The facility should not be used for reporting serious faults - these should be reported to the Advisory Service as soon as possible.

Users are warned that although an effort will be made to reply to all SUGGESTIONS eventually, they should not expect a prompt response. An indication will be given in the reply as to the likelihood of their suggestion being implemented.

The method of use is to type the command with no parameter. The replies to the prompts 'SURNAME:', 'ADDRESS:' and 'TEXT:' should be the personal (not EMAS) name of the user, the address for a reply, and the text of the suggestion terminated with an asterisk on a line by itself. The example shows this:

```
        COMMAND: SUGGESTION
        SURNAME: DR G. JONES
        ADDRESS: 119 GEORGE SQUARE
        TEXT: TEXT OF SUGGESTION
        TEXT:*
```

As many lines of text as desired may be given.

APPENDIX 4
PARTITIONED FILES

This appendix describes the creation and use of Partitioned Files. A partitioned file is a complete EMAS file which has all the normal attributes of a file: a name, access permissions, cherish status and so on. Its contents are called Members, and each member is similar to a complete file, of any type. Thus one partitioned file might contain members which include character files, data files and even partitioned files. For many purposes a member of a partitioned file can be used in the same way as a file of the same type. For example if the member is of type character it can be listed on the line printer. There are, however, a number of exceptions to this rule - see below.

Creating a Partitioned File

The command NEWPDFILE is used to create a partitioned file. It takes one parameter - the name of the file to be created; for example

```
          NEWPDFILE(HOLD)
```

A file of the name given must not already exist.

Operations on a complete partitioned file

All of the general file utility commands described in chapter 7 can be used with a whole partitioned file - for example PERMITFILE and CHERISH, since these commands are not concerned with the contents of a file. Additionally COPYFILE can be used to copy a complete partitioned file and FILEANAL can be used to obtain a list of its members.

Naming of individual members

Each member of a partitioned file must have a name of up to 8 upper case letters or numeric characters, the first of which must be a letter. When referencing an individual member the member name is written after the partitioned file name and is separated from it by an underline (_). For example member LIST1 in partitioned file HOLD could be accessed thus:

```
          LIST(HOLD_LIST1,.LP)
```

If the appropriate access permission existed another user could access it. Example:

```
          LOOK(ERCC06.HOLD_LIST1)
```

Creating a member of a partitioned file

The command COPYFILE must be used to create a member of a partitioned file. The first parameter should be the name of the file whose contents are to be copied, the second is the full name to be given to the member, (see above). Examples:

```
          COPYFILE(SOURCE,HOLD_SOURCE77)
          COPYFILE(ERCC27.FILES,HOLD_FILES)
```

If a member of the same name already exists in the specified partitioned file its contents will be overwritten; if not a new member will be created. Any type of file can be copied in this way, including a whole partitioned file. There is effectively no restriction on the number of members in a partitioned file.

Destroying and Renaming members

The commands DESTROY and RENAME can be used in respect of individual members. The following examples should be self-explanatory:

```
          DESTROY(HOLD_FILED)
          RENAME(HOLD_TEST1,HOLD_OLDTEST1)
```

Note that when a member is destroyed, the remaining members are compacted to use the space it occupied. This means that there is no need for an explicit "tidy" operation.

Accessing individual members

In general individual members can be used wherever a file is to be read from. The main restriction is on the use of a member of a partitioned file by the loader (chapter 10). Neither Library Index files nor Object files can be used whilst they are members of a partitioned file. This restriction does not prevent their being stored in a partitioned file - it means that they have to be explicitly copied from the partitioned file into individual files before they can be used. The following pair of commands could be used to RUN the program held in PD_MYPROG

```
          COPYFILE(PD_MYPROG,TEMPOBJ)
          RUN(TEMPOBJ)
```

The following table indicates which of the standard commands can be used to access individual members of partitioned files.

Command	Notes	Example
ALGOL	Source only.	ALGOL(PD_ALGTEST,AY)
CONCAT	Both for input files and as control file	CONCAT(PD_CONCONT)
COPYFILE	for input and output (see above). Note that a member cannot be copied to another member in the same partitioned file.	COPYFILE(AB_SRCE,SRCE) COPYFILE(AB_NAME,BC_NAME)
DEFINE	For input file only.	DEFINE(FT5,FPD_DATA)
DETACH		DETACH(ACT_JCL,.P20)
EDIT	Can be used for input file and for I<filename> facility only.	EDIT(PD_EFILE,SRCE)
FILEANAL		FILEANAL(PD_FLENZ)
FORTE	Source only.	FORTE(FPD_SOURCE,Y)
IMP	Source only.	IMP(DIRSRCE_CPUT,CPUTY)
LIBANAL		LIBANAL(ARCH_LB)
LINK	Both for input files and as control file.	LINK(PD_LINK CONT)
LIST	But not SEND.	LIST(PD_OUTLIST,.LP)
LOOK		LOOK(DIRSPECS_CPUT)
OBEYFILE		OBEYFILE(PD_OBEY)

Efficiency Considerations

Partitioned files are particularly suited to applications involving many small files. This is because file space is allocated in units of a page (4096 bytes), which means that a file containing only a few hundred bytes of information contains a high proportion of wasted space. Even files larger than one page often contain a significant amount of unused space because they too are rounded up to a full page boundary.

Quite apart from the file space consideration, it is often convenient to be able to group sets of files together, and partitioned files provide a possible method. The following points should be noted, however, before embarking on the use of partitioned files:

* There is no significant difference between the cost of reading from a member of a partitioned file and reading from a conventional file containing the same information.

* The cost of adding a new member to a partitioned file is similar to the cost of making a copy of the same file.

* There can be a significant cost in destroying a member of a partitioned file. Note that a member is destroyed either by use of the explicit DESTROY command or as part of COPYFILE if the member being created has the same name as an existing member. This cost will not be significant if the whole partitioned file is only a few pages long, but for a large partitioned file - say more than 100 pages - the cost will be noticeable. Thus partitioned files are less suitable for applications involving frequent replacement.

INDEX