UNIVERSITY OF EDINBURGH

Department of Computer Science

Edinburgh Regional Computing Centre

Edinburgh Multi Access System

USER MANUAL

Edited by

J.G. Burns

First Edition

October 1972

UNIVERSITY OF EDINBURGH

DEPARTMENT OF COMPUTER SCIENCE

AND

EDINBURGH REGIONAL COMPUTING CENTRE

SYSTEM 4-75

EDINBURGH MULTI-ACCESS SYSTEM

USER MANUAL

FIRST EDITION

OCTOBER 1972

Work on EMAS began in October 1966 and, until September 1970, was the responsibility of the Edinburgh Multi-Access Project (EMAP) - a joint project between the Department of Computer Science at the University of Edinburgh and International Computers Limited. The then Ministry of Technology paid the University's share of the cost. The present system is the work of D.J. Rees, H. Whitfield and A.S. Wight of the Department of Computer Science (all members of the EMAP team) assisted by members of the Edinburgh Regional Computing Centre who have now assumed service responsibility for the System. Although a great deal of revision has been done, acknowledgement is due to the many members of the EMAP team who contributed ideas and basic software.

This Manual describes the user appearance of the System and the Subsystem which supports it. The latter is the work of ERCC staff, of whom Geoff Millard and Peter Stephens deserve particular mention.

Readers who are already EMAS users will appreciate that the rapid evolution of the facilities offered means that this Manual is already obsolete in certain areas. It is hoped however that it will serve a useful function until a more stable situation justifies a reissue.

OCTOBER 1972

J.G. Burns

CONTENTS

0. Introduction

0.1 The System

0.2 The Subsystem

1. Foreground User Guide

1.0 Name and Password
1.1 Logging In
1.2 Basic HELP Information
1.3 Resource Use and Description of System Facilities
1.4 The File System and Library Facilities
1.5 Logical Association of Files
1.6 Access to Physical Devices and Background Facilities
1.7 Calling Compilers, Editors and User Programs
1.8 Error Diagnostics
1.9 Interaction
1.10 Examples

2. Background User Guide

3. Archiving

4. Input/Output Facilities

4.1 The Console Demon
4.2 The Slow Device Demon

5. Accounting

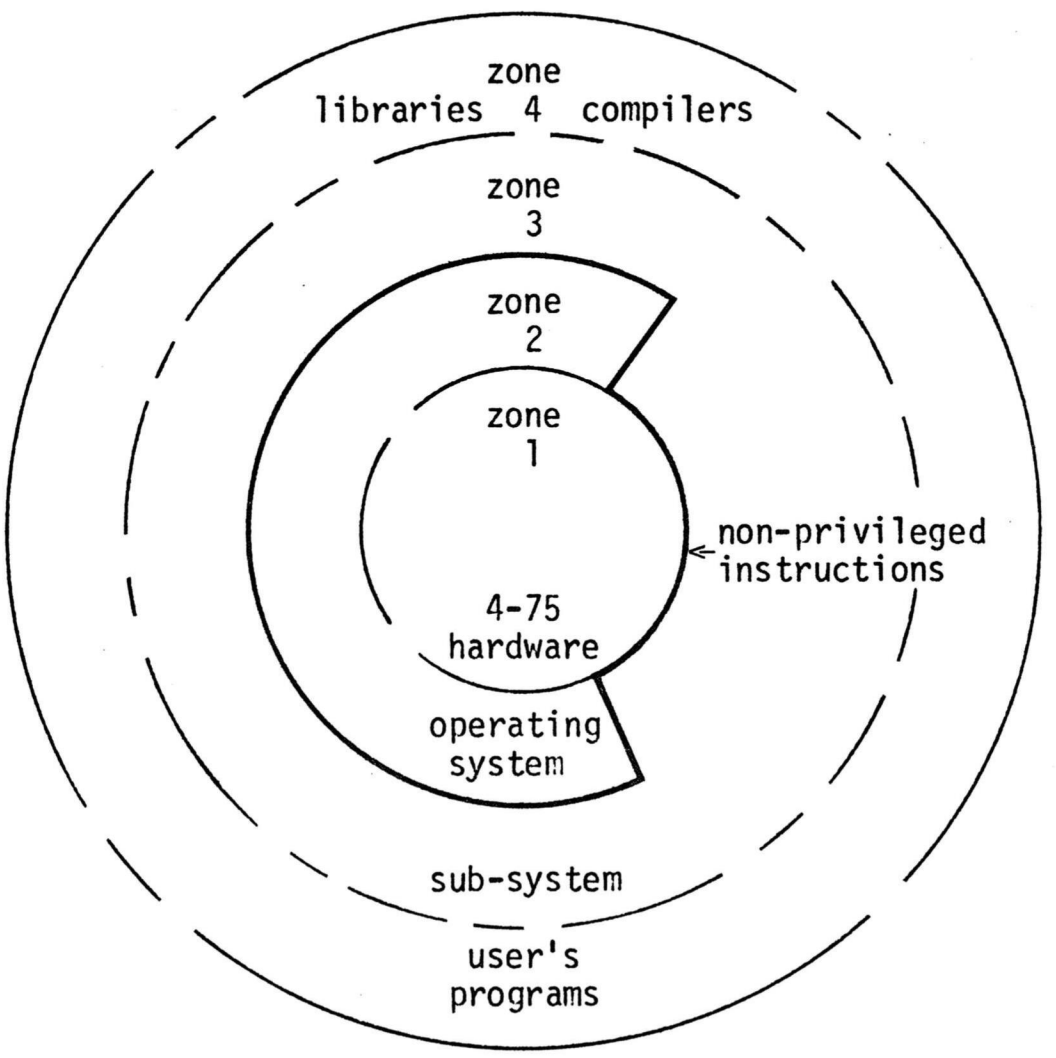
References

0. Introduction

0.1 The System

The USER is the person demanding service, which is provided by a complex of hardware and software consisting of the basic hardware, the operating system, the subsystem (command interpreter, file management utilities etc), library programs (compilers, editor, etc.,) and the user's own programs.

The EMAS Referece Manual(1) provides a definitive description of the appearance of the two inner zones of this complex as seen from zone 3. This section outlines the action of the System in so far as it affects a user, and is intended to provide a general guide rather than a definitive statement.



The execution of a User's job is referred to as a USER PROCESS, which is owned by and accounted to the User who initiates it.

The computing resources of the System are shared amongst the current processes in such a way as to give the impression that each process has a processor to itself.

A PROCESSOR consists of a processing unit (C.P.U.) and an addressable memory, areas of which may be associated with information held in file storage.

The 4-75 has a 24-bit addressing system and addresses units of one byte (8 bits) and can thus address 2**24 (=16,777,216) bytes which is much larger than the actual core store of the machine (currently 786,432 bytes). A processor concerns itself with addresses in this VIRTUAL MEMORY and has no knowledge of actual core addresses. It is the responsibility of the System to maintain the appropriate correspondence between the processor's virtual memory and the real physical locations of the programs and data addressed by it.

The backing store hierarchy which is used by the System to maintain the virtual/real correspondence is organised into ACTIVE STORAGE and two levels of file storage, IMMEDIATE and ARCHIVE storage.

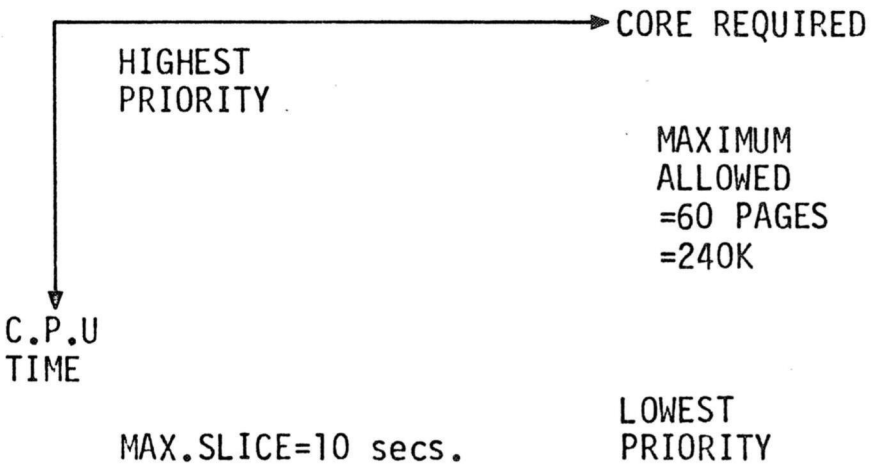
The active storage (6M-bytes of drum storage) provides a buffer between core storage - which is the only area on which the processing unit can operate - and the file storage, and is used by the System to facilitate the efficient running of active processes. The units which the system moves between the various levels of storage are PAGES (4096 bytes).

The file storage provides permanent storage for files belonging to Users of the system, the immediate storage being currently a 700M-byte fixed disk file which is always on-line when the system is running, whilst the archive storage requires operator intervention before it can be used and is normally a magnetic tape.

The layout, maintenance, and security of files in file storage is the responsibility of the system, although the contents and use of the files (e.g. as program files, direct access data files etc., whether read only, read/write etc.,) is the concern of the process which uses the file. The System allows the owner to specify a choice of access permissions to a file, ranging from read only private to read/write shared, the immediate consequence of this being that only one copy of system and library programs need exist for all processes. This attribute, SHARING, effectively increases the apparent core size of the machine by at least 25%.

The System, as indicated above, is responsible for sharing the available resources among the current processes and, although a process has no control over this function, an understanding of the scheduling algorithm can assist in the planning of programs, mode of access to data etc.

A process is assigned a priority by the System on the basis of its predicted demand for C.P.U. time and for pages of core storage. A process will be rescheduled if it exceeds either of its current allocations (for C.P.U. time and core). The scheduling space may thus be represented by the following diagram:-



The System, however, arranges to remove pages from core if they are unused and a process may thus proceed through a large memory area without incurring severe scheduling penalties provided it does so in a systematic fashion. The scheduling is thus designed to penalise those processes which require heavy system intervention to allow them to run and to favour those which make small demands on the System. Most interactive processes, in particular the subsystem editor, fall into the latter category.

It will by now be evident that the System concerns itself entirely with files held at various levels of storage. Input/Output in the conventional sense (e.g. to a line printer) is not directly controlled by a user process, but by special processes owned by the System called DEMONS, which are able to transfer information between peripheral devices and files. Convenient access to the demons is provided by the subsystem and the facilities available are described in a later chapter.

As yet, no mention of a console (or any other communication device) has been made since all the concepts introduced apply equally to all processes using the System.

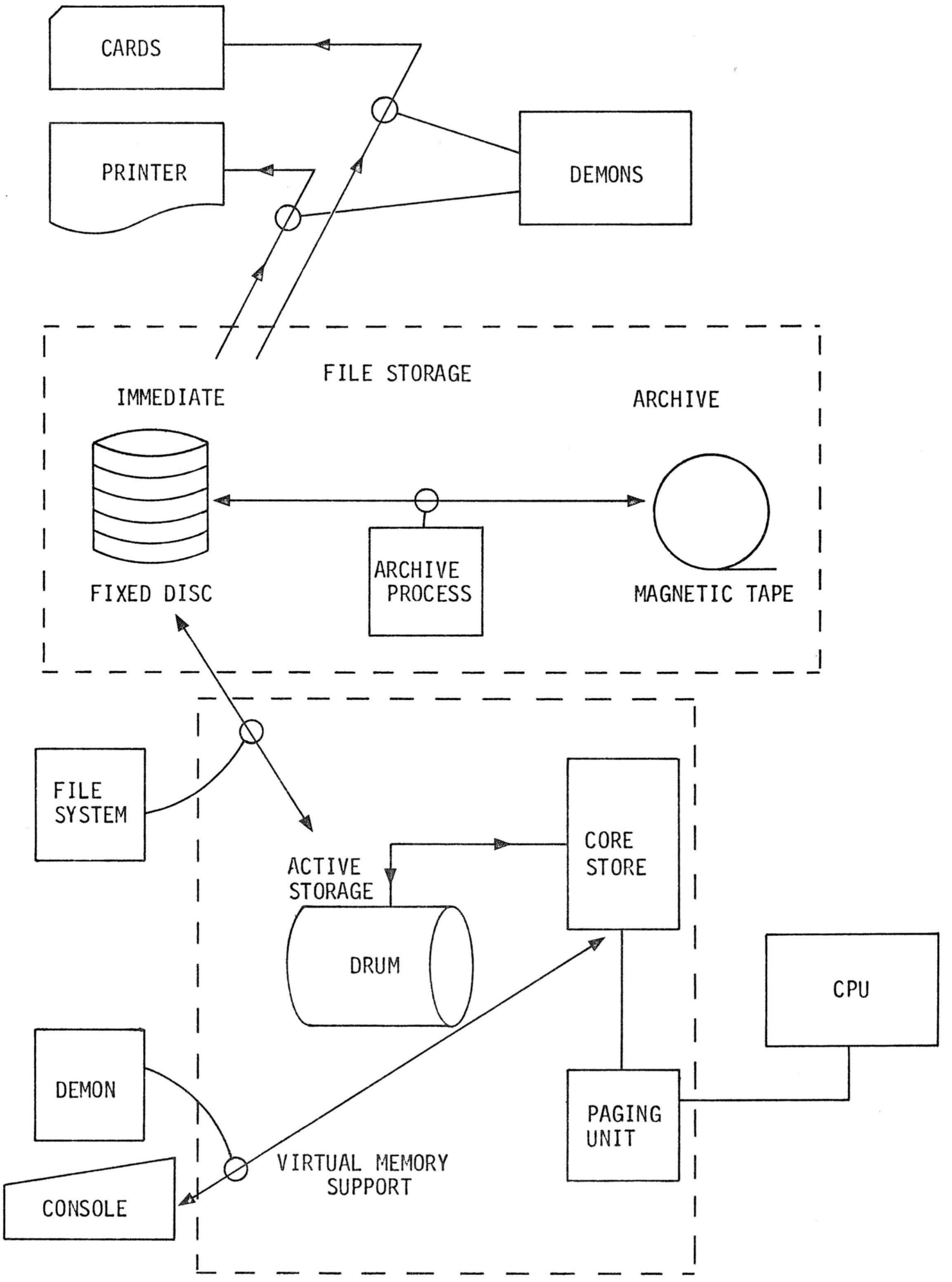
It is however relevant to introduce a distinction between FOREGROUND and BACKGROUND processes. A background process may be regarded as a conventional job on a batch processing system, whilst a foreground process, which 'owns' one (or more) consoles has additional capability for interacting with the User.

At the simplest level, a process may for example, reject illegal data and ask for it to be repeated. The System also allows the User to queue messages to his process without otherwise affecting its execution. The interrogation of these messages - at suitable nodes in the process - provides a powerful means of directing a process. Certain messages are reserved for System use and, in general, cause entry to diagnostic and recovery procedures.

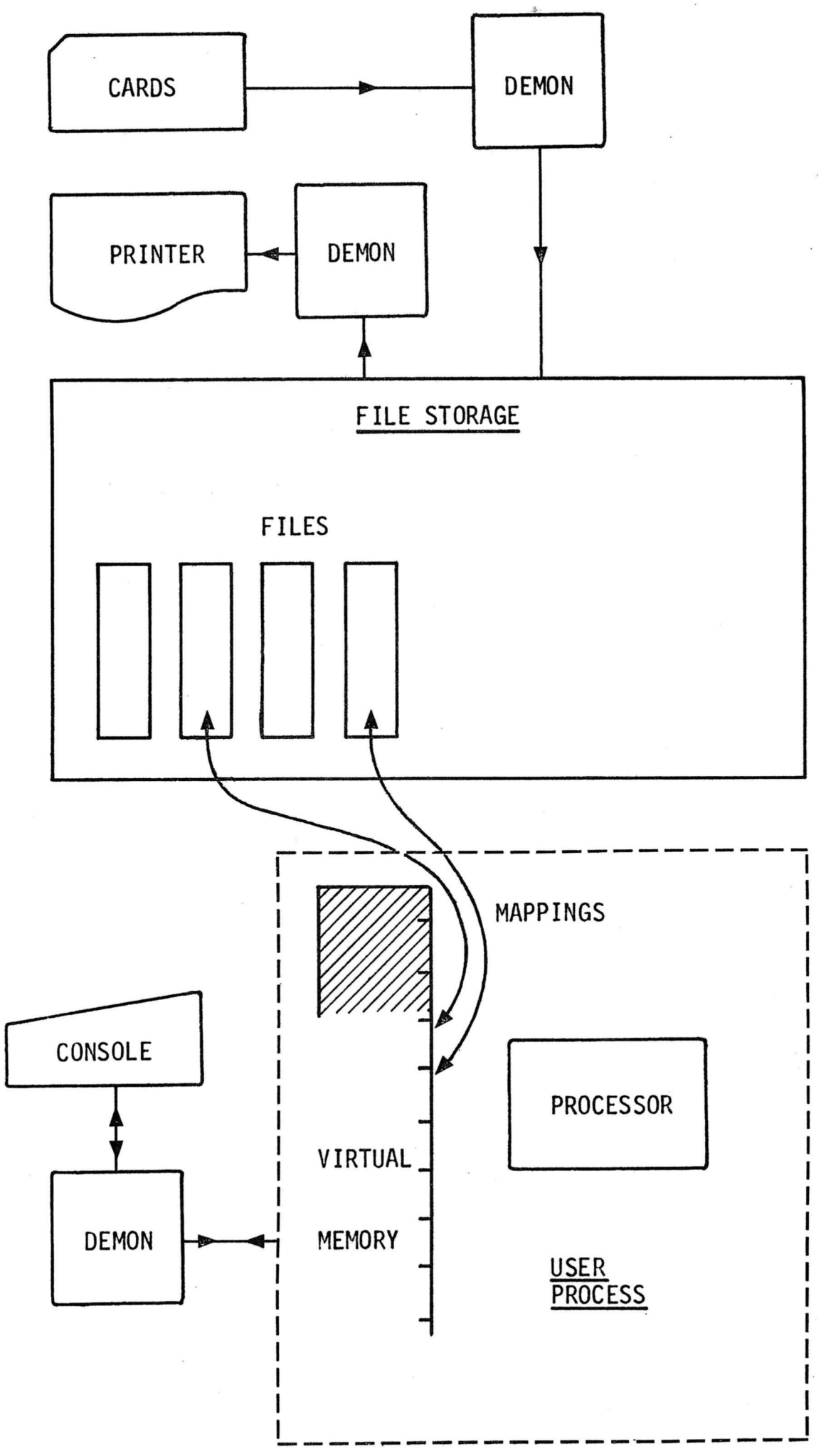
The attributes of the System, from the User viewoint, may then be summarised as follows:-

1) The scheduling of system resources in a prescribed manner.
2) Control and maintenance of the File System.
3) Provision of interactive and other communication routes.
4) Accounting for use of System resources.

S Y S T E M S T R U C T U R E



U S E R ' S V I E W


CARDS
DEMON
PRINTER
DEMON
FILE STORAGE
FILES
MAPPINGS
CONSOLE
PROCESSOR
VIRTUAL
DEMON
MEMORY
USER
PROCESS

0.2 The Subsystem

This outline description refers to the currently supported Subsystem on EMAS which is fully described in the Subsystem Reference Manual(2). The user should appreciate that the supported subsystem is in no way unique relative to the System and merely represents a convenient collection of components to facilitate use of the System.

The heart of the subsystem is the main control routine, the BASIC COMMAND INTERPRETER (BCI). This interprets a command from the console, or from the job document in the case of a background job, and causes entry to the designated routine. On return it interprets the next command and so on.

Grouped around the BCI are routines which provide the following main functions:-

1) Interrogation of and intructions to the System and Subsystem including:-
   Logical association of files with streams, direct access files and sequential files.
   Creation and maintenance of user files and libraries.
   Control of access permission.
   Running of user programs.

2) Access to sub-system library programs, such as the IMP and FORTRAN compilers and libraries.

3) Access to information on the current facilities and state of the System and Subsystem.

4) Access to the slow device demons.

5) Provision for passing interactive messages to and from the System and for initiating recovery procedures in certain circumstances.

All components of the Subsystem are external routines and may thus be invoked either at command level or dynamically within a user program. The subsystem is thus (almost) entirely symmetric with respect to command and program function. A user can extend and modify the command language to suit his own requirements.

To take advantage of the ability to share files, the IMP and FORTRAN compilers produce shareable code. The subsystem and the libraries are composed of shareable code files, and only one copy of any component exists, no matter how many processes are concurrently accessing it. However the working variables used by the processes are replicated in each virtual memory, so that no interference results from this sharing.

1. Foreground User Guide

Information on the use of EMAS from a console is available from two main sources, this guide and the HELP facility which enables the user to dynamically query the system. These sources complement one another to a certain extent, the guide attempting to provide fuller background information whilst HELP attempts to give concise and up to date notes.

1.0 NAME and PASSWORD

Before you can use EMAS, you need a username and a password. The service support unit (667-1081 ext. 2637) will accept requests.

1.1 Logging In.

Read the instructions on the console, switch on and set on DUPLEX. Press the space bar (several times if necessary).
The system responds, if it is available:

EDINBURGH MULTI-ACCESS SYSTEM

NAME:

Type your name followed by return and line feed
System responds:

PASSWORD:

Type your password followed by return and line feed. Your password will not be printed.

The system will now respond with one of the following messages:

PROCESS STARTED	date time	Normal start
PROCESS RUNNING		Already in use
INVALID NAME		Unknown username
INVALID PASSWORD		Incorrect Password
SYSTEM FULL		Too many users
NO USER SERVICE		System testing in progress

If the user is unable to start his process, he may obtain information from the 4/75 operator on 667-1081 ext. 2612.

Having successfully started a process, the system hands control to the subsystem which proceeds to initialise various conditions and files and to report the verson number by:

SUBSYSTEM version date

This may be followed by a brief message reporting, for example, a hardware condition or a change in schedule. The system will then rise to basic command level and issue the prompt

COMMAND:

indicating that the user should instruct it on how to proceed.
The user now has access to the HELP facility by responding
HELP CR-LF
which will cause the basic list of available commands to be typed on his console as follows:

1.2 Basic HELP Information

The format of a command is :

```
        COMMAND  (  PARAMETER  )  CR-LF
or      COMMAND   CR-LF
```

when typing commands spaces are ignored and double quote characters may be used to delete incorrect characters as far back as the beginning of the current line.
Commands available are :

```
ALERT        :   print system alert information
APPENDLIB    :   add library
BATCH        :   run batch job
CHERISH      :   mark file for archiving
CLEAR        :   clears DD definitions
CONCAT       :   concatenate source files
DDLIST       :   prints out current DD definitions
DEFINE       :   associate file with logical I/O channel
DELIVER      :   change delivery information
DESTROY      :   destroy a file
DETACH       :   send job to batch queue
EDIT         :   context editor
ENTRIES      :   prints the names of entries in an object file
FLIST        :   prints out list of files
FORTE        :   compile FORTRAN source file
HAZARD       :   cease archiving file
HELP         :   usage information
IMP          :   compile IMP source file using IMP(AA) compiler
INSERT FILE  :   insert object file into library
LIBINFO      :   prints library contents
LINK         :   link object files
LIST         :   list a source file
LUNAR        :   demonstration program
METER        :   prints metering information
PARM         :   set compiling parameters
PERMIT FILE  :   permit access to other users
REMOVE FILE  :   remove object file from library
REMOVELIB    :   remove library
RENAME       :   rename a file
RUN          :   run object file
SEND         :   lists and destroys source file
SET STREAMS  :   defines multiple streams
STOP         :   stop subsystem and log out
USERS        :   number of users on system
PASSWORD     :   reset passwords
```

More information about individual commands may be obtained by typing the name of the command as the parameter to the HELP command

Example : HELP(EDIT)

More information is available concerning system facilities under the following heads :

```
ACCOUNT      ARCHIVE
FILES        INPUT
INTERRUPT    PROMPT
SCHEDULE
```

EXAMPLE: HELP(FILES)
A complete line printer listing of the current 'HELP'
information can be obtained by typing: HELP(.LP)

These commands can be conveniently divided into five main groups and the following sections discuss the facilities in each group.

1.3 Resource Use and Description of System Facilities.

There are nine commands which enable the user to obtain general information about the system and to change certain system parameters associated with his process. In addition there are facilities which enable him to interact, in a general way, with his process.

The commands are:

ALERT which prints, on the console, up-to-date messages concerning the system and subsystem, for example warning that a specific feature is faulty or incorrectly described. The subsystem may 'prompt' the user at log-in time to consult the alert information.

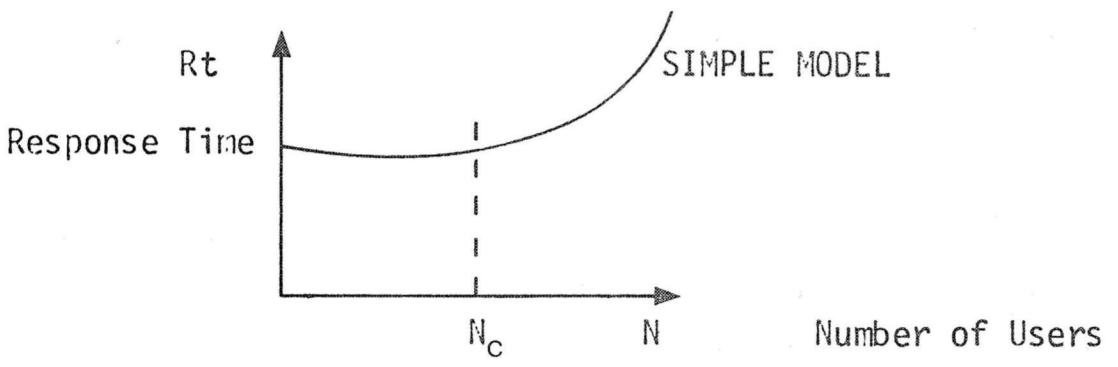
DELIVER changes the delivery information which, together with the username, identifies all output to the line printer, card punch and paper tape punch. The new information, up to 19 characters, is passed as the parameter to the command. Note that spaces cannot be included as they are removed before the string is passed - the underline character is an acceptable substitute. Delivery information remains in force until a subsequent change
DELIVER (ALISON_HOUSE)

HELP has already been discussed.

METER prints on the console information concerning the use made of the system in the current session and also the amount of file space currently allocated to the user.

STOP calls METER to report the use made of the system, causes all files in use to be returned to immediate storage, disconnects the console and stops the process. This is a complex operation involving many checks for consistency and may take a considerable time to complete if the system is heavily loaded. If the system detects that a process has been totally inactive for approximately 12 minutes, it automatically initiates the stop sequence, reporting on the console the reason for so doing. This measure is designed to maximise availability of the system, particularly when there are restrictions on the number of processes which may be logged-in.

USERS reports on the number of processes currently using the system. This can give one an indication of the response one may expect from the system. Simple models suggest that response to trivial requests may be constant up to a critical number of users and will thereafter deteriorate linearly as further users log into the system. The diagram below indicates this model and experience to date suggests that for the 4/75 with the current work load that N , the critical number of users, is about 30.



The critical number of users is closely dependent on the availability of the full complement of fast storage, in particular the drums and core store. The loss of a single drum will reduce N to about 75% of the normal situation, and users are advised to appraise themselves of the hardware state of the system as frequently as possible.

PASSWORD enables the user to redefine both his foreground password, used for logging in from a console, and his background password which enables him to submit work to be run in batch mode. The distinction is made to attempt to preserve security for accredited users since a knowledge of the background password will not allow one to log into a process from a console.

A password must be four printable characters and they are passed as a parameter to the command:
PASSWORD (+**?,BACK)

1.4 The File system and Library facilities.

1.4.1 The File System

The System, as we have already indicated, is responsible for the basic maintenance of the immediate and archive file storage, but does not concern itself with the manner in which a file is used, nor with the detailed control of the loading of object files.

The File Directory Package (FDP) provides the user interface to the basic file system by maintaining detailed information concerning the use and relation of files which are made available to the foreground user through a set of thirteen commands.

Files may be created by a variety of commands and the rule followed is that a new file is automatically created if no suitable file exists. Files are named in the conventional fashion :

USERNAME.FILENAME

where filename may be up to 8 characters, must start with a letter, and may use only alphanumeric characters.
e.g.
EGNP67.SURVEY2

When working from the console, the username is implicit when referencing one's own files, but must be included if reference is made to another user's files.
The maximum number of files which may by owned by a process should be discussed with Service Support when a user is first accredited.

The subsystem also provides for default selection of suitable file sizes and for automatic extension where appropriate, up to a default maximum size. In most circumstances, the default maximum size is 1/4M bytes. Thus a call on a compiler must name an existing source file but can cause an object file and listing file to be created, and the definition of files as output streams for a user program will cause their creation where appropriate.

In situations where the user wishes the file to be sent to a physical device, he may use a set of formal file names which normally cause a temporary file to be created which is then sent by the subsystem to the nominated device :

FORMAL NAMES	.TT	Teletype, for both input and output
	.LP	Line Printer
	.MLP	'Money' Line Printer (4.2.3)
	.CP	Card Punch
	.PP	Paper Tape Punch
	.BCP	Card Punch - Binary Data
	.BPP	Paper Tape Punch - Binary Data
	.NULL	A null output or input source file.

The user may query and manipulate his files using the following commands :

FLIST prints a list of the files belonging to a user. The device or file on which the listing is produced may be passed as a parameter, the console being the default
FLIST(.LP)

Files are listed in alphabetic order and those which have been marked as requiring archive protection are indicated with an '*'.

FLIST(.TT,ALL)
A second parameter, ALL, may be passed which causes additional information concerning each file to be listed.

FILEINFO(filename) gives information about the length of the file, and whether it is currently in use by its owner or by any other users who have been permitted access to it. If the file is in use, then the mode of access is also reported.
e.g.
FILEINFO(MATRIX)

```
* MATRIX CONNECTED AT SEGMENT 58      Being used by self, file cherished
FILE LENGTH   :  7 PAGES
ACCESS MODE   :  SHARED READ
NO. OF USERS  :  3                    Two other users are using file
ACCESS PERMISSION: SELF F OTHERS 8    Other users have read shared access.
```

A user may request information on another user's file which will be granted, provided the necessary access permission has been given.

PERMITFILE(filename,username,accessmode)
allows the user to grant a variety of access permission to a file, both for himself and for other users of the system.

The file directory contains information on each file, which determines who may access it and in what modes. On creation of a file, the owner is given permission for all modes of access, and no permission is given to anyone else. PERMITFILE enables access permission to be given to other people, or any permission already given to be changed. By using the command repeatedly several people may be allowed access.
One can also give access permission to 'everyone' by setting 'username' in the above comand to null. Users may then access the file in the modes permitted to everyone, as well as in the modes permitted by any individual access permission.

The access mode is specified by a four bit field in which the bits have the significance

read shared (most significant)
write shared
read unshared
write unshared

The principal combinations, which are specified to PERMITFILE by a hexadecimal digit, are :

```
0 = no access      :  may be useful to protect a backup copy of a file
2 = read unshared  :  to prevent a file being overwritten
8 = read shared    :  to allow several users access to a file
F = any mode       :  default on creation
```

```
PERMITFILE(JACK,ERCC45,8)
PERMITFILE(TOM,,8)
```

The components of the subsystem, to which everyone has access, reside in a normal process and have been permitted to all users in 'read shared' mode. Note that this means that only one copy of a compiler need physically exist although many users may have it connected to their virtual memory. Sharing of components in this way assists throughput and minimises use of file storage.

DESTROY Files are not destroyed on completion of the operation which caused their creation, unless the subsystem has explicitly created them for a temporary purpose and the command DESTROY enables the user to return unwanted space to the system. He may destroy one or more files :
DESTROY(FILE1)
DESTROY(FILE2,FILE3,FILE4)

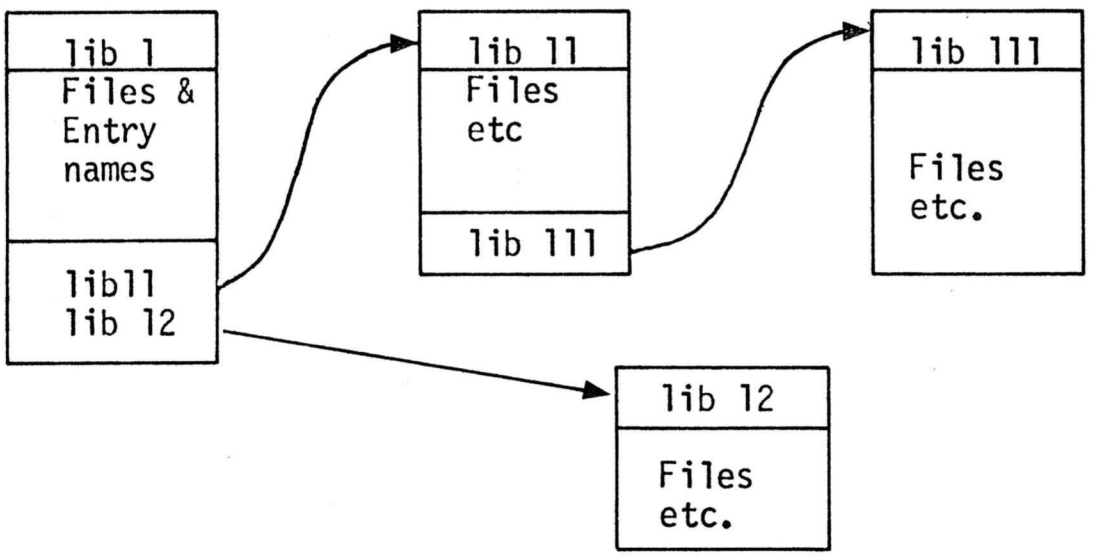
Users are advised to tidy up their files at frequent intervals.

RENAME has the obvious meaning, e.g.
REAME(OLDNAME,NEWNAME)
Care should be taken when renaming files which have been permitted to other users, the rule being that the access permission moves to the newname. Object files should be removed from the library (see REMOVEFILE) before being renamed.

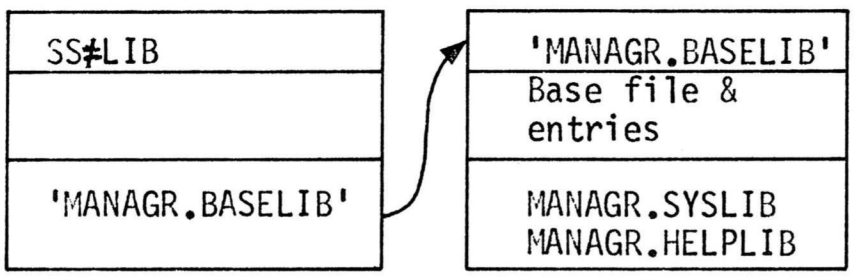
1.4.2 The Library System and Linkage Conventions

FDP also provides, through the subsystem, facilities for setting up and maintaining a library structure which enables fully automatic loading of programs to be accomplished.

A library, which is a normal file from the system viewpoint, may be thought of as containing a list of entry names and their associated files - a catalogue - and also a list of pointers to other libraries



On attempting for example, to load and run a program the complete library structure is searched to satisfy any references made by the primary object. The order of search in the example above would be lib 1, lib 11, lib 111, lib 12. The present subsystem provides the following structure when first started :



SS≠LIB is the owner's own library, initially containing no files, but with a link to the library BASELIB owned by the system manager, MANAGR, which contains all the subsystem entry name (commands) and is shared by all users. BASELIB also contains links to other libraries containing, as indicated, the HELP information and the System library components (sin, cos etc).

The user has a set of commands which enable him to alter the contents of his library and to alter the linkage structure between libraries.

INSERTFILE (filename) allows an object file to be added to the library. If the object file consists of a complete program, then the entry name is that of the file itself whilst a set of routines causes entries to be made for each routine name. A library is not allowed to contain duplicate entries and a search of a library to satisfy references will thus always be unique.

Certain procedures, each as compilations, cause INSERTFILE to be called on completion, and thus further ease the maintenance of the library

REMOVEFILE (filename) is the converse of INSERTFILE and is also invoked by compiler procedures - the object file being removed from the library, if it had been entered, before compilation.

LIBINFO (libraryname)
lists the files which have been entered into the library and also the libraries to which links have been set

```
LIBINFO (SS LIB)
 FILES :
ERCC03.TEST1
ERCC03.PAL11
LIBRARIES :
MANAGR.BASELIB
```

Repeated calls on LIBINFO thus enable the user to explore completely his library tree since he will have been given access permission to any libraries which are pointed to.

ENTRIES (libraryname, filename)
causes a complete list of the entries associated with a specific file to be output on the console. As already noted, the entry name corresponds to the file name for a complete program.

CREATION of LIBRARIES
The Service Support Unit will, on request, create an additional library within a user's file system and insert a nominated set of files into it, permitting a nominated set of users access to this library. Such libraries should contain material of general interest which must be adequately tested. The Service Support Unit can take no responsibility for the performance or documentation of such private libraries. SSU does however, make available to user's certain standard libraries which currently include SSP and SPSS as well as the KEYSET libraries.

APPENDLIB (library name)
allows a further pointer to be added to a library and, together with
REMOVELIB (library name)
permits the link structure of the library tree to be amended.

It will be evident that there are opportunities for creating linked lists with undesirable properties - such as a circular list. The subsystem attemps to protect the user from such situations but users should treat these facilities with caution, checking frequently that they have achieved the desired effect.

1.4.3 File System Utilities

CONCAT enables the user to concatenate several files into a single file. These will normally be source files.
The concatenator will query the user for the list of files:

```
CONCAT
CONC: FILEA
CONC: FILEB
CONC: END
FILE: FILEC
```

A parameter may be passed to CONCAT in which case the data for the concatenator should be in the file whose name is passed:

```
CONCAT(CONCLIST)
```

LINK has a parallel function to the concatenation of source files enabling the user to consolidate several object modules into a single module. It accepts data in the same way as CONCAT.

Both these commands may fail if the input files are not of the correct type but both will overwrite the output file, whatever its previous contents and type. The output file should not, in general, be one of the input files.

Archiving of user files held on immediate storage is fully covered in section 3 but the following commands enable the user to access the facility.

CHERISH (filenames)
causes the nominated files to be marked as requiring archiving
CHERISH(A,B,C)
while
HAZARD (filenames)
performs the reverse function.
Note that files are not cherished on creation but that user files which are not cherished are regarded as semi-permanent - that is they will not be destroyed without warning except through a serious hardware or software collapse. Users should therefore cherish files which would be impossible or difficult to recreate but may choose to leave other files in the semi-permanent state.
(A source file may be cherished but the corresponding object file, which can be recompiled, may be semi-permanent)

1.4.4 Transfer of Ownership

A user may wish to transfer ownership of a file to another user and this process is accomplished in two steps for security reasons - not only must the first user be prepared to transfer the file, but the second user must be prepared to accept it.

OFFER(filename, username) performs the first function. An offer may be revoked by calling OFFER (filename).

ACCEPT(filename) allows an offered file to be accepted into the recipient's file index and removed from the first user's index. A second parameter enables the collected file to be renamed on being accepted.
ACCEPT(ERCC03.FRED,JACK)

1.5 Logical Association of Files

The file system, as so far described, has made little reference to the type of the file since, to the system, this is of no consequence. The programmer, however, wishes to associate physical files of differing characteristics with his logical input/output streams.

The Subsystem provides facilities for maintaining tables of logical association through a data definition list and for providing the mappings necessary for the various file types.

1.5.1 Definition of Logical Association

DEFINE is the command which causes an entry in the data definition table and which, if necessary, implies creation of the file
The complete form is:
DEFINE (ddname,filename,size,type)
The ddnames which are recognised imply, for the first time, a language association and the following table lists the ddnames recognised with respect to the currently supported languages, IMP and FORTRAN

ddname	Language	File Type
STREAMnn STnn (1 =nn =80)	IMP	character stream
SQFILEnn (1 =nn =80)	IMP	sequential binary data
DAFILEnn (1 =nn =80)	IMP	direct access binary data
FTnnF001 FTnn	FORTRAN	sequential file

Logical numbers may not conflict between different ddnames. The filename may be any legal name (1.4.1). The filesize is expressed in Kbytes, the default being 255 and the maximum 1023. Note that, except for direct access files which are cleared on creation (to the unassigned pattern for IMP), the actual file size is maintained at approximately the size required to hold the current data.

The record format is normally implied by the ddname, the defaults, where choice exists, being as follows:

ddname	Language	Default	Options
SQFILEnn	IMP	V1024	F,FA
FTnnF001	FORTRAN	V1024	

Direct Access files in IMP have a fixed blocksize of 1024 bytes and are defined by a language statement in FORTRAN.

The concept of a blocked file (VB or FB) does not exist in EMAS since all files are physically blocked in page units regardless of the mapping imposed by the subsystem.

DDLIST causes the current logical association of all data definitions to be reported on the console.

CLEAR causes the data definition table to be completely cleared while an additional parameter enables a specific definition to be cleared:

CLEAR(DAFILE22)

CLEAR(STREAMS) has the specific effect of clearing all STREAM definitions whilst leaving direct access and sequential file definitions intact.

1.5.2 Additional Facilities

Concatenation of appropriate files is allowed through the '+' operator:

DEFINE(STREAM11,A+B+C)

Streams and sequential files may be concatenated, the implication being that they are input files.

Specification of the disposition of a file, which is not normally necessary, is through the '-' operator:

DEFINE(SQFILE11,FRED-MOD)

Recognised dispositions are NEW, OLD, MOD, SHR

The intention has been that in almost all situations the default parameters when defining a logical stream will be appropriate and DEFINE will, therefore, require only two parameters. An alternative view of the options available is given in section 2 where they are discussed from the viewpoint of the batch user.

1.6 Access to Physical Devices and Background Facilities

Both these functions are performed by a demon which, as previously described, spools all input and output on the fixed disk. The batch processor, which runs background work, may in this context be regarded as a process to which a file may be sent.

1.6.1 Output of Files

LIST (filename, device, copies)
causes the named file to be listed 'copies' times on 'device'
Recognised device names are:

.TT	Console
.LP	Line printer
.MLP	'Money' Line printer (4.2.3.1)
.CP	Card punch
.BCP	Binary Card Punch (4.2.1)
.PP	Paper Tape Punch
.BPP	Binary Paper Tape Punch (4.2.3.2)

List causes the file to be copied and ownership of the copy to be transferred to the demon. The file thus still exists in the user's process.

SEND (filename, device, copies)
has a similar function to LIST except that no copy is made and the file is effectively destroyed by this command.

The first parameter is obligatory for both commands, the second is .TT and .LP by default for LIST and SEND respectively, whilst the third, one by default, is normally not required.

1.6.2 Access to Background Facilities

Background use of EMAS is fully described in section 2 and these notes relate to communication between foreground and background.

BATCH enables a job which has been placed, by any means, in the background stream to be placed into execution from the console. If no parameter is given, the first job in the queue for the process will be executed, but a specific job may be chosen by adding the job identifier as a parameter :
BATCH(EGNP03AB)
The normal use of BATCH is envisaged as causing the entry of card or paper tape files to the file index of a user process (2.1.5).

DETACH (filename, mode) causes a file containing appropriate JCL and, if necessary, source text to be placed in the background queue for the process. The user may also specify the urgency of the job through the second parameter :

```
DETACH(JOBA,NOW)          ;! run as soon as possible
DETACH(JOBB)              ;!run overnight
```

Note that an 'as soon as possible' request may mean that an attempt to log in from a console later in the day will find the process running the background job and will report accordingly.

1.7 Calling Compilers, Editors and User Programs.

The Regional Centre supports two languages, IMP and FORTRAN, both being implemented on EMAS. The intention in FORTRAN is to provide facilities corresponding to the ASA definition while IMP, which is a local derivative of Atlas Autocode and closely akin to ALGOL in structure, is the language which will support interactive system dependent features. It should be noted that, subject to the parameter passing limitations of FORTRAN, a user may execute a program which calls routines in either language and that the diagnostics reflect correctly the source language for each routine entry.

The primary intent of EMAS is to cater for free format input and the editor operates in a free format context mode. Utilities, or options to the compiler, may be provided to transform EMAS source files to the fixed format demanded by standard FORTRAN.

.7.1 Compilers

The standard parameter list required by a compiler (or assembler) is a source file to compile from, an object file to compile to, and a listing file in which errors are also reported in a standard format.

IMP(source, object, list) calls the current IMP compiler. The listing file may be omitted in which case the System work file SS#LIST is used. Primary diagnostics are reported on the console unless the null file (.NULL) is used for listing. Detailed diagnostics may be obtained by using the Editor to search the listing file (see example in HELP).

FORTE(source, object, list) calls the current Edinburgh FORTRAN compiler, as is described for IMP.

Note that the compiler procedures remove entries associated with the object file from the library before compilation and insert them after successful completion (1.4.2). A number of options which are described in the language manuals may be specified to either compiler using

PARM(option list)

The available options, the default being asterisked are :

*CHECK	*LIST	*TRACE	*ENTRIES
NOCHECK	NOLIST	NOTRACE	NOENTRIES
ARRAY	*DIAG	*NOMAP	OPT
NOARRAY	NODIAG	MAP	LABEL

A call on PARM first resets to the default state and then obeys the option list. The default options need therefore never be specified and will be faulted if they are. Parm settings remain in force until reset by a further call on PARM.
PARM(NOCHECK,NOARRAY)

1.7.2 Editors

EDIT(oldfile, nextfile) accesses the context editor. The properties of the editor are fully described in the HELP information and, since it is a highly interactive program, its properties are best learnt by direct experiment rather than by 'off-line' reading.

1.7.3 Execution of User Programs

RUN(filename) causes the object file 'filename' to be executed provided it is a main program block. If the program is an IMP program, the initial input stream is the console (STREAM00) and the initial output stream is also the console (STREAM00) whilst a FORTRAN main program selects FT05F001 as the console for input and FT06F001 as the console for output.

External routines (IMP) and routines (FORTRAN) may be called directly by name from the console but the only parameter which may be passed is of type 'string' thus effectively limiting this function to the IMP user in accordance with the philosophy described. Note that the string passed to an external routine consists of the characters within the bracket pair, with double quote deletion performed and spaces removed :

```
MYCALL(DO NEXY''T TASK)      ;  DONEXTTASK passed
```

The user will note that he is able to create his own command language and also to redefine the meaning of standard commands if he wishes to. It will now also be apparent, as was hinted at in section 0, that subsystem commands are calls on external routines in the IMP sense and may be called from within programs. Note however that the standard parameter, %STRING(63) S, must now be enclosed in quotes or passed as a string. (see Examples 1.10).

1.8 Error Diagnostics

The foreground Subsystem attempts to report in a meaningful way on errors which arise because of inconsistent requests or other errors. In general, the System returns a flag when a request is made to it for a service (e.g. to create a file) and a non zero value denotes the specific error. We are attempting to interpret these flags and pass an English message to the console as well as the flag but

FAULTINFO(command, flag) is provided to supplement the basic fault reply. If a non-specific fault is reported in response to command, a call on FAULTINFO naming the command and the error flag should provide further information :
FAULTINFO(LIST,3)
The commands currently recognised, or their abbreviations, are :

CREATE	REMOVE(FILE)	CONNECT
APPENDLIB	FILEINFO	DISCONNECT
REMOVELIB	LIBINFO	ENTRIES
INSERT(LIB or FILE)	DESTROY	LIST
PERMIT(LIB or FILE)		

1.8.1 More Serious Faults

Faults may occur from which the Subsystem is unable to recover directly. In these circumstances it will attempt to report the basic nature of the fault and to return to command level. Some faults, particularly communication failures, are sufficiently serious as to cause the System to disconnect the console with no diagnostics.
Reported Faults:

SIGNAL WT. 4	Transfer failure from disk or drum. May result in a 'replaced page' (all zero) in a file
SIGNAL WT. 92	Address error. Should not occur with subsystem programs or User programs running with checks on. May occur with incorrect user program running with checks off.
SIGNAL WT. 84	Opcode error. Either a compiler fault or machine error.
SIGNAL WT. >=130	Input/Output Control Program error. Should not occur.

1.9 Interaction

Provision is made for a user program to prompt the user to respond to a specific request and also for the user to interrupt his process to place a message in a queue for it.

1.9.1 Prompts

The user will have already noted that the Subsystem prompts him with
COMMAND:
when it has exhausted all available data and that the Editor uses a variety of prompts depending on the context.

A user may, from within a program, issue a prompt of his own choosing, provided he is working in IMP. He should specify
%EXTERNALROUTINESPEC PROMPT(%STRING(15) M) at the head of his program and may then call, for example
PROMPT('BAD DATA:')
at an appropriate point
Note (4.1.1) that a prompt is only issued if no input data is available and is independent of normal input/output streams.

1.9.2 Interrupts

The user may interrupt his process at any time by pressing the 'ESC' key on the console (or any key if output is in progress). This will cause the prompt
INT:
to be typed by the System (1.4.3)
A null response, i.e. CR-LF , is ignored.
Single character responses are reserved for System use and currently have the following meanings

```
INT:A          ; Abort all action and
                 return to Command level

INT:Q          ; Abort current action, call
                 diagnostic procedure to report,
                 then return to command level.
```

Multiple character responses, up to 15 characters, are stored by the System and may be interrogated by the user. Up to 8 distinct interrupts may be queued and a successful interrogation removes that entry from the list.

%EXTERNALINTEGERFNSPEC TESTINT(%INTEGER C,%STRING(15)INT) should be specified at the head of the program and a call will return a value 0 if it finds a matching prompt:
%IF TESTINT(0,'REPORT')≠0 %THENSTART
(C should be 0)

1.10
Examples

```
%EXTERNALROUTINE MYDESTROY(%STRING(63) S)
!TRIVIAL EXAMPLE WHICH ALLOWS YOU TO CHANGE YOUR
!MIND AFTER REQUESTING THAT FILES BE DESTROYED.
%EXTERNALROUTINESPEC DESTROY(%STRING (63) S)
%EXTERNALROUTINESPEC PROMPT(%STRING(15) MESSAGE)
%INTEGER REPLY

      PRINTSTRING('DO YOU REALLY WANT TO DESTROY THESE FILES?')
      NEWLINE
      PROMPT('Y OR N:')
SKIP: READSYMBOL(REPLY)
      ->SKIP%UNLESS REPLY='Y'%OR REPLY='N'
      DESTROY(S)%IF REPLY='Y'
%END


%EXTERNALROUTINE DISPATCH
!LISTS FILES WITH STANDARD NAMES ON VARIOUS DEVICES
%EXTERNALROUTINESPEC LIST(%STRING(63) S)
LIST('FILE1,.LP')
LIST('FILE1,.CP')
LIST('FILE2,.PP')
LIST('FILE3,.TT')
%END

%ENDOFFILE
```

The effect of calling these routines is as follows (note that the standard names required by DISPATCH do not exist)

```
COMMAND:MYDESTROY(TOM,ALEC,DICK)
DO YOU REALLY WANT TO DESTROY THESE FILES?
Y OR N:NO

COMMAND:DISPATCH
FILE1 DOES NOT EXIST
FILE1 DOES NOT EXIST
FILE2 DOES NOT EXIST
FILE3 DOES NOT EXIST
```

2. Background User Guide

2.0 Introduction

EMAS provides facilities for running conventional batch processing jobs, and this section of the manual describes these facilities. For operational convenience there are two types of jobs; those which make no use of permanent user files, and those which do. An example of a job of the first type would be a compilation of an IMP program and a run of the program reading all its data from cards and producing output on the line printer. The job control language for this type of job is described below and in the IMP and Edinburgh FORTRAN reference manuals. Anybody who has an E.R.C.C. job number can run jobs on EMAS in this way.

The second type of job requires access to permanent user files. Before running work in this category you must be an accredited EMAS user (see Section 1.1). Your username and its process are common to both foreground and background access. This means that you can access files using either the foreground facilities described in section 1 or using the background facilities described here. Normally it is not possible for these two methods of access to occur simultaneously.

2.1 Job Control Language

The Job Control Language used for background access to EMAS is based on I.B.M. OS/360 Job Control Language. There are three types of job control statements relevent to EMAS users:

(a) JOB delimiter statements which are used mainly for accounting and scheduling purposes.
(b) EXEC statements which are used to indicate which facilities are required, and to specify which options are to apply when using the facility.
(c) DD - Data Definition cards are used to define the files to be accessed.

A job always starts with a JOB statement and can comprise one or more steps, each of which starts with an EXEC statement. Each EXEC statement should be followed by one or more DD statements. The last statement of the job should contain // in columns 1 and 2 and nothing else.

2.1.1 Format of JCL statement

(a) Job Control Statements are punched in the first 71 columns of the line.
(b) Columns 1 and 2 always contain //

(c) Spaces within JCL statements should only occur where indicated by ␣ in the text below. Either one or several spaces can be used.
(d) If a statement is too long to fit on one line it should be stopped at or before column 71 with the comma separating two parameters, and continued in one of columns 4 to 16 inclusive on the next card. The first two columns of continuation cards should contain //.

2.1.2 JOB statement

This is the first statement of a job. Its form is:-

//jobname␣JOB␣[(controlparams)][,'programmername']

where

jobname — consists of username followed by two optional characters to distinguish this job from others for the same user.

controlparams — this field is enclosed in brackets and uses keyword parameters which may appear in any order and should be seperated by commas. Keywords, permitted abbreviations, meanings and defaults are as shown below.

Keyword	Abbreviations	Meaning	Default
CARDS=	C=	No of output cards	100
LINES=	L=	1000s of lines	1
PASS=	PA=	Background password	None
PRTY=	P=	Priority	Standard
TIME=	T=	Job time in minutes	2

Notes

The background password is only required on card or paper tape jobs which access permanent user files. It is not required on DETACHED jobs.

The maximum number of lines output allowed is 10000.

The only valid priority is LOW. It is only significant on card or paper tape jobs, and by punching it the user can ensure that the job is not run in the prime shift. In the case of jobs which use no permanent user files the job will be charged at the minimum rate.

'programmername' — This field is only needed for jobs which use no files. The text, maximum 19 characters within the quotes will be printed on the output from the job and will be used as delivery information. For file handling jobs the text is provided by the foreground DELIVER command.

2.1.3 EXEC statements

The EXEC statement is used to introduce a job step. The form of an EXEC statement is:-

//stepname␣ EXEC␣ procedurename [,PARM='pppp'][,TIME=tt][,COND=(condition)]

where:-

stepname — is optional. It is used with the refer-back facility, and with the COND facility - see below.

procedurename — valid names are IMP, FORTE, FILE, SIMRUN. The procedures are described below.

PARM='pppp' — This is used to select compiler options'. The details are described in the IMP and Edinburgh FORTRAN reference manuals. It may be omitted when the default options are suitable.

TIME=tt — This is used to set a CPU time limit for the step. The default is 40 seconds. tt can be in minutes, e.g. TIME=3 or in minutes and seconds e.g. TIME=(1,50)

COND=(condition) is an optional parameter used to check the return code set by one or more preceding steps. If any one of the stated conditions is satisfied the job step is bypassed. Thus, this is a means of preventing further execution of a job if one part of it fails

(condition) is a relation of the form (n,operator ,stepname) where

n is an integer in the range 0-4095 and operator is one of

- EQ - equal to
- NE - not equal to
- LT - less than
- LE - less than or equal to
- GT - greater than
- GE - greater than or equal to

stepname is the name of the step on whose condition code this check is to be made. If stepname is omitted, the check is applied to the condition codes from all previous steps in the job. Multiple conditions may be tested

e.g. COND=((condition),(condition))

The code n is set to standard values by IMP and FORTE compilers (see 2.1.5)

Fortran programs may generate a condition code by the STOP n statement. IMP programs do so by the routine specified as:

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

Examples of EXEC statements
//␣ EXEC␣ IMP,TIME=10,PARM='NOARRAY,NOCHECK'
//A␣EXEC␣ FORTE
//STEP1␣ EXEC␣ SIMRUN,TIME=20

2.1.4 DD Statements

DD statements are used to select files to be used for the current step. Some DD defnitions are implied by the chosen procedure - e.g. FORTE has an implied definition which links logical file 6 with the line printer. A basic DD card is of the form

//ddname␣ DD␣ DSN=username.filename,DISP=(before,after)

where

ddname — describes the type of file and, where relevent its logical number e.g. LINKIN, DAFILE02, FT10F001. (see 2.1.5 below)

username — is the username of the person owning the file. This will normally be the same as the username on the job card, but if access permission has been given (see PERMITFILE in section 1.4.1) you can read from files belonging to other users.

filename — is up to eight alphanumeric characters, the first of which must be a letter.

before,after — The DISP parameter describes the status of the file before and after this step. The first parameter should be

- NEW if the file does not exist
- OLD if the file already exists.
- SHR if the file is old and only to be read from. This provides a means of write protection.

The second parameter should be

- KEEP if you wish to keep the file after this step
- DELETE if you wish to destroy the file after this step
- PASS if you wish to pass the file to a later step, in which you will define what you want done with it.

Alternative forms of DD cards are

//ddname␣ DD␣ *

This indicates that the file itself follows immediately in the same stream. This is used, for example in an all card job to introduce the program source cards.

//ddname␣ DD␣ SYSOUT=A

This indicates that output on the logical file indicated by the ddname should be printed on the line printer, and similarly, SYSOUT=B indicates output to the card punch.

Additional parameters
SPACE=(record size, no. of records)
where

record size is the average size of records being written. In the case of Variable record length files the record size should allow for the 4 bytes of control information added to each record.
no. of records is the approximate number of records to be written to the file.
SPACE is ignored except when the file DISP is NEW. If it is omitted a default of 255K bytes is used. The maximum is normally 511K bytes. The SPACE parameter is used to determine the maximum size a file is allowed to reach. The size of the file in the file storage system is determined by the amount that has been written to it, and this is the size used for charging purposes. In the case of direct access files the space occupied is as defined at the time of creation.
DCB=(RECFM=rr,LRECL=n)

where

rr is the record format.
n is the record length.
For most purposes it is unnecessary to provide DCB information. The following table shows where it can be used.

FILE TYPE	LANGUAGE	COMMENTS
STREAM	IMP + GENERAL	not needed
OBJECT	IMP + FORTE	not needed
DAFILE	IMP	not needed
SQFILE	IMP	can be used (default V, 1024)
SEQUENTIAL	FORTE	can be used (default V, 1024)
DIRECT ACCESS	FORTE	not needed - set up by DEFINE FILE statement

For data files the default Record format is V and the default LRECL is 1024. Valid alternatives for RECFM are F and FA.

Referback
An alternative method of referring back to temporary files is available for multi-step jobs. The second DISP parameter of the file being passed must be PASS. It can be used by all users, not just those who own permanent files. The form of the DSN parameter is

DSN=*.stepname.procstepname.ddname

where stepname is the name of a previous step (see 2.1.3)
procstepname is the procedure stepname. This is 'ONE' except in the case of FORTE (see 2.1.5)
ddname is the ddname of the file to which reference is being made

Example
```
//A␣ EXEC␣ IMP,PARM=NORUN
//OBJECT␣ DD␣ DISP=(NEW,PASS)
//SYSIN␣ DD␣ DSN=ERCC06.IMPR,DISP=OLD
//␣ EXEC␣ FORTE
//C.SYSIN␣ DD␣ DSN=ERCC06.FPROG,DISP=OLD
//G.LINKIN␣DD␣ DSN=*.A.ONE.OBJECT,DISP=(OLD,DELETE)
//
```

Concatenation
It is often useful to be able to use more than one file for the same DDNAME. For example, two files containing compiled routines may be needed in one step. This process which is referred to as concatenation is achieved by omitting the DDNAME from the second and subsequent DD statements.
e.g.
```
//LINKIN␣ DD␣ DSN=ERCC06.RT1,DISP=OLD
// ␣ DD␣ DSN=ERCC06.RT2,DISP=OLD
// ␣ DD␣ DSN=ERCC06.PR3,DISP=OLD
```

Concatenation can be used with the following DDNAMES:-
LINKIN
FTnnF001 - sequential data files only - Read only
SQFILEnn - Read only
STREAMnn - Read only

2.1.5 PROCEDURES
EXEC FILE
This procedure is used to create new files from cards or paper tape. There are no parameters for the EXEC statement. A DD statement is required for each file to be created. The file name will be the DDNAME on this statement.

Example:
To create two files PROG1 and PROG2 in the process ERCC06
the following JCL could be used

```
//ERCC06FF ␣ JOB ␣ (PA=PASS)
// ␣ EXEC ␣ FILE
//PROG1 ␣ DD ␣ *,NOIDENT,QUOTES,TRAIL
      text of PROG1
//PROG2 ␣ DD ␣ *
      text of PROG2
//
```

The optional parameters NOIDENT, QUOTES, TRAIL are described in section 4.2.2.1

EXEC IMP
This procedure is used to compile and run an IMP program. The optional parameters on the EXEC card are described in the IMP reference manual. The following DD cards are recognised

DDNAME	PURPOSE	NOTE
SYSIN	Program source and data to be read on default input stream. (STREAM98)	Essential
OBJECT	Compiled program will be put in this file.	Only needed if you want to run program again using SIMRUN
LINKIN	Nominates file, or files of compiled external routines to be linked in after compilation.	Optional
DAFILEnn	Used to define Direct Access Files - used by READDA routine etc.	Optional
SQFILEnn	Used to define Sequential Files - used by READSQ routine etc.	Optional
STREAMnn	Used to define user streams other than default streams.	Optional

The following files are defined and used unless overridden by user definitions.

OBJECT	Temporary file - deleted after end of step
STREAM99	Line Printer - DCB=(LRECL=121,RECFM=FA)
STREAM97	Card Punch
STREAM95	Paper Tape Punch

If there are compile time faults, the return code is set to 8. If compilation fails because the OBJECT file cannot be opened, the return code is set to 16. In both cases, the program is not run.

EXEC FORTE
This procedure is used to compile and run a FORTRAN program, using the Edinburgh FORTRAN compiler. The procedure initiates two steps a Compile step, which has a procedure stepname of 'C' and the GO step which has a procedure stepname of 'G'. DD names and EXEC parameters have to be qualified by the procedure stepname and all parameters relating to the 'C' step must come before those relating to the 'G' step.
e.g.
// ␣ EXEC ␣ FORTE,TIME.C=5,TIME.G=15

COMPILE STEP
The compile time options which can be put on the EXEC statement are described in the Edinburgh FORTRAN reference manual. The DD statements are

DDNAME	PURPOSE	NOTES
C.SYSIN	used to introduce program source text	essential
C.OBJECT	used to hold compiled program	only required if you wish to run compiled program later using SIMRUN

If there are compile time faults, the return code from the C step is set to 8. If the OBJECT file cannot be opened, the return code from the C step is set to 16. In either case, the G step is not executed.

GO STEP
DD Statements

DDNAME	PUPOSE	NOTES
G.LINKIN	Used to nominate file or files of compiled routines to be linked in.	Optional
G.SYSIN	Used to define data to be read on logical file 5.	Optional
G.FTnnF001	Used to define file with logical number nn.	Optional

By default FT06F001 is used to define the line printer with default DCB=(LRECL=121,RECFM=FA) and FT07F001 defines the card punch.

EXEC SIMRUN
This procedure is used to run a compiled IMP or FORTE program. The file containing the comiled program is defined on a DD statement with the DDNAME 'LINKIN'. Apart from this, the file definitions are as for IMP and the Go step of FORTE except that whereas in IMP, SYSIN is used for program and data, here it is used only to introduce input data for use on Stream 98. The procedure step name is 'ONE'.

2.1.6 Running Background Work

Jobs which make no use of permanent files should be submitted to Job Reception on cards or paper tape and will be run as soon as machine time is available.

Jobs which use files can be punched on cards or paper tape and submitted to Job Reception. Normally they will be run over-night to ensure that they do not conflict with foreground access to your files. They can be run during the daytime by use of the foreground command BATCH. This facility is intended primarily for EXEC FILE jobs. An alternative method of running background jobs is to initiate them from the teletype using the foreground command DETACH. In this case the JCL statements are generated in a file and its name is used as the parameter to the DETACH command. The file can be created using the foreground editor. The file can contain text as well as JCL statements.

For example the following sequence of commands could be used to run the file PROG1Y which reads data from STREAM01.

```
EDIT (.NEW,JCL)
EDIT:I.//ERCC06JC ␣ JOB ␣ (L=5,T=5)
.:// ␣ EXEC ␣ SIMRUN
.://LINKIN ␣ DD ␣ DSN=ERCC06..PROG1Y,DISP=(OLD,KEEP)
.://STREAM01 ␣ DD ␣ *
.:7    3    208    15    -1
.://
.:.
EDIT:E
COMMAND:DETACH(JCL)
```

2.1.7 Programs used in both Foreground and Background Mode
There are some differences in the use of default file definitions between foreground access and background access. The implied definitions for data files are:-

LANGUAGE	DDNAME	FOREGROUND	BACKGROUND
IMP	SYSIN STREAM98 INPUT-STREAM00	TELETYPE	FILE DEFINED AS SYSIN
IMP	STREAM99 OUTPUT STREAM00	TELETYPE	LINE PRINTER
IMP	STREAM97	CARD PUNCH	CARD PUNCH
IMP	STREAM95	PT PUNCH	PT PUNCH
FORTE	FT05F001	TELETYPE	FILE DESCRIBED AS G.SYSIN
FORTE	FT06F001	TELETYPE	LINE PRINTER
FORTE	FT07F001	CARD PUNCH	CARD PUNCH

3.0 Archiving

The immediate storage on EMAS consists of 700M byte of fixed disk on which a file must normally reside for a user process to access it. Provision is made for archiving these files on to magnetic tape at both the system level - to enable the fixed disk to be restored in the event of its contents being lost - and at the private level to enable users to secure important material.

3.1 System Archiving

A user may mark a file using the CHERISH command as requiring to be archived. (1.4.3)

This action sets a marker in the file header and markers are also set by the system whenever the file is used and also when it is written to. This combination of markers enables the system to archive all cherished files on a weekly basis and to archive those which have been written to on a daily basis. The use marker enables the system to detect inactive files which are being archived. In the event of a loss of information from the disk it should, therefore, be possible to restore it to the state of the last archive - not more than twenty four hours ago.

3.2 Private Archives

Users who wish to retain private copies of important files or who wish to clear their files from the fixed disk for a period may have this done at any time by arrangement with the Service Support Unit. Such requests, which may be either to archive or retrieve a file, will normally be completed within 24 hours.

3.3 Other Files

For completeness mention is made of the two other categories of file which are resident on the fixed disk. Temporary system files (e.g. SS≠LIST) are normally allowed to remain on the disk, to avoid the overhead of creation, but may be destroyed at any time.

User files which are not cherished are regarded as semi-permanent and will not be destroyed unless it is necessary to reprime the disk. Notice will be given, using ALERT, of any planned restorations. Users are encouraged to keep material in semi-permanent files if it can be easily recreated from archived files. The cost of the storage reflects this (5).

4.0 Input/Output Facilities

A User process running in its virtual memory performs 'input-output' by transferring information to and from files which have been connected to its memory. The actual transfer of information to and from peripheral devices is regarded as being divorced from this and is accomplished by special processes called DEMONS.

The subsystem provides convenient methods for making requests to the demons and these facilities are described in section 1.

This section seeks to describe the basic properties of the demons rather than their appearance as seen from the subsystem.

4.1 The Console Demon

The console is a unique device in that it provides both an input and output stream at the User process level and an interactive input/output stream using the interrupt and prompt facilities.

4.1.1 Input

The User must first succesfully complete the Log in procedure described in 1.1. The subsystem will then select the console for input on stream 0 and the user may then proceed, if he wishes, to select it for input on other streams (1.5).

The user may type his input ahead of any requests to read made by his process, although he would normally wait for the process to prompt him for input. If the User requires a prompt message other than the default message (DATA:) provided by the subsystem he may assign this using the PROMPT routine in the subsystem (1.9). Note that the prompt message will only be issued if no input has been typed, and also that it is completely independent of any output stream. Input is transmitted to a process, in response to requests, in 'blocks', rather than character by character, a block consisting of either a string of characters terminated by one of the characters 'LF' (line feed), 'EM' (end of medium), 'ETX' (end of text) or 128 characters.

If the User types more than 128 characters ahead of read requests by his process the demon will send the message ' WAIT' to the console. The demon will signal the User to ' CONTINUE' when his process has accepted the block of data. Any characters typed between a wait-continue pair will be treated as interrupt characters (see 4.1.3.). The subsystem, although it is handed blocks of input by the demon will normally pass them to the User on a character by character basis.

4.1.2 Output

As for input the console is initially selected by the subsystem on stream 0, and other output streams may be selected as required (1.5). The demon splits lines longer than 72 characters by inserting a CR/LF. A User process will normally be suspended if it sends more than 128 characters to the demon until some of this output has been typed.

4.1.3 Interrupts

The user may interrupt his process at any time by pressing the 'ESC' key, or any key if output is in progress. The system will prompt 'INT:' and will then accept a character string up to 15 characters long. Up to 8 such messages may be queued by the system and facilities are provided in the subsystem (1.9.2) for interrogating them. As noted in 1.4.1 any characters typed while ' WAIT' is in force, are treated as interrupt characters.

4.2 The Slow Device Demon

4.2.1 File Formats

Internal source text files are normally stored in free format ISO and thus consist of variable length lines separated by carriage control characters. These are:

10 = newline (LF)(NL)
12 = newpage (FF)
13 = carriage return (CR)

Various input options are available which are described in detail for each device. The normal mode of output consists of splitting the file into line images and discarding redundant CR characters. Full interpretation of the carriage control characters is only possible on the line printer.

Binary files may be transferred directly to or from paper tape whilst the internal format corresponding to column binary cards is a sequence of 160 byte records the least significant 12 bits of each 2 byte pair corresponding to rows 12,11,0,---,9 of the card.

4.2.2 Input

The DEMON will accept input documents from the card reader and the paper tape reader, a document being delimited by document control language (DCL) statements. Four types of DCL statement are recognised.

(1) JOB statement
(2) DATA Document statement
(3) DATA Document delimiter statement
(4) JOB delimiter statement

The JOB statement marks the start of a new job document (and the end of any improperly terminated previous document). It takes the form
//<username><job discriminator> JOB (<resource list>),'<title>'
the various fields having the meanings described in section 2 with the password being the background password of the User. If no password is specified, the DEMON passes the document to the Batch Process (section 2).

The Data Document statement marks the start of an embedded data document, and, where appropriate, the end of any previous data document.
It takes the form
//<ddname> DD <term>,<options list>
where
<ddname> identifies the document
<options list> specifies the transformations required on the data document, see 4.2.3.1
<term> = * specifies that the first line starting with // terminates the document.
<term> = BINARY The terminator is to be the same pattern as that on the line following this statement.

The Data Delimiter statement marks the end of a data document, taking two forms corresponding to the two cases of term described above. The Job Delimiter terminates the job document and takes the form
//
on a line to itself. Note that this statement will have no effect in BINARY input mode. It is never included in the job document.

The DEMON, on reading an input document, separates it into a file containing the JCL statements - the job document - and a file for each Data document, these files being made available to the User as described in section 2.

4.2.2.1 Card Reader Input

In normal character mode, cards are read and converted to ISO code. Cards containing columns which do not correspond to legal EBCDIC codes cannot be read in this mode, and are rejected. Columns corresponding to EBCDIC codes with no ISO equivalent are translated into the SUB character (X'1A').

Within the job document, columns 73-80 are discarded and trailing spaces are deleted. No double quote deletion is done.

Within data documents introduced by * the reading mode depends on the options specified. The default is to do double quote deletion, replacing deleted characters by DEL(127), and to remove trailing spaces. The information is then packed in free format separated by LF(NL) characters. The following options are valid:

NOIDENT	Ignore columns 73-80 (the identifier field)
QUOTES	Do not do double quote deletion
TRAIL	Do not delete trailing spaces

Data read in BINARY mode is read as follows. Each card image consists of 80 short integers. The high order bits of each short integer are set to zero, and bits 4-15 contain card rows 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The card images are packed in fixed format, and no LF(NL) characters are inserted. The file appears to the subsystem as a fixed format sequential file.

4.2.2.2 Paper Tape Reader Input

In the normal character mode, null characters and deletes are ignored, and parity errors are transformed into X'1A'. Spaces and carriage returns preceding line feed are ignored.

In binary mode, there is no parity checking, translation or deletion. The reader hardware ignores runout at the beginning of paper tapes, but not at the end. The termination sequence is defined as LF<PATTERN>LF where <PATTERN> is that on the line following the relevant

```
//ddname  DD  BINARY
```

statement. Neither the initial <PATTERN>LF nor the terminal LF<PATTERN>LF appears in the data document. <PATTERN> must be less than 254 bytes. Paper tape binary input files appear to the subsystem to be fixed format sequential files, with a record length of 80 bytes.

4.2.3 Output

An output request to the DEMON specifies the file concerned, the device and any options required.

The DEMON checks the validity of the request and queues it until a device is available. The User may further specify to the DEMON whether the file concerned is to be destroyed on completion of the request, is to be copied and ownership of the copy transferred to the DEMON or whether the DEMON is to be allowed shared access to the file (1.6). The first facility is suitable for temporary files, such as listings, whilst the second would normally be used for permanent files. The third method avoids the overhead of copying the file, but restricts the User's access until output is completed.

All output requests reference the delivery information for the User (DELIVER command) and use this string of up to 19 characters to identify the output. The date and time of receipt of a file by the DEMON is also noted on all output and additional information is added to line printer output giving the time and date at which it was printed.

4.2.3.1 Lineprinter Output

Maximum file size = 16x2**16 bytes = 1,048,576 characters

Code : ISO

Carriage Control Characters: LF = 10 newline
CR = 13 specifies overprinting
FF = 12 throws a new page

Lines longer than 132 characters are split and continued on the following line. CR is ignored immediately following a control character and also if it precedes LF. FF has the same effect as LF followed by FF

The mapping from internal codes is shown in Appendix Table 3. One option is available which allows internal code 35 to be printed as either # or as £ .

4.2.3.2 Card Punch Output

Maximum file size = 3x2**16 bytes = 196,608 bytes (2500 cards)

Code : ISO

Only LF is regarded as a line delimiter. A CR preceding LF is ignored as are CR and FF after LF. Lines exceeding 80 characters are continued on the next card.

The optional mapping of internal code 35 is as described in 4.2.2.1.

Code : BINARY

Column binary cards are punched according to the file format described in 4.2.1.

4.2.3.2 Paper Tape Output

Maximum file size = 3x2**16 bytes = 196,608 bytes.

Code : ISO

Only LF is recognised as a line delimiter and is replaced by a CR/LF pair. A CR/LF pair is treated as LF and CR after LF is ignored.

There is no maximum line length. Even parity tape is generated, codes 127 being converted to X'1A' (SUB).

Code : BINARY

The file is punched as the sequence of bytes given.

5.0 Accounting

EMAS measures the use made of resources on the machine in a variety of ways and these are available for use both in scheduling the work on the System and for accounting purposes.
The accounting currently in use takes note of use made of the central processor, the core store, the spooling system and the file storage.

5.1 Foreground Accounting

During a foreground session, working from a console, an updated accounts record is generated every 20 minutes of elapsed time. The current formula is:

charge(p) = K*R*(t+pt/256)

where K = 2

R = Rate, which is a function of time of day.

Currently:	0000	-	0800	R=0.6
	0800	-	1000	R=0.8
	1000	-	1800	R=1.0
	1800	-	2400	R=0.8

t = cpu time in seconds

pt = number of page turns (a measure of core use)

A call on METER will give the current charge and rate. A final record is generated on logging off.
In the event of a system crash, on average, the previous 10 minutes will not be charged for.

5.2 Spooling Charges

A separate accounting record is generated for each document which the slow device demons process (cards input, line printer output etc.)

The current formula is :

charge (p) = u/128

where u = no of unit records (cards, lines, etc)

5.3 Background Charges

A background job which accesses permanent files is accounted in the same way as a foreground job (5.1), the user however having some control over the rate, R, by means of parameters to the DETACH command.
Two options are currently available:
An immediate request, which asks for the job to be run as soon as possible, whatever the rate and a deferred request which will ensure that the rate is not greater than 0.8

A background job which accesses no permanent files (e.g. cards in, printer out) generates a single accounting record at the end of the job this record including any spooling charges incurred. The rate will be that current at the time of generating the record.

5.4 File Storage

An accounting record is generated daily for use made of file storage on the fixed disk. The unit of space allocation is 1 page and the charge is for units of 'page-days'.
The current charges are:
Archived storage 0.15p/page-day
Semi-Permanent storage 0.075p/page-day,
(See Section 3).

References and Appendices

1. Edinburgh Multi Access System Reference Manual
   Edited by H. Whitfield — September 1971
2. Edinburgh Multi Access System - Subsystem Reference Manual
   Edited by G. Millard — In Preparation
3. Edinburgh IMP Language Manual
   Edited by A. McKendrick — July 1970
4. Edinburgh FORTRAN Language Manual
   Edited by G. Millard — August 1970

Appendices - Input Output Mappings, see Ref. 3.
HELP Information - type HELP(.LP)