EMAS 2900 SUBSYSTEM WORKING NOTE

No. 1
Date 5/4/77

1.1 Introduction

This is the first in a series of notes which will describe features of the initial subsystem being implemented on EMAS 2900 There is a need to satisfy two conflictin requirments.

(1) To provide a stable base for Subsystem users - including VOLUMES and JOBBER

(2) To retain freedom to alter the specification in detail.

As a compromise I propose to provide with each routine specification an indication of its likely stability. In the following notes it can be assumed that EMAS rules apply unless otherwise indicated.

1.2 File names

Initially I propose that we stick to EMAS1 file name limit. These are:-

User (6) . file (8) _ member (8)

This requires 24 characters. For various reasons it may be desirable to extend these fields so a compromise length of 31 is suggested for all filename parameters. In fact the 2900 IMP parameter passing mechanism does not put strings on the stack it puts a descriptor there, so precision is not as important as on current EMAS.

1.3 Subsystem structure

There will be a small kernel of routines and tables to organise the virtual memory and provide minimal character output - probably only one stream. This will be used by the SPOOLER and JOBBER and later by the rest of the Subsystem. My initial aim is to get this kernel working reliably. The routines provided in it will be very similar to their counterparts on System 4.

1.4 Basic File manipulation

The processes of creating, destroying, connecting and disconnecting are much as on EMAS 1. Director does not yet support changes to connect mode,nor changing size of connected files. These activities will be provided by the subsystem using disconnect and connect. Where necessary the re-connection will be back at the same address. The first version of the kernel will contain the following file manipulation routins:

CONNECT
DESTROY
DISCONNECT
FILELIST
FINFO
OUTFILE

1.4.1 CONNECT

Specification:

%SYSTEMROUTINE CONNECT (%STRING(31)FILE, %INTEGER MODE, HOLE,
PROTECT, %RECORDNAME R, %INTEGERNEME FLAG)

MODE= 1 for write 0 for read

HOLE= connect hole - in bytes - default is current size of file

PROTECT= 0 normally

R has following format

%RECORDFORMAT RF (%INTEGER CONAD, FILETYPE, DATASTART, DATAEND)

FLAG values will be supplied later.

Note that this routine can also be used to connect a member of a partitioned file.

Stability - changes of detail likely.

1.4.2 DESTROY

Specification:

%SYSTEMROUTINE DESTROY (%STRING(31)FILE, %INTEGERNAME FLAG)

File will be disconnected (if necessary) and destroyed. Can also be used to destroy a member of a partitioned file.

Stability - unlikely to change.

1.4.3. DISCONNECT

Specification:

%SYSTEMROUTINE DISCONNECT (%STRING(31)FILE, %INTEGERNAME FLAG)

Use obvious. Not applicable to member of partitioned file.

Stability - unlikely to change.

1.4.4 FILELIST

Specification:

%SYSTEMROUTINE FILELIST (%INTEGER MODE, AD1, AD2)

Get list of files in format according to mode. AD1 and AD2 used to provide addresses of areas in users program for information. Initially:

MODE= 1

AD1 points to start of %STRING (8) %ARRAY

AD2 points to an integer into which is put the number of files.

Stability - significant changes and enhancement intended.

1.4.5 FINFO

Specification:

```
%SYSTEMROUTINE  FINFO (%STRING(31) FILE, %INTEGER MODE,
    %RECORDNAME  R,  %INTEGERNAME FLAG)
```

R has following format:

```
%RECORDFORMAT RF (%INTEGER CONAD, FILETYPE, DATASTART, DATAEND, SIZE, RUP, EEP,
MODE, USERS, ARCH, %STRING(6) TRAN, %STRING(8) DATE, TIME  %INTEGER COUNT, SPARE1,
SPARE2)
```

For MODE= 0 the following fields will be filled:

SIZE, RUP, EEP, MODE, USERS, ARCH, TRAN

Stability - significant changes and enhancements intended.

1.4.6 OUTFILE

Specification:

```
%SYSTEMROUTINE  OUTFILE (%STRING(31)S, %INTEGER SIZE, HOLE, PROTECTION,
%INTEGERNAME CONAD, FLAG)
```

File will be created with header containing default values: 32, 32, sized, 0,0,0,0,0
(see 1.5 - below) The rest of the file will be cleared

Stability - stable apart from minor details.

1.5 File Format

1.5.1 The file header format will be as follows:

Length= 32 bytes = 8 words used as follows:

Word	
0	Total length of header and data
1	Length of header (default 32)
2	Physical size
3	Type = 0 non-standard 1 object 3 character 4 data file (discrete records) 5 partitioned
4	Data last altered (packed)
5	Time last altered (packed)
6	Type dependent
7	Type dependent

1.6 Ancilliary routines

1.6.1 MOVE

Specification:

```
%SYSTEMROUTINE  MOVE (%INTEGER BYTES, FROM, TO)
```

Move 'BYTES' bytes from address 'FROM' to address 'TO'

Stability - stable

1.6.2 FILL

Specification:

%SYSTEMROUTINE FILL (%INTEGER BYTES, START, PATTERN)

The least significant byte in 'PATTERN' is copied into the 'BYTES' bytes starting at 'START'

Stability: stable

1.6.3 DUMP

Specification:

%SYSTEMROUTINE DUMP (%INTEGER FROM, TO)

The area 'FROM' to 'TO' is dumped on the current output stream.

Stability stable

1.6.4 PHEX

Specification

%SYSTEMROUTINE PHEX (%INTEGER N)

The value of N is printed as 8 hex. digits

Stability stable

1.6.5 PON, POFF, OUT

These all take one parameter of type %RECORDNAME with format:

%INTEGER DEST, SRCE, A, B, C, D, E, F

1.7 Character output

The initial version of Kernel will allow for character output to a single steam - the line printer. Output will be put in a file which can be listed an any time by calling systemroutine CLOSEST. The effect of this routine will be to list the current output file and reset the output pointer to the start of the file for further output. (c.f. System 4 VOLUMES)

Roderick McLeod

Existing and proposed developments

2.1 Introduction

This note, which follows 3 months of development of the Subsystem, contains details of additional facilities and some discussion of proposed developments. One major change has occurred since the first note: VOLUMS is no longer based on the standard Subsystem. Whilst this potentially increases the amount of software to be maintained it significantly simplifies the Subsystem in that it no longer has to be completely general in the file handling area.

Currently Alan Anderson is working on mounting the Scientific Jobber on EMAS and this will use, initially at least, a kernel of routines from the standard Subsystem. The future of this development depends on some difficult decisions about compatibility between foreground and batch access.

2.2 Filenames

A "final" decision has been reached concerning the length of filenames. The new form is

user(6).file(11)_member(11)

Although I have reservations about this decision I accept that it has its good points - such as the fact that the maximum length of a filename will fit into the string (31) parameters I proposed for filenames in my last note. All existing Subsystem software has been adjusted to allow for this change.

2.3 Foreground commands

Currently the following foreground commands are implemented:

```
ACCEPT
ALGOL
COPY        (as COPYFILE)
CPULIMIT
DEFINE      (first parameter is channel no. - e.g. DEFINE 1,.LP)
DESTROY
DISCONNECT
EDIT
FILES       (as FLIST)
FORTRAN     (as FORTE)
IMP
LINK
LIST
LOOK
METER
OBEY        (as OBEYFILE)
NEWGEN
```

OFFER
PARM
PERMIT (as PERMITFILE)
RENAME
RUN
STOP

In general commands operate as on System 4 EMAS. There are a few restrictions and differences - details are available from me. Many of the hash commands are also available - details later.

2.4 Data handling

2.4.1 Character Files

Character I/O for IMP and ALGOL has been implemented using a completely rewritten IOCP. Its main differences are as follows:

* no line reconstruction - hence no trailing space deletion nor detection of SUB character.

* no margins - the compilers currently do their own input framing

* stream selection no longer works on a record basis - it can now be done between individual characters

2.4.2 Data Files

Currently I am working on this. On its own it is reasonably straightforward. The Fortran I/O package from Geoff's group has fairly well-defined interfaces. The snags concern mixed access to the same file or device - e.g. the input teletype being used both to read commands (the command interpeter being written in IMP of course) and for input to a Fortran program. The switch between the two is not easy to organise in an elegant way.

2.5 Compatibility.

I am constantly considering the implications of compatibility with System 4 EMAS. Although there is the possibility of external use of EMAS 2900, my primary interest is still in its use by our current EMAS fraternity. I am quite sure that the quest for compatibility must not be allowed to hamper development at this time but there are obvious advantages in allowing a straightforward transfer of work from the System 4 to the 2900. There are two parts to the compatibility problem:

* Changes to foreground commands affect everyone but are probably fairly easy to get used to.

* Changes to commands called from programs mean alterations to existing working programs. This will involve significant work for people transferring such programs from System 4 to 2900.

2.5.1 Compatibility of files

Currently the position is as follows:

Type	Compatibility
Character	Vital - no great problem.
Data	V format files will not be compatible - a conversion utility should be straightforward. F format files OK.
Object Files	Not needed.
N.R. Object Files	Currently compatible but ought to change to conform to standard file header format.
Library Index files	Not compatible.
Partitioned files	Almost compatible - existing directory format has room for 11 character member names. Header format not quite right.

2.5.2 Compatibility - other features

One difficult one is .TT. Most of us would find great difficulty in defining our interactive terminals other than as .TT I suspect. Nevertheless it is an outdated abbreviation. One possibility is something like .IT for 'interactive terminal' - but currently I propose to introduce the pseudo devices .IN and .OUT. These would direct input and output to the current main input and output devices. During foreground sessions this would normally be the interactive terminal - unless an OBEY command was being executed.

2.6 Conclusion

As always I welcome comments on the above ideas.

Roderick McLeod

EMAS 2900 SUBSYSTEM NOTE

No: 3 (revised)
Date: 11/5/79

User process information

The SSINFO facility provided on the System 4 has proved to be a clumsy solution to the problem of providing information about a user's process, in that the user has to provide a long record format even when he only requires one minor item. The equivalent information on the 2900 is provided by a pair of functions: a string function for textual information, and an integer function for integer information.

They have to be specified:

```
externalstringfnspec    UINFS (integer ENTRY)

externalintegerfnspec   UINFI (integer ENTRY)
```

The parameter ENTRY in each case specifies which piece of information is required. If a value is given which is out of the current range then the result will be a null string or zero, as appropriate. Currently the following values of ENTRY can be used:

UINFS

1	User name
2	Delivery information
3	Session start time hh.mm.ss
4	PROMPT text
5	Active directory
6	Subsystem version

UINFI

1	File system number
2	Mode: result = 1 implies foreground result = 2 implies background result = 3 implies foreground OBEY
3	Users currently logged on
4	ACR level
5	Current CPU limit (seconds)
6	Max size in Kilobytes for DEFINE
7	SYNC1DEST
8	SYNC2DEST
9	ASYNCDEST
10	Address of Director USERINF record
11	Process number
12	DEFINE level
13	Process incarnation number
14	AUXSTACK size (Kilobytes)

Further entries will be added as required.

R.R. McLeod

New Facilities

Date and Time

I have had a few comments about my proposal (System Note 5) for packed date and time. Apart from treating bit 2**31 as the most significant bit of 'year' (hence giving a range 1970-2033), I propose using the format in the Subsystem and will be changing to it during the next few weeks. FILEANAL will only give the time and date for recently altered files.

System Routine ANALYSE

To enable users to more readily write their own file analysis routines I am providing a new general-purpose systemroutine ANALYSE. This will return information in a recordarray for the user to analyse. It will be useful for Object, Directory and Partitioned files in particular. The standard command FILEANAL will call this new routine. Full details later.

New Command ALIAS

This new command allows you to add an alias to a procedure name. The format is

```
ALIAS(procname,alias)
```

Hence if you want to access the command (procedure) IMP using the alias I, you should type

```
ALIAS(IMP,I)
```

To remove all the aliases of a given name omit the second parameter; e.g.

```
ALIAS(IMP)
```

String Function INTERRUPT

A new externalstringfn INTERRUPT is available. It replaces TESTINT. It has no parameter, and returns and clears the latest interrupt of more than one character from the user terminal. A null string is returned if no multi-character interrupt has occurred or if the last one has already been returned by INTERRUPT. Note that multi-character interrupts are not queued. Single-character interrupts are reserved for Subsystem use.

Example:

```
INT = INTERRUPT
if INT = "STOP" then return
if INT = "RESET" then .........
```

Directory Files and Related Commands

Directory files serve a similar role to library index files on the 4-75. Their functions are:

* For each procedure entry name or data entry name in a directory, to provide a pointer, either to the name of the object file that contains the entry, or to another directory (from which the name of the appropriate object file can be found, directly or indirectly).

* To hold the ALIAS information (see above).

NEWDIRECTORY

This command is used to create a new directory file. Only one parameter is required, the name of the directory. Currently this must be 'SSWDIR' since the other commands only operate with respect to this directory; later it will be possible to create other directories. Two further parameters can be used to specify the size of the directory - see Mac for details.

INSERT

This command is similar to INSERTFILE on the 4-75. Note however that it has to be called explicitly once for each object file. Once it has been called for an object file, that object file will be REMOVEd from and INSERTed into the currently selected directory every time it is compiled into. If a compilation into the file fails, only the object file name will remain "inserted", but all the information will be put back when it is successfully recompiled.

INSERT has only one parameter - the name of the object file.

It is not necessary to INSERT main program object files.

REMOVE

This command is the equivalent of REMOVEFILE on the 4-75. Only one object file can be removed at a time.

Program Loading

The program loader can now handle external data linking. Note that a check is enforced on the lengths of external and extrinsic items. The order of satisfying references differs from that on the 4-75. The change has been made in the interests of efficiency, and does currently restrict the generality compared with the 4-75. If necessary I shall provide additional facilities to enable users to override standard command names. The mechanism works as follows:

* There is a session directory of all currently loaded entries; at process start-up this includes only the standard commands which are in the basefile.

* If a standard command is accessed which is outwith the basefile, e.g. one of the Subsystem editors or a compiler, then the necessary files are loaded for the rest of the session and their entry information is added to the session directory.

* If a user-written command is accessed, it is loaded, and then unloaded on return to command level. A facility will be provided to nominate user object files which are to be loaded for the whole session.

* When searching for a procedure name the loader first searches the session directory, then the user's own SSHDIR (if it exists), then the Subsystem base directory. There are two implications:

  * it is not possible to override commands in the basefile

  * it is not possible to override other Subsystem commands if they have been accessed earlier in the session

For example, if you wish to use a new IMP compiler you must insert it in your directory. Even so you will not be able to access it if you have already used the standard IMP compiler during the session. If you log off and then on again you will obtain access to the new compiler.

Current Developments

1. Extensions and improvements to FILEANAL.

2. Provision of an OPTION command. This will incorporate APPENDLIB.

3. Provision of full I/O facilities for IMP and FORTRAN.

Roderick McLeod

Edinburgh Subsystem Standard File Formats

Introduction

This note supersedes all earlier descriptions of file headers. It contains information about the contents of headers of files produced by the Edinburgh Subsystem. It also contains some comments on backward compatibility with 4-75 files. Object files still have a format which does not fit in with other types. I am awaiting changes to LPUT from Geoff Millard, but I have to warn that there will be considerable upheaval when the change is made.

Information common to all files

The first six integers are used as follows:

1. Useful length of file: this is the sum of the length of the header and the length of the data (bytes).

2. Length of header (bytes).

3. Physical size of file: this will normally be the total size of the minimum number of full pages needed to hold the useful length (bytes).

4. File type: 1 - Object file (eventually)
   2 - Directory file
   3 - Character file
   4 - Data file (discrete records)
   5 - Partitioned file

   These are also the values returned by CONNECT.

5. This word is reserved for an optional sum-check. The facility of sum-checking vital files on the 4-75 (e.g. compilers) has proved to be very useful. A similar but standardised scheme is envisaged here; further details later. Note that in the case of fixed-up Director and Subsystem files this word may have to be used for a PC relative jump, since the Local Controller and Director respectively cannot readily read an offset from file; so always set the PC to 16 bytes from the start of the file. There is no obvious resolution to this.

6. Packed date and time of last writing to the file. This is updated whenever the file is connected by the Subsystem in write mode. The format is described in EMAS System Note 5.

Integers 7 and 8 are used in type-specific ways, as follows:

Type	7	8
Directory	Offset of P list (Note a).	Spare
Character	Spare	Spare
Data	Record format and maximum or fixed record size (Note b).	No. of records
Partitioned	Offset of directory of members (Note c).	No. of members

Notes

a) The internal format of a directory file will be published later. Briefly a directory comprises two parts: a hashed table of entry names and aliases, each of which points to a name in the "P list". Additionally, for long names in the hashed table the first part of the name goes in the table, together with a pointer to the rest of the name in the P list.

b) The format of this integer is

```
(R<<16)!(F)
```

where R is the maximum or fixed record length and F is 1 (fixed), 2 (variable) or 3 (unformatted). An unformatted file is the equivalent of an SM file on the System 4.

c) The directory of members is an array of records with the following format:

```
recordformat PDF (integer START, string (11) NAME, c
                  integer HOLE, S1, S2, S3)
```

Backward compatibility to 4-75

In the short term files will be transferred between the 4-75 and 2900 via controlled routes, and utility programs can be provided on each machine to carry out the necessary conversions. Eventually it is proposed to have a common format for archive and backup tapes between the systems. The desirable aim of maintaining identical file formats between the two Subsystems has been sacrificed in the interests of a tidier situation on the new Subsystem. In detail, only three types of file can sensibly be transferred:

Character	Minor change: dual standard could be accommodated in both Subsystems.
Data	Significant changes: will require conversion program, which will only be made available on the 2970.
Partitioned	Minor change: will require a conversion program; also a problem of individual members having the "wrong" format; also a backward transfer problem in that member names of up to 11 characters are allowed on the 2970, while only 8 characters are allowed on the 4-75.

Summary

Even with a common format for System archive and backup tapes the ready transfer of files between Systems, other than character files, will not be achieved.

Roderick McLeod

Diagnostic Aids in the Edinburgh Subsystem

The following commands are available for diagnostic purposes; in the main they replicate facilities on the 4-75. Note that:

* The fact that they are described here should not be taken as a guarantee of their long term support.

* In all relevant cases lengths are expressed in bytes.

Command structure

Each command has a # as its first character. Note that hash commands do not invoke the Loader and there is no one-to-one correspondence between each hash command and an external routine. Therefore hash commands cannot be called from programs.

The parameters to hash commands include numeric constants. These can be typed as decimal integers in the normal way, or as hexadecimal numbers, in which case they should be preceded by an 'X'. For example, the following two commands would have the same effect:

```
#SWORD(X840000,256)

#SWORD(X840000,X100)
```

Individual command descriptions

The hash commands available are as follows:

#ACR

prints out the current ACR level.

#CONNECT(filename)

connects the file, if possible in write mode, otherwise in read mode. Prints the connect address. Can also be used for a member of a partitioned data set, but the member is always connected in READ mode. Since a file connected by #CONNECT remains connected on return to command level, it is advisable to disconnect it explicitly when any operations upon it have been completed.

#DEC(hex value)

converts the hexadecimal value to decimal.

#DUMP(address, bytes)

dumps the specified area to the line printer.

#DUMPFILE(filename, offset of start, bytes)

dumps the specified area of the file to the line printer.

#HEX(decimal value)

converts the decimal value to hexadecimal.

#PCOM(integer)

prints out the value of the specified location in COMREG.

#PMESS(integer)

prints out the Subsystem error message associated with the specified fault number.

#PVM

prints table of connected files.

#QUIT

logs off even when the session directory is corrupted.

#REGS

prints out registers at the time of the most recent failure contingency. Note that information on the last four contingencies is held; the three previous ones can be obtained using #REGS(-1), #REGS(-2) and #REGS(-3).

#SBYTE(address,value)

sets the specified byte to value. Clearly the byte must be accessible in write mode. This can be achieved by connecting the file using #CONNECT.

#SCOM(integer,value)

sets the specified location in COMREG to value.

#SETBASE(filename)

sets the name of the basefile to be used for subsequent sessions. If the parameter is omitted the default Subsystem basefile will be used.

#SNAP(address,bytes)

dumps the specified area on .OUT. The amount printed on a line will depend on the current setting of OPTION ITWIDTH.

#SNAPCODE(address,bytes,out)

decompiles code and prints it on the device or file specified. Default is .OUT.

#SNAPCH(address,bytes)

prints the specified area as ISO characters on .OUT.

#SSTRING(address,"string")

sets the string at the address specified to the string given. Note that the string should be enclosed in double quote characters.

#SWORD(address,value)

sets the word at the address specified to value.

Additional commands will be added in the light of experience.

Roderick McLeod

EMAS 2900 SUBSYSTEM NOTE

No:8
Date:20/3/79

Dynamic Loading

This note describes the initial dynamic loading facilities in the Edinburgh Subsystem. For the user of IMP and ALGOL they are very straightforward and operate much as on the 4-75. For the FORTRAN user there is a complication because of the way in which the FORTRAN compiler uses the stack.

Compiling

It is important to appreciate that for a routine to be loaded dynamically it is not necessary to modify the routine in any way. It is the program or procedure that references the routine that must be marked to inform the loader that the reference should be satisfied dynamically. Eventually this will be done at compile time, thus:

ALGOL+FORTRAN: compile with PARM(DYNAMIC) - all references are marked for satisfying dynamically.

IMP: use %DYNAMICROUTINESPEC in place of %EXTERNALROUTINESPEC for all routines and procedures which are to be loaded dynamically. It will also be possible to use PARM(DYNAMIC), as for ALGOL and FORTRAN.

Current compilers do not recognise the dynamic requests, so for a short period it will be necessary to proceed as follows:

Compile the file with any required PARMs.
Mark the file by use of the command DYNAMIC(object file).

To check the references in an object file, use

```
ANALYSE(objectfile, R)
```

Running the program

For IMP and ALGOL there is no further special action required. The program is run or command called just as at present. When the file containing the dynamic references is loaded any references to procedures already loaded are satisfied immediately. All others are left until the first call of the procedure. At this point a jump is made to the loader and an attempt made to load the required procedure. If this fails then messages of the form

```
Failed to load RESET dynamically RESET not found
```

are output, followed by diagnostics and program termination. If it is successful then the reference is filled in the programs GLA (general linkage area) and the routine is called. Subsequent calls behave exactly as for calls to statically loaded routines.

FORTRAN

Where the routines being loaded dynamically include FORTRAN code there is an additional requirement. FORTRAN uses part of the User Stack for its variables, but unlike IMP and ALGOL this space is allocated all the time the file is loaded - not just during the execution of the routine. This so called INITIALISED STACK must be put at the bottom of the user stack, and for this to be possible a space must be allocated. With dynamic loading the loader cannot allocate space because, until execution begins and the additional routines are loaded, it has no information about how much space to leave The method therefore is for the usser to specify how much space is to be allocated for the FORTRAN initialised stack.

This number can be calculated by using the 'S' option in ANALYSE in respect of all the FORTRAN files that are to be loaded in any one RUN. The output includes a table which contains a line of the form

```
ISTK 000038EE 00002AB0
```

The second of these two numbers gives, in hexadecimal, the length of initialised stack for this object file. By adding all of these it is possible to get an accurate figure. This figure should then be specified as the size of the initialised stack using

```
OPTION(INITSTACKSIZE=n)
```

Note that the parameter is in Kilobytes, and that the number specified must be at least 32 (Kilobytes) less than the size of the USERSTACK.

As a simple starting point the following values will probably serve the majority of users:

```
OPTION (USERSTACKSIZE=128)

OPTION (INITSTACKSIZE=60)
```

For those running very large FORTRAN programs or packages, the maximum settings allowed are:

```
OPTION (USERSTACKSIZE=252)

OPTION (INITSTACKSIZE=100)
```

Conclusion

It is hoped that these facilities will prove useful. They may be modified slightly in the light of experience.

R.R. McLeod

Edinburgh 2900 Compilers: Object File Format

This Note describes the format of the object files created by ERCC 2900 compilers using LPUT. It is an intermediate format in the sense that ICL standard object files will be generated from this form, which will not necessarily reside in a permanent file, other than on EMAS. This format is, however, appropriate to the provision of 'fast loaders' in a conventional ICL file system or when object files can be directly mapped into the virtual memory.

The object file may contain up to seven areas in addition to red tape and linkage information, laid out in a contiguous area as below

	area code
standard file header	
code	1
sst	4
plt	3
gla	2
ust	5
initcmn	6
initstack	7
linkage data	
object file map	

The standard file header consists of eight words defined for use on EMAS, and which will be retained on all systems to assist mobility of object code and utilities. These words are used as follows:

word 0 offset of end of data from start of file
1 displacement of start of code from start of file (used by EMAS basic system loaders and compatible with current practice)
2 physical size of the file (bytes)
3 1 (file type for 2900 object)
4 sum check (optional)
5 date and time when file was last written to
6 offset of load data from start of file
7 offset of object file map from start of file

The object file map consists of twenty-two words with the format:

(integer N, recordarray R(1:7))

where

N is the number of areas defined

R has the following format

(integer START, L, PROP)

where R(i) describes area i and
START is the displacement of the start of the area from the start of the object file
L is the length of the area
PROP will be used to define properties of the area (zero at present)

The contents of the seven areas are described in Subsystem Note 15.

LDATA

The linkage data contains a fifteen element array, LDATA, which provides links to records, or lists of records, also held in the linkage data area. These records provide information about entries and references, both for procedures and data items, common areas and relocation requirements between areas. All links within the linkage data are byte displacements from the start of the object file. The significance of the entries in the initial block of pointers is given below. The allocation of these entries has been constrained by the requirement to provide a simple transition from 4-75 EMAS object files and, consequently, some entries are not used.

LDATA	content
0	14 (number of significant pointers following)
1	listhead of procedure entries
2	0
3	0
4	listhead of data entries
5	0
6	0
7	listhead of static procedure references
8	listhead of dynamic procedure references

LDATA	content
9	listhead of data references
10	0
11	0
12	pointer to a record defining object file listing (format to be defined)
13	0
14	listhead of blocks of relocation requests

In the following definitions, identifiers may contain up to 31 characters. However, only sufficient space for the actual identifier (rounded to a word boundary) is reserved. All displacements are in bytes from the start of the appropriate areas, which are identified by the area codes defined above. LINK is the byte displacement of the next record in the list, from the start of the object file map, or zero to terminate the list.

Procedure entries - listhead LDATA(1)

The format of each record is

(integer LINK, LOC, string(31) IDEN)

where

LOC is A<<24!EP and

EP is the displacement of the entry descriptor from the start of area A (2 gla, 3 plt)
IDEN is the name of the entry point

The most significant bit of LOC is set if this is a main program.

Data entries - listhead LDATA(4)

The format of each record is

(integer LINK, DISP, L, A, string(31) IDEN)

where

DISP is the displacement of the data area from the start of area A (2 gla, 5 ust, 6 initcmn)
L is the length of the data area
IDEN is the name of the data area

Procedure references - listheads LDATA(7) and LDATA(8)

The format of each record is

(integer LINK, REFLOC, string(31) IDEN)

where

REFLOC is A<<24!REFAD
REFAD is the displacement from the start of area A (2 gla, 3 plt) of a descriptor which is to be filled in by the loader
IDEN is the name of the referenced procedure

Data references - listhead LDATA(9)

The format of each record is

(integer LINK, REFARRAY, L, string(31) IDEN)

where

REFARRAY is the offset from the start of file of a record of the form

(integer N, integerarray REFLOC(1:N))

where each REFLOC(i) contains A<<24!REFAD and

REFAD is the displacement, from the start of the specified area, of a word which is to have the address of data area IDEN added to it by the loader.
A nominates the area containing the reference (2 gla, 3 plt, 7 initstack).
L is the expected length of the data area

If the reference is to a "common" area, i.e. an area to be created by the loader if not defined, the most significant bit of REFARRAY is also set.

Initialisation data - listhead LDATA(13)

The format of each record is

(integer LINK, A, DISP, LEN, REP, ADDR)

where

A is the area to be initialised
DISP is the offset within the area at which initialisation is to start
LEN is the length of the initialisation data
REP is the number of copies of the initialisation data to be copied to consecutive LEN byte areas
ADDR is the displacement from the start of the file of the initialisation data if LEN >1
it is the initialisation value if LEN =1

Relocation request blocks - listhead LDATA(14)

The format of each record is

(integer LINK, N, recordarray R(1:N))

where

N is the number of relocation entries in this block
R has the following format

(integer AREALOC, BASELOC)

where

AREALOC is AREACODE<<24!AREADISP,
BASELOC is BASECODE<<24!BASEDISP

AREACODE identifies the area containing the item to be relocated.
AREADISP is the byte displacement of aa word within the area to be relocated by the load address of the start of the area identified by
BASECODE which may be any of areas 1 - 7
BASEDISP allows relocation by a sub-component of that area

G.E. Millard

Generation of Object Program Files for 2900

systemroutine LPUT (integer TYPE, P1, P2, P3)

This routine generates an object program file from information passed through the parameters in a sequence of calls. The object file name must previously have been assigned by a string addressed by COMREG(52), and a workfile of at least one segment created and addressed by COMREG(14). (The specification of COMREG is systemintegermap COMREG(integer N).)

The generated object file may have up to seven areas, as defined below, in addition to the linkage information generated by LPUT. The content of each of these areas, any of which may be empty, is at the compiler writer's discretion, subject to the comments below.

area

1 code — should ideally contain only executable code and constants accessed only from within the code area, thus enabling a connect mode of execute, shared. As this mode apparently inhibits the most efficient method of access to constant strings and vectors the connect mode, initially at least, will be execute, read, shared. It is anticipated that connection in execute only mode will be considered essential in some instances.

2 gla (general linkage area) — normally contains descriptors for accessing external objects, entry descriptors and static (normally scalar) data. The connect mode is read, write, unshared. The first eight words have a prescribed use (see below).

3 plt (procedure linkage table) — may be used to contain entry and reference descriptors. If a plt exists then all such descriptors must be contained in it, rather than in gla, which may still exist to contain static data. The connect mode is read, unshared. The first eight words have a prescribed use (see below).

4 sst (shareable symbol tables) — is expected to contain information relating to run-time diagnostics and is connected in read, shared mode.

5 ust (unshared symbol tables) — normally contains static arrays and is connected in read, write, unshared mode.

6 initcmn (initialised common areas) — is an accumulation of separately specified and initialised common areas connected in read, write, unshared mode.

7 initstack (static initialised area on stack) — may be used to contain local data and data descriptors. This area should not normally be used for arrays as total stack space is constrained.

The first eight words of the gla, or plt if it exists, are currently defined as follows:

```
word 0      code descriptor to the
     1      first or only entry point
     2      address of ust area
     3      address of sst area
     4      byte 0 language flag 1 IMP
                                 2 FORTE
                                 3 IMPS
                                 4 NASS
                                 5 ALGOL
                                 6 optimised code (no diag tables)
                                 7 PASCAL
                                 8 SIMULA
            byte 1 compiler version
            byte 2 compiler options
            byte 3 (may be language dependent)
     5      reserved for address of gla if plt is used, otherwise 0
     6      reserved
     7      reserved
```

After opening the object file (TYPE=0) fragments for any of the areas may be passed to LPUT in any order. The location for the fragment within the object file is specified relative to the relevant base. If a fragment overlaps a previous fragment then that part of the previous fragment is replaced. Additional calls on LPUT can provide information to enable linkage to other external objects to be achieved and also to specify locations in the non-shareable areas which are to be relocated by the actual value of one of the bases when the object file is loaded. When all the information relating to the current external routine has been passed to LPUT a type 6 call confirms the sizes of 1 to 5 (the sizes of initialised common areas having been separately specified).

Subsequent calls on LPUT will generate a new set of areas. A type 7 call indicates that the file is now complete. The type 6 call may be omitted if only one set of areas is produced.

The generated object file has, in its simplest form, the layout shown below:

```
header
code
sst
[plt]
gla
ust
initcmn
initstack
linkage data
```

A particular compiler may choose to generate only one set of areas even when compiling a number of external routines. Alternatively each external routine may generate a separate set of areas. The layout of an object file with multiple sets of areas is shown in the figure below. (This also represents the layout of the file generated by linking files generated by separate compilations).

```
header
code 1
code 2
sst 1
sst 2
[plt1]
[plt2]
gla 1
gla 2
ust 1
ust 2
initcmn 1
initcmn 2
initstack 1
initstack 2
linkage data
```

No assumptions should be made about the relative position of the areas when a program is loaded. An auxiliary stack (for use by dynamic data structures) may be located by a data reference to 'ICL9CEAUXST'. This is a two word location containing a descriptor to an area with the format below:

```
word 0     current 'free aux stack' pointer (absolute)
     1     reserved
     2     physical end of aux stack
     3     reserved
```

Calls on LPUT

The calls on LPUT are as follows. Any parameters not specified for a particular call should be set to zero to allow for future expansion. All displacements must be in bytes.

TYPE =0 Initialisation call. The object file is opened.
P1 = language flag
P2 = release no.
P3 = version no.

TYPE =1 code area
2 gla area
3 plt area
4 sst area
5 ust area
This set of calls specify that P1 bytes of information (a 'fragment') currently held at address P3 in the calling routine are to be placed at the location P2 bytes from the start of the nominated area, unless P3 < 256. In this case P1 bytes each containing the value P3 are to be set up at the location specified by P2.

TYPE =31 code area
32 gla area
33 plt area
34 sst area
35 ust area
36 initcmn area
37 initstack area

This set of calls act as for 1 - 5 above, but providing for the extended set of areas.

TYPE =41 code area
42 gla area
43 plt area
44 sst area
45 ust area
46 initcmn area
47 initstack area
This set of calls provides for repeated initialisation with up to 255 bytes of data.
P1 = number of bytes of data <<24! number of copies
P2 = displacement from area start at which initialisation is TO commence
P3 = address of the initialising data

TYPE =6 This call indicates that the information relating to the current set of areas is complete.
P1 = 32 (current length of information)
P3 = address of eight word information field containing the lengths of areas 1 to 7 and the sum of these lengths for the preceding set of areas.

TYPE =7 Indicates that the information relating to the current file is complete.
P1 = 32
P3 = similar to type 6, but with sub-totals for each area type and the grand total.

TYPE =10 Define a 'common' area reference
P1, P2 and P3 are as for type 15, substituting 'common' for data.

TYPE =11 Define an entry point.
P1 = code of the area containing the entry descriptor (2 gla, 3 plt). If the most significant bit is set this signifies that the entry point is to a main program unit.
P2 = location of the entry descriptor relative to the start of the area defined by P1.
P3 = address of a string defining the entry point name. This is truncated to the most significant 31 characters.

TYPE =12 Define an external routine reference.
P1 = code of the area containing the reference block (2 gla, 3 plt).
P2 = location of a two-word reference block relative to the start of the area defined by P1. This will be filled, at load time, with a descriptor through which a call may be made to the referenced routine.
P3 = address of a string defining the name of the referenced routine. (31 characters significant).

TYPE =13 Define an external routine reference which is to be satisfied dynamically, i.e. when it is actually called. If dynamic loading is not available this is treated as type 12.
P2 and P3 are as for type 12.

TYPE =14 Define a data entry point.
P1 = (AREA <<24) ! length of the data area, where AREA identifies the area containing the data area (2 gla, 5 ust, 6 initcmn). If AREA is 6 this is an initialised common area.
P2 = location of the data area relative to the start of area AREA.
P3 = address of a string defining the data area name (31 characters significant).

TYPE =15 Define a data area reference.
P1 = (AREA CODE <<24)! minimum length the data area is expected to have, where AREA CODE identifies the area containing the reference (2 gla, 3 plt, 7 initstack).
P2 = location of a word relative to the start of the area defined by AREA CODE which will have the address of the data area added to it when the object file is loaded.
P3 = address of a string giving the name of the referenced data area.

TYPE =17 Define a data entry point in unshared symbol tables (area 5).
P1 = length of the data area.
P2 = locations of the data area relative to the start of gla symbol tables.
P3 = address of a string (maximum 31 characters) defining the data area name.

TYPE =18 Relocate an 18 ~~last~~ bit address in code (area 1).
P1 = 0
P2 = displacement from the start of the code area of a 4 byte instruction.
P3 = 18 bit address to be added to the last 18 bits of the instruction located by P2.

TYPE =19 Request relocation of a single word.
P1 = code for area containing the reference:
2 gla
3 plt
5 ust
7 initstack
P2 = byte displacement within area P1 of a word to be relocated at load time by the base of the area specified by P3.
P3 = code for area base:
1 code
2 gla
3 plt
4 sst
5 ust
7 initstack

G.E. Millard

No: 16
Date: 31/8/79

Use of COMREG for Storing PARM Options

The PARM options used by the compilers, by LINK and by the Loader are held in elements 27 and 28 of COMREG. COMREG must be specified if access is required to it:

systemintegermapspec COMREG(integer N)

The table below associates the pairs of PARM options with the bits of COMREG(27) and COMREG(28). The effects of these PARMs are described in Subsystem Note 7.

COMREG(27)

Bit n (2**n)	Set (Default underlined)	Unset
31	n/a	
30	JOBBER MODE	NORMAL MODE
	(set by Jobber)	
29	MISMATCH	NOMISMATCH
28	PARMX	NOPARMX
27	PARMY	NOPARMY
26	PARMZ	NOPARMZ
25	spare	
24	Stack size set	
23	NOLINE	LINE
22	EBCDIC	ISO
21	Diag stream set	
20	DYNAMIC	STATIC
19	FREE	FIXED
18	DEBUG	NODEBUG
17	MAP	NOMAP
16	OPT	NOOPT
15	ATTR	NOATTR
14	CODE	NOCODE
13	LET	NOLET
12	LABELS	NOLABELS
11	XREF	NOXREF
10	ZERO	NOZERO
9	INHIBIOF	ALLOWIOF
8	IMPS	IMP
	(compiler command)	
7	PROFILE	NOPROFILE
6	NOTRACE	TRACE
5	NOARRAY	ARRAY
4	NOCHECK	CHECK
3	STACK	NOSTACK
2	NODIAG	DIAG
1	NOLIST	LIST
0	QUOTES	PERCENT

COMREG(28)

Bit n (2**n)	Set (Default underlined)	Unset
31-16	n/a	
15	MINSTACK	NORMALSTACK
14	(May be used for more 'stack' options)	
13-9	n/a	
3	R8	R4
2	L8	L4
1	I8	I4
0	n/a	

J.M. Murison

Modifying Object Files on EMAS 2900

This note describes a program which performs a variety of operations on object files. It is likely to be useful where large object files with many entries and references are handled.

The following types of operation are available:

- changing the names of procedure or data entries and references.
- suppressing procedure and data entries.
- changing static procedure references to dynamic and vice versa.
- merging the code and shareable symbol tables areas or the GLA and unshared symbol tables areas.
- binding the object file.

The BIND operation is intended to alleviate the problem of lengthy loading times for large packages. Essentially, it processes an object file to produce a module which can subsequently be loaded at minimal cost. To do this it assumes fixed sites for the code, GLA and stack, and hence can satisfy all relocation requests. It also resolves all cross-references in the object file.

The program is accessed by:

```
Command:OPTION(SEARCHDIR=CONLIB.GENERAL)
```

and called by:

```
Command:MODIFY(object file [,output object file] [,report file])
```

24/4/81 | The input (and output) object file may be a pd file member.

If the second parameter is omitted, the modified object file overwrites the input object file.

If the third parameter is omitted, the report, which contains details of the operations performed, is sent to T#MODLIST. Fault messages are sent to the terminal and do not appear in the report.

The program issues the prompt "Operation:", the replies to which are given below. Most operations require further input and issue appropriate prompts.

The "Operation:" prompt is reissued until the reply "CLOSE" is given. This terminates the modification and causes the output file to be generated.

All input may be given in upper or lower case; spaces are not significant.

The parameters for the first five operations consist of one or more lines, each containing a pair of names separated by a comma, the list being terminated by .END, e.g.

```
Operation:RENAME
Proc ent pair:DREAD, READDATA
Proc ent pair:DPRINT, PRINTDATA
Proc ent pair:.END
Operation:
```

1. RENAME

   Changes the names of procedure entries.

   Parameters: procedure entry, newname

2. REDIRECT

   Changes the names of procedure references.

   Parameters: procedure reference, newname

3. RENAME DATA

   Changes the names of data entries.

   Parameters: data entry, newname

4. REDIRECT DATA

   Changes the names of data references.

   Parameters: data reference, newname

5. ALIAS

   Allows a copy of a procedure entry to be made and given a different name.

   Parameters: procedure entry, copy name

The parameters for operations 7-13 below consist of a list of items separated by commas, or the keyword .ALL. The list is terminated by newline unless the last character is a comma, in which case it continues on the following line. The remaining operations (14-17) take no parameters.

Examples:

```
Operation:MAKEDYNAMIC
Proc ref list: A1,A2,A3,
Proc ref list: A4
Operation:
```

```
Operation:SUPPRESS DATA
Proc ent list:.ALL
Operation:BIND
Operation:
```

6. MAKE DYNAMIC

   Makes static procedure references dynamic.

   Parameters: static procedure reference list or .ALL

7. MAKE STATIC

Makes dynamic procedure references static.

Parameters: dynamic procedure reference list or .ALL

8. SUPPRESS

This operation allows procedure entries to be suppressed. Other operations such as SATISFY REFS will still be able to find and use a suppressed entry - the only effect is that the suppressed entry will not appear in the load data for the output object file. Note that a main program entry cannot be suppressed.

Parameters: procedure entry list or .ALL

9. RETAIN

This cancels a SUPPRESS request. Since by default all entries are retained, this operation is provided so that SUPPRESS with parameter .ALL can be followed by explicit RETAIN calls for the procedure entries which are actually required. Note that an object file must have at least one main entry, procedure entry or data entry.

Parameters: procedure entry list or .ALL

10. SUPPRESS DATA

This command allows data entries to be suppressed (see SUPPRESS).

Parameters: data entry list or .ALL

11. RETAIN DATA

This cancels a SUPPRESS DATA request (see RETAIN).

Parameters: data entry list or .ALL

12. SATISFY REFS

For each procedure reference given, a search is made of the list of procedure entries. If a matching entry is found, then the reference is satisfied and removed from the load data. This operation is performed as a side effect of BIND, and is hence redundant if BIND is also called.

Parameters: external procedure reference list or .ALL

13. SATISFY DATA

For each data reference given, a search is made of the list of data entries. If a matching data entry is found, then the data reference is satisfied and removed from the load data. This operation is performed as a side effect of BIND, and is hence redundant if BIND is also called.

Parameters: data reference list or .ALL

14. FUSE CODE

This causes the shareable symbol tables (SST) to be permanently appended to the code area. Hence a subsequent LINK command will not have its usual effect of separating these areas - LINK normally collects all code areas together and follows this by a corresponding series of SSTs. The intention is to prevent a degradation of paging behaviour. All relocation requests with base SST are amended to use the code area as base address.

Parameters: None

15. FUSE GLA

This causes the unshared symbol tables (UST) to be permanently appended to the GLA. See FUSE CODE.

All relocation requests with base UST are amended to use the GLA area as base address. In addition, the list of data entries is searched. If any is defined with respect to the UST, then the definition is amended to refer to the GLA.

Parameters: None

16. BIND

This command performs pre-loading operations on the object file so that it can be loaded subsequently at reduced cost. To do this it assumes fixed sites for the code, GLA and stack, and hence can satisfy all relocation requests. It also creates an area for uninitialised COMMON and satisfies data references to it. An entry is then added to the object file listing. To achieve the greatest savings in load time, the BIND command should be preceded by calls on the commands to suppress procedure and data entries. BIND automatically attempts to satisfy all procedure and data cross-references, so the explicit commands for these operations need not be called where BIND is used.

Note that once bound, an object file cannot subsequently be modified or linked. In addition, a bound file must be the first file loaded if called at all; this also means that only a single bound file may be loaded at a time.

Parameters: None

17. CLOSE

This terminates the user input. A confirmatory message is printed if the modification is completed successfully.

Sandy Shaw

Use of COMREG (27) for Compile Time Options

The following table shows the current use of the bits in COMREG (27). COMREG (26) is also allocated for use for additional PARMs.

Bit n (2 ** n)	SET	UNSET	NOTES
31			Not yet allocated
30	JOBBER MODE	NORMAL MODE	Not set by user
29	MISMATCH	NO MISMATCH	Fortran only
28	R8/PARMX	R4	Fortran - default REAL*8. Imp for test facility.
27	L8/PARMY	L4	Fortran - default LOGICAL*8. Imp for test facility.
26	I8/PARMZ	I4	Fortran - default INTEGER*8. Imp for test facility.
25			Not yet allocated.
24	Stack Size Set		Temporary use - not set by user
23	NOLINE	LINE	
22	EBCDIC	ISO	
21	Diag stream set		Temporary use - not set by user
20	DYNAMIC		
19	FREE	FIXED	
18	DEBUG		
17	MAP		Used by command LINK
16	OPT		
15	ATTR		Fortran - list attributes of variables
14	CODE		
13	LET		
12	LABELS	NO LABELS	
11	XREF	NO XREF	Fortran - list variable cross reference list.
10	ZERO		
9	INHIBIOF		Fortran - inhibit some I/O checks
8	IMPS	IMP	
7	PROFILE		
6	NOTRACE	TRACE	
5	NOARRAY	ARRAY	
4	NOCHECK	CHECK	
3	STACK		
2	NODIAG	DIAG	
1	NOLIST		
0	QUOTES	PERCENT	

Roderick McLeod, October 1978

This command allows EMAS users on machines attached to the RCOnet to send each other messages. The current pre-release version operates only on the 2972.
Please send suggestions or comments to S. Shaw.
For the present, INSERT(ERCC27.MAILY) to gain access to the command.

Basic operations

MAIL provides a number of facilities for manipulating messages; however to get started, the subset described below should be sufficient.

1. Composing and sending a message

   First invoke MAIL:

```
Command:MAIL
Mail:
```

   The prompt 'Mail:' is issued whenever MAIL expects the next command to be specified. To send a message to "Bill Smith", first check on the correct form of the name to use, (it may be "B.Smith" or "W.Smith" or "B.A.Smith" etc) :

```
Mail:DIRECTORY *SMITH
B.Smith, H.T.Smith, S.Smith
```

   Then use COMPOSE to create the draft message:

```
Mail:COMPOSE
Subj: Test Message
To:   B.Smith
Text:
: Here is a test message
::
Send or send-and-file-a-copy now?
Y or Fcc or N:
```

   If you are happy with the message, send it by replying "Y" ("Fcc" will in addition file a copy of it). If the reply "N" is given, then the contents of the draft can be amended or reviewed before sending:

```
Mail:LIST DRAFT
```

   Once the draft is satisfactory, use SEND to dispatch it:

```
Mail:SEND
Message sent
```

   The draft message may be filed without being sent:

```
Mail:FILE DRAFT
```

2. Receiving and displaying messages

You can receive messages into your process either when calling MAIL or by subsequent use of the ACCEPT command:

```
Command:MAIL(*)
```

If there are any outstanding messages, they are taken from MAILER and a one line summary is printed for each. To accept messages when you are already within MAIL:

```
Mail:ACCEPT
```

Newly received messages can be listed on the console:

```
Mail:LIST NEW
```

No action is required to retain received messages; see DISCARD, described below, about deleting them.

3. Reviewing and deleting mesages

A one line summary of each message held can be produced using SCAN:

```
Mail:SCAN ALL
```

The sequence number printed at the start of each one line entry can be used as a parameter to LIST or DISCARD, described below.

To review specific messages, find the required sequence number from the SCAN output, then use it with LIST; e.g., for message 3:

```
Mail:LIST 3
```

To get rid of unwanted messages use DISCARD:

```
Mail:DISCARD 3
```

To get rid of all currently held messages use the keyword ALL:

```
Mail:DISCARD ALL
```

Discarding simply marks messages as eligible for removal. A subsequent call of TIDY is needed to purge them entirely:

```
Mail:TIDY
```

On-line assistance is provided by HELP. Given no parameter it offers a table of contents; otherwise it lists information about a specified MAIL command. Return from viewing the help file back to MAIL by typing QUIT:

```
Mail:HELP SCAN
.
.
.
View:QUIT
```

Overview

The MAIL command provides facilities for composing, amending, sending, receiving and storing messages. Messages are held in store map files called folders. All the correspondence which relates to one topic can be conveniently held in one folder.

In turn, messages themselves consist of a number of components. The body of the message consists of a text component, and the header of the message consists of all the other components, e.g.

```
Subj: Meeting on Wednesday, 2 p.m.
From: S Shaw
To:   J Smith, j jones

Please note the new location for our meeting is Room 2019.
S.
```

The process of composing a message entails placing text into the various components of the draft message (which is not held in a folder). For example, addresses go into the "To:" and "cc:" components and the body of the message goes into the "Text:" component. A draft message can be Sent which causes its delivery to all the indicated recipients; each will receive a one-line TELL message indicating that they have outstanding mail. After Accepting messages, the user may selectively List them. Further manipulation of a message can involve Forwarding copies to additional recipients, Replying to its author, Filing for later reference or Discarding.

The messages in a folder are ordered chronologically and may be referenced by their index number (i.e. by their position in the folder), or by using special labels. At any given moment the system has a current folder and within it a current message which are under scrutiny; this avoids having to specify a folder and message index for every command.

A standard folder called M#INBOX is created by the system. When the MAIL command is invoked, the normal action is to Open the standard folder and select it as the current folder.

Only the draft message may be modified. However, any message in a folder can be Copied to the draft; similarly, the draft message can be Filed in a folder.

You are notified when a message has arrived for you by a TELL message from the executive process MAILER. At this stage the message has not been placed in your process - you must call the MAIL command and explicitly Accept it to cause the transfer to be performed.

Calling MAIL

The MAIL command takes two parameters, both optional:

```
Command:MAIL(<R-name>/<folder name>)
```

With no parameters, the program issues the prompt 'Mail:' to indicate that it is expecting a MAIL directive to be given. This prompt is

reissued after each directive has been performed until the directive STOP or QUIT is given:

```
Command:MAIL
Mail:
```

If a folder name is specified (preceded by '/'), that folder is OPENed and hence made the current folder; otherwise M#INBOX is made the current folder:

```
Command:MAIL(/RFOLDER)
Folder RFOLDER contains 16 messages
```

If <R-name> is specified (or '*', interpreted as the surname string of the process in use) then an ACCEPT <R-name> command is invoked, i.e. mail from other users is accepted:

```
Command:MAIL(*)
2 new messages
.
.
Mail:
```

The parameters for each MAIL command generally take the form <input>/<output>. If no <output> parameter is given then the slash (/) can be omitted.

A command name need not be typed out in full and in most cases may be abbreviated to a single letter. Where an ambiguous name is given, e.g. "F", then alphabetical order determines the command selected. Hence "F" will invoke "File", "FO" will invoke "Forward". Lower case input is accepted throughout.

Summary of commands

```
Accept    - take outstanding messages; may be implied when calling MAIL
Accredit  - add an alias R-name to the name/address directory
Compose   - create a draft message and offer to SEND it
Copy      - create (or append to) components of the draft message
Directory - list an extract of the name/address directory
Discard   - mark messages as discarded, or destroy draft components
Discredit - remove an R-name from the name/address directory
Ecce      - invoke ECCE to edit a component of the draft message
Edit      - invoke the standard editor to edit a component of the draft
File      - copy messages to another folder, then discard the originals
Forward   - package up an existing message for retransmission
Goto      - make the message specified the current message
Help      - provide on-line information about MAIL commands
List      - display messages on the console or list to a file
Next      - list the next message in the folder on the console
Open      - make the folder named the new current folder
Output    - list a message component to a file,device or the console
Previous  - list the previous message on the console
Quit      - exit from MAIL and return to the Subsystem
Reply     - create a draft message in reply to one received
Retrieve  - remove "discarded" status from messages
Scan      - produce a list of contents for the current folder
Send      - package up a message and submit it for transmission
Stop      - exit from MAIL and return to the Subsystem
Tidy      - purge discarded messages from a folder
```

Addressing conventions

Messages are addressed to individuals by using their "recipient names" (R-names). Each R-name is unique and, for EMAS users, corresponds to the surname string associated with their EMAS process (the same string is printed on line printer output banners). Every EMAS user with a unique R-name is automatically accredited to the MAIL system, i.e. the name is entered in MAIL's name/address directory. Information on R-names known to MAIL can be found by means of the command DIRECTORY (see below). A user can have his surname string (and hence R-name) changed by making a request to the System Manager.

Each R-name has an associated address of the form [username @ host]. Either the R-name or the address may be used to direct messages, e.g.

```
S.Shaw   and   [ERCC27 @ 2972]
```

are both valid forms for specifying recipients.

When a specified R-name is being processed by MAIL, the case of the letters is ignored, as are spaces and dots.

A user can avoid having his surname string made known to other users as an R-name by setting permission NONE to the executive process MAILER:

```
Command: PERMIT(.ALL,MAILER,N)
```

An arbitrary number of R-names may be associated with a given address. This may be convenient where several people share a process or where one person has several roles. Requesting additional R-names as aliases is provided as a user facility (see the command "Accredit", below).

Message lists

Within a folder, messages can be referenced by index number (position in the folder), and a collection of messages can be referenced at one time, by using commas and dashes as connectors. Hence the specification "1, 7-9, 90>89" refers to messages one, seven, eight, nine, ninety and eighty-nine. The angle bracket is like dash except that it indicates that the sub-list is in descending order.

Also, certain keywords define groups of messages, so that "new, 10-15, last" will reference all new messages, the tenth to fifteenth and the last message in the folder. The same message may appear more than once in such a list, but this does not imply any "repetition" of the message.

Keywords may be abbreviated to a shorter form (as short as a single letter). But note that "D" is a short form of "draft" (not "discarded") and "N" is a short form of "new" (not "next").

The terms defined below (with the exception of "draft") relate to messages in the current folder; they are as follows:

n	-	message n in the folder
n1-n2	-	messages n1 to n2 (n1 less than n2)
n2>n1	-	messages n2 to n1 (n2 greater than n1)

all	-	all messages in the folder
current	-	the message currently under scrutiny
discarded	-	messages which have been discarded to a "wastebin" but which the janitor has not yet taken away (see the "Discard" command, below)
draft	-	the single message which is currently being prepared; it is not held in any folder
last	-	the last message in the folder
new	-	received messages which have not yet been LISTed
next	-	the first message after the current one
old	-	messages in the folder which are not discarded, new or saved
previous	-	the first message preceding the current one
saved	-	former draft messages which have been saved by being FILEd in the folder

Message components

A message is composed of a series of components. The body of the message consists of a "Text:" component - all the other components together constitute the header of the message. The following components are defined:

Date: Indicates the date and time when the message was sent.

Subj: Gives a brief indication of the content of the message and is displayed when the message is SCANned.

From: Indicates who sent the message. This field may be set by the sender to contain any text he wishes (if, for example, he is sending a message on someone else's behalf); in this case, MAILER will add a "Sender:" component to the message to show who actually sent it.

Sender: Shows who actually sent the message and indicates that the "From:" component is not authentic.

To: The one or more primary recipients of the message.

cc: One or more secondary recipients (who receive carbon copies).

bcc: One or more tertiary recipients (who receive blind carbon copies, see below).

Msg ID: This holds a unique message identifier and is composed of three parts:

1) message server name (= host)
2) a numeric identifier allocated by the server
3) date and time when the message was sent

In reply to: This component is added to a message by the REPLY command, and identifies the message being replied to.

References: Not filled in or used by any MAIL function, this component may contain any text.

Keyword: Again, this component may contain any text and may be used to classify messages or to direct the operations of programs which automatically receive and manipulate messages.

Folder: Indicates the name of a folder in which the sender recommends the recipient to file the message.

Mode: If specified, may contain the value BINARY or EHEADER. BINARY indicates that the body of the message is a binary file. EHEADER indicates that the body of the message includes a standard EMAS header.
The LIST command does not attempt to display the body of a message which has either mode set. The original file may be reconstituted using the OUTPUT command.

When sending a message to a number of recipients, the sender may determine how much information each recipient receives about his co-recipients by selective use of the "To:", "cc:, and "bcc:" components.

To: a recipient in this list has details of all the recipients included in his copy of the message

cc: a "cc:" recipient does not have the "bcc:" component included in his copy of the message.

bcc: a "bcc:" recipient will receive a copy of the message containing neither the "cc:" nor "bcc:" components; his own name is added to the "To:" component in his copy of the message.

Hence, a message addressed

```
To:     Black
cc:     Brown
bcc:    White
```

will be received by each as follows:

```
Black - To:   Black
        cc:   Brown
        bcc:  White

Brown - To:   Black
        cc:   Brown

White - To:   Black, White
```

This feature may be useful where a distribution list containing many names is given - the sender can avoid burdening each recipient with a long list of names in which he has no interest.

Many of the commands of the MAIL program take a message component parameter. The full specification is:

<component name>:<message>(<folder name>)

e.g. cc:4(LETTERS)

The folder name can usually be omitted - by default the message referred to will be in the current folder. The <message> part of the specification may also be omitted, in which case the current message under scrutiny is assumed. Hence the message component specification "To:", refers to the "To:" component of the current message in the current folder.

Message component names can be abbreviated to a shorter form - in all cases, the first two characters of each component name give a unique abbreviation (hence "SE:" is equivalent to "SENDER:").

Note that a message keyword which may define more than one message can be used - the message selected is the first message found which matches the keyword.
Hence "Subj:NEW" refers to the "Subj:" component of the first NEW message in the current folder.

Mail commands

Mail:ACCEPT <R-name>, <password> / <folder name>

This command is used to accept messages sent to you by other users. By default, the messages are put in the current folder:

Mail:ACCEPT

If the <folder name> parameter is specified, then the messages are stored in that folder (it is made the current folder). The <R-name> parameter is required in two cases:

- where you want to accept mail directed to an alias R-name, e.g.

  Mail:ACCEPT EMAS Suggestions/SUGGBOX (puts mail addressed to "EMAS Suggestions" into folder SUGGBOX)

- where you are accepting mail within an EMAS process other than your own, e.g.

  Mail:ACCEPT S Shaw, PASS

After taking the outstanding messages, a SCAN of new messages (i.e. messages not yet LISTed) is performed. The first new message in the folder becomes the current message.

Mail:ACCREDIT

This command allows users to add additional R-names for themselves (aliases) to the name/address directory. This may be useful where several people share one EMAS process, or where one person has several roles. The facility should not be used to define names which

will be meaningless to most MAIL users, such as nicknames known to only one or two people.

You may also set a password for the R-name which will make it possible to accept messages at another address (see ACCEPT). Passwords may be up to seven characters long.

It is also possible to set a "Department" field (31 characters) to be associated with the R-name. The information is displayed when a search is made of the name/address directory (see DIRECTORY) and is intended as an aid to distinguishing recipients with similar R-names.

Note that if you want to have your default R-name changed, this can only be done by application to the System Manager.

```
Mail:ACCREDIT
Rname:      Mail Suggestions
Password:   ABCDE
Department: EMAS MAIL Suggestion Box
```

Mail:COMPOSE <component names>

This command offers a convenient way of creating a draft message. The draft is first cleared, then prompts are issued as indicated:

```
Mail:COMPOSE
To: J.Jones
Subj: Old joke
Text:
: Send 3/4d were going to a dance.
::
Send or send-and-file-a-copy now?
Y or Fcc or N:Y
Message sent
```

Once the draft has been composed, the program offers the option of sending the message or returning to MAIL command level. Input for each component (except "Text:") is terminated by a newline unless the character before the newline is a comma; thus, for example, the "To:" component can be given a list of R-names which extends over several lines. Null input is accepted.
The "Text:" component is terminated by a colon (:) on a line by itself. The prompt printed when lines of the "Text:" component are being input is ':'.

In addition to accepting text, the COMPOSE command will accept the contents of an EMAS file or of a component of an existing message. The escape character '@' must be used to indicate this form of input. You may request that prompts are issued for additional message components by giving the component names as parameters:

```
Mail:COMPOSE cc:
To: @ERCC27.NAMES
cc: J.Jones
Subj: @Subj:
Text:
:@text:4
Send or send-and-file-a-copy now?
Y or Fcc or N: N
```

In this example, a file is copied to the "To:" component of the draft, the "Subj:" component of the current message is copied to the "Subj:" component of the draft, and the "Text:" component of message 4 is copied to the "Text:" component of the draft.

If the draft message is not sent (as in the example above), it may be modified further then dispatched using the SEND command.

Mail:COPY <input>/<component of the draft or DRAFT>

This command allows text to be copied to a component of the draft. Alternatively a complete message may be copied to the draft. If <input> is not specified then a prompt is issued and the text to be copied is read from the terminal. If the component of the draft is not specified, the "Text:" component is assumed. Alternative sources of input are an EMAS file or a component of an existing message.

```
Mail:COPY              (No parameters, so input is prompted
Text:                  for and is taken to be for the "Text:"
:This is input         component of the draft message)
::

Mail:COPY MSGTEXT      (EMAS file MSGTEXT is copied to the "Text:"
                       component of the draft message)

Mail:COPY CC:/TO:      (copies the "cc:" component of the current
                       message to the "To:" component of the draft
                       message)
```

If the input is to be appended to the existing contents of a component of the draft message rather than overwrite it then the symbol '+' should be inserted before the component name:

```
Mail:COPY MYLIST/+TO:  (appends the contents of the file MYLIST to
                       the "To:" component of the draft)

Mail:COPY CC:4/+CC:    (appends the contents of the "cc:" component
                       of message 4 in the current folder to the
                       "cc:" component of the draft)
```

Alternatively, a complete message (i.e. all its components) may be copied to the draft message. In this case the default for the first parameter is the current message.

```
Mail:COPY /DRAFT       (copies the current message to the draft)

Mail:COPY NEW/DRAFT    (copies the first new message in the
                       current folder to the draft)
```

Mail:DIRECTORY <R-name mask>, FULL/<output>

This command allows a search to be made of the name/address directory. The search may be for a specific R-name, or for all R-names that fit a mask. As in the Subsystem command FILES, the mask consists of up to three fields where a field is either a string of explicit characters or the symbol "*", representing any characters. Upper and lower case characters are not distinguished, and space and dot characters are ignored.

If the second parameter ("FULL") is specified, then additional information is given on each R-name selected, under the following headings:

User - the user number to which messages are delivered

Server - the host on which the recipient is accredited

Options - currently takes one of two values:

S'name - the R-name is the standard process surname string, set by the System Manager

Alias - the R-name was accredited by the user himself

Dept - a user-defined field set by ACCREDIT

The <output> parameter may be null (implying output to the terminal), or a filename or device name.

```
Mail:DIRECTORY *shaw
A.Shaw, B.C.  Shaw

Mail:DIRECTORY *MAC*/.LP

Mail:DIRECTORY *JONES,FULL/TEMP1
```

Mail:DISCARD <messages or draft components>

This command marks one or more messages in the current folder as being discarded, but does not physically remove them or re-number the remaining messages in the folder. The action is like placing a message in the wastebin - it is still available though less convenient to access, and is subject to permanent removal later by the TIDY command (see below). If no TIDY has been performed after a DISCARD then the RETRIEVE command can be used to recover the messages.

As with other commands, DISCARD can be applied to the draft. In addition, for the draft only, individual components may be discarded. (However, discarded components of the draft are destroyed immediately and cannot be recovered by RETRIEVE.)

```
Mail:DISCARD 1-4,SAVED,CC:  (discards messages 1-4 and all SAVED
                            messages in the current folder, plus the
                            "cc:" component of the draft)

Mail:DISCARD DRAFT          (discards all components of the draft
                            message)
```

The last message specified in the list becomes the current message if it is not the draft.

Mail:DISCREDIT

This command removes an R-name from the name/address directory. A password must be supplied if the command is called from any process other than that associated with the R-name. In order to have your default R-name removed permanently from the name/address directory it is necessary to set a file index permission of "none" to MAILER:

```
Command: PERMIT(.ALL,MAILER,N)
```

Discredit is called as follows:

```
Mail:DISCREDIT
Rname: MAIL Suggestions
Password: ABCDE
```

Mail:ECCE <component or EMAS file>/<component of the draft>
Mail:EDIT <component or EMAS file>/<component of the draft>

This command allows an EMAS file or an existing component of any message file to be edited, and the result placed in a component of the draft.

If no parameters are given, then the "Text:" component of the draft message is edited. If an output draft component (i.e. one following "/") is given, then the existing contents of that component are overwritten.

Mail:ECCE	- edits the "Text:" component of the draft, creating it if none already exists
Mail:EDIT CC:	- edits the "cc:" component of the draft, creating it if none exists.
Mail:ECCE Text:current/text:	- edits the "Text:" component of the current message to the "Text:" component of the draft.
Mail:EDIT /TO:	- edits an empty file to the "To:" component of the draft.
Mail:EDIT MYLIST_N1/TO:	- edits an EMAS file to the "To:" component of the draft.
Mail:ECCE CC:2	- this will fail; a component of the draft must be given following "/" in this case.

Mail:FILE <list of messages in the current folder>/<folder>

This command copies messages from the current folder to another folder, then discards the messages from the current folder. The input list of messages defaults to the current message. The draft message may also be filed (this is the only message which can be filed in the current folder); a filed draft message is given SAVED status (see SCAN).

The last message in the list of messages to be filed becomes the current message.

```
Mail:FILE /folder2      - files the current message to folder2

Mail:FILE DRAFT         - files the draft to the current folder

Mail:FILE NEW,1,LAST/folderb  - files all new messages plus the
                                first and last in the folder to folderb
```

Mail:FORWARD <message>

This command sends a copy of a message to another user or users. MAIL issues a prompt for the name of the user or users to whom you wish to forward the message. You reply to this with the name(s) or alternatively specify an EMAS file or a message component which contains the names.

It is often useful to add comments to the end of a forwarded message. This can be done after FORWARDing but before SENDing the message by using COPY or EDIT.

As with COMPOSE and REPLY, you are offered the option of sending the message once the recipients have been specified. If <message> lies within the current folder, it becomes the current message.

```
Mail:FORWARD 1          - forward the first message in the
To: Rowland Hill          current folder

Mail:FORWARD LAST(FOLDER2)  - forward the last message in folder2
To: @MYGROUP                to the list of recipients specified
                            in file MYGROUP

Mail:FORWARD NEW        - forward the first new message in the
To: @cc:                  current folder to the list of recipients
                          held in the "cc:" component of the current
                          message.
```

Mail:GOTO <Message>

The message specified becomes the current message. If a message keyword is used which may select more than one message, the first message found is selected.

```
Mail:GOTO 1     - go to the first message in the current folder

Mail:GOTO NEXT  - the next message after the current one

Mail:GOTO NEW,LAST  - go to the first new message; if there are
                      none, go to the last message in the current
                      folder
```

Mail:HELP <command>

This command provides information about the MAIL system and includes descriptions of all the MAIL commands. It operates by VIEWing a file containing the help text; hence the whole of this file can be explored at one time. If no parameter is given a table of contents is printed and further input requested. Alternatively, the name of a MAIL command may be given as a parameter.

Return from viewing the help text to MAIL using Q or QUIT.

```
Mail:HELP
•
•
View: QUIT

Mail:HELP COMPOSE
•
•
View: QUIT
```

Mail:LIST <messages>/<file or device>

This command displays messages in the current folder on the console, or alternatively lists to a file or device. In the latter case, a SCAN (see below) is prepended to the listing. If <messages> is omitted, the current message is listed.

```
Mail:LIST  - displays the current message on the console

Mail:LIST NEW,DRAFT/.LP    - lists all new messages plus the draft
                             to the line printer
```

If a message listed contains a "Mode:" component, this indicates that the body of the message contains a binary file or an EMAS file complete with header - hence the "Text:" component of the message is not listed. The "Text:" component may be reassembled as a file using the OUTPUT command.

Note that as a side effect of LIST, the status of a NEW message is changed to OLD.

Mail:NEXT

This command LISTs on the console the first undiscarded message after the current message. (Note the difference in meaning between this and the "next" message-reference keyword). The message listed becomes the current message.

Mail:OPEN <folder name>, NEW

This command switches primary attention to another folder, i.e. makes the folder named the current folder. It is also used to create a folder; in this case the second parameter NEW must be given. If no parameter is given, the standard folder M#INBOX is made the current folder.

```
Mail:OPEN BUGS,NEW - creates a new folder BUGS and makes it the
                     current folder

Mail:OPEN F2       - makes an existing folder F2 the current
                     folder

Mail:OPEN          - makes M#INBOX the current folder.
```

If the parameter '?' is given, then the name of the current folder and the number of messages in it is printed:

```
Mail:OPEN ?
Folder M#INBOX contains 16 messages
```

Mail:OUTPUT <component>/<file or device>

This command is used to transfer a single component of a message to a file or device or to the console. The default component is the "Text:" component and the default message the current message. If the "Text:" component of a message is to be output, then a check is made on whether a "Mode:" component is also present: for a "Mode:" value of BINARY, a store map file is output; for a "Mode:" value of EHEADER, an EMAS file complete with header is output.

Mail:OUTPUT	- displays the "Text:" component of the current message on the console.
Mail:OUTPUT CC:4(F3)	- displays the "cc:" component of message 4 in folder F3 on the console
Mail:OUTPUT /OBJECT1	- outputs the "Text:" component of the current message to a file

Mail:PREVIOUS

This command LISTs on the console the first undiscarded message prior to the current message. (Note the difference in meaning between this and the "previous" message-reference keyword). The message listed becomes the current message.

Mail:QUIT

Exits from MAIL and returns to Subsystem command level.

Mail:REPLY <message>

This command provides a convenient way of replying to a received message.

If no parameter is given then a reply to the current message is produced. A prompt is issued for the "Text:" of the reply. The escape character '@' can be used at this point to indicate input from an EMAS file or from an existing message component:

```
Mail:REPLY 2
Text:
:Your message received
::
Send or send-and-file-a-copy now?
Y or Fcc or N: N
```

```
Mail:REPLY LAST(FOLDER3)
Text:
:@ACK_VPOLITE
Send or send-and-file-a-copy now?
Y or Fcc or N: Y
```

If a <message> in the current folder is specified, it is made the current message.

Mail:RETRIEVE <messages>

The complement of DISCARD, this command changes the status of the specified messages in the current folder from "discarded" to "old". Note that once discarded the draft message cannot be retrieved. If <messages> is null, the current message is retrieved.

The first message in <messages> becomes the current message

```
Mail:RETRIEVE DISCARDED

Mail:RETRIEVE 4,10-12
```

Mail:SCAN <messages>/<device or file>

This command scans the specified messages in the current folder and produces a "list of contents" - a series of one line summaries for the messages. By default, NEW messages are scanned and the output is directed to the console.

The format of the one line summaries is as follows:

status	-	null	= old message
		s	= saved draft message (created when a draft message is FILEd)
		x	= discarded message
		*	= the draft message
		n	= new messages, i.e. messages not yet LISTED
index	-	the index number of the message within the folder	
<=	-	for the current message, the indicator '<=' is printed	
(length)	-	the length of the "Text:" component of the message in bytes	
date	-	the day and month that the message was sent	
from	-	the "From:" component of the message; if this component is empty, the "To:" component is displayed (prefixed with "To:")	
subject	-	the "Subj:" component of the message; if this component is empty the first few bytes of the "Text:" are displayed.	

Mail:SEND <message>, FCC/<folder name>

This command packages up a message and submits it for transmission. If <message> is omitted, the draft message is sent.

The FCC parameter indicates "File carbon copy", and causes a copy of the message to be filed in the current folder or the specified folder (this action is conditional on the message having been sent successfully). If the message sent lies within the current folder then it becomes the current message.

```
Mail:SEND , FCC          - sends the draft and files a copy in the
                           current folder

Mail:SEND 1(STANDARD) - sends the first message in folder STANDARD
```

Mail:STOP

Exits from the MAIL program and returns to Subsystem command level. STOP is identical to Mail:QUIT.

Mail:TIDY <folder name>

This command causes discarded messages to be purged from the indicated folder (by default, the current folder). TIDY is irreversible.

The remaining messages in the folder are ordered by transmission date (in the case of SAVED draft messages, by date of filing), and hence the message index numbers change.

If the current folder is tidied, the current message becomes the first in the folder.

```
Mail:TIDY F2
Mail:TIDY
```

EMAS 2900

New HELP facility

After lengthy discussion with advisors and others in the Centre I have decided that there is a continuing role for the HELP command. I see it as a very simple to use source of first level information on a wide variety of topics. I propose to change the user interface slightly in the following ways :

* reduction to a single parameter - a keyword - hence removal of mechanism to send output elsewhere.
* removal of facility to list all entries HELP (.ALL).
* generalisation of keyword structure - no longer restricted to valid membername. This allows e.g.

```
HELP COMMAND PARAMETERS
HELP .LP                (information about .LP)
```

* a facility may be included to attempt to provide possible matches for unknown keywords.

Apart from these changes in appearance the system will monitor all calls - particularly un-successful calls to determine what subjects users are asking about, for which we currently provide no information.

Improvement to Information

The changes listed above are minor compared to the changes to the information on which the system relies. It is proposed to expand this vastly to include references to many more topics. For many entries the information returned will be the source of further information - e.g. PACKHELP, MICROAID etc. Enough information must be given for the novice user to get to the further information.

The aim will be to provide brief, concise information and it is suggested that a maximum of 10 lines of 72 characters be provided per entry. This will be appropriate for use on slow terminals as well as faster terminals.

Development

Brian Murdoch will be responsible for providing the software for the routine HELP itself and for any tools required to maintain the data base. Information for entries will be collected from the existing HELP system, and a wide variety of other sources by Roderick McLeod and Nick Stroud. Advisors will be asked via ADLIB to suggest areas and to provide information on the specialities.

Conclusion

This development is thought to be a necessary step in the provision of a simple, single entry, source of information on a wide variety of EMAS related topics. It is consistent with packages such as MAIL, VIEW, SCREED which all provide their own internal HELP information, in that it will provide just enough information for the user to get to the facility concerned. In the case of the basic EMAS commands it will continue to provide a main source of information - backed up in the case of the more complex commands with references to the Users Guide.

Roderick McLeod

SCREED: A Screen Editor

11/2/81

Introduction

SCREED is a screen editor available on EMAS at ERCC. Details of access and a list of suitable video terminals are given in Appendix A.

The purpose of SCREED is to provide a simple means of editing character files. It does this by exploiting the fact that a video screen has 24 lines (usually) and that it is possible to move the "cursor" (current position marker) to any position on any of the lines. When characters are typed thereafter they appear on the screen at the cursor position.

Now, if a video screen were filled with lines of a character file, the effect of moving the cursor to some position on the screen and then typing would be to overwrite the text at that position. If a text editor running on EMAS had caused the file lines to appear on the screen in the first place, then this editor could note the user's subsequent movement of the cursor and when the user overwrote some of the text on the screen, it could make an exactly similar change to the original file.

The main advantage of this approach to editing over that used by, say, EDIT or ECCE is that instead of typing a "command string" to cause the editor to change the file, the user is "changing it himself", or so it appears, and the effect on the file is immediately obvious.

This summary of the principle of screen editing leaves a number of questions unanswered. For example:

* How do I tell the editor which part of my file is to be displayed on the screen?

* What happens if the text to be added at some point in the file is smaller (or larger) than the text to be removed?

* The cursor control keys on my terminal (the "arrow keys") do not work with EMAS.

To take the last point first: the communications network rejects most of the "control" characters if you type them on your keyboard. However, the network has been modified to allow a new mode of operation in which all characters are acceptable. (This is necessary because different types of video do the same thing, such as moving the cursor around, in quite different ways, and so it is not possible to identify a set of "cursor moving" control characters.) This new mode of operation is automatically selected within SCREED.

To delete text without replacing it by new text, a delete character is provided. This is '^' by default, but it can be changed. To add text to the file you need to use "control commands", described below.

Before reading further, you are advised to log on to EMAS and invoke SCREED (refer to Appendix A for details of access):

```
Command:SCREED(input file, output file)
```

The file specification is as for ECCE. Choose as input an unwanted file, or a copy of a file. SCREED will ask you to indicate which type of video terminal you are using. Once this has been replied to, the screen will go blank and then the first two lines of the input file will appear at the top of the screen. Holding the 'control' button down, type 'R' followed by 'S'. The first screenful of the input file will appear on the screen. You will find that the cursor moving buttons will operate - line feed, return, up arrow, down arrow, backspace, home, etc.

TO TERMINATE AN EDIT SESSION, TYPE
STOP followed immediately by ctrl X (ctrl Z on some terminals)
or ABORT followed immediately by ctrl X (ctrl Z on some terminals)
(Details of this type of edit command are given below.)

[Note that '(A)' in the description below stands for control A, i.e. the code generated by holding down the control button and typing 'A'. Similarly (B), (C), etc.]

When SCREED is being used, there are no "command line" or "menu" areas on the screen, and - apart from delete characters - no characters stand for other than themselves: all that you see is file. However, some blank lines on the screen may not correspond to blank lines in the file. Such screen lines are called "null".

To move the cursor to a specific part of the screen the user can make use of whatever keys are available on his keyboard: Tab, Backspace, Return, Line Feed, |, |, <-, ->, Home - but not the space bar since it inserts a space character at the current position. In addition to those keys there are cursor positioning commands provided by SCREED, as described below.

The user commands SCREED by means of pairs of control characters. The first indicates an action and the second the text unit to which the action applies. If the "action" control is omitted the last action control given is assumed.

Action		Text Unit
(A)dd (B)eginning of (C)onjoin (D)isjoin (E)nd of (O)mit (N)ext (P)revious (R)ewrite (T)runcate	followed by	(W)ord (L)ine (S)creen

Any combination of an action followed by a text unit may be given, but may not be valid in some circumstances. For example, (O)(W) (Omit Word) will fail if the cursor is not on a "word" (a piece of text delimited by spaces or the beginning of a line or the end of a line). The editor signals failure by wiggling the cursor back and forth on the current line.

It is probably simpler to try out the effects of these commands on a file than to read a detailed description of their effects. Explanatory notes on some of the combinations may be useful, however:

1) Some commands move the cursor only, some delete text and some rewrite the screen completely.

2) (O)(L) turns the current line into a null line.

3) (O)(W) causes the current word and appropriate space characters beside it to be replaced by delete characters.

4) (R)(L) removes all delete characters from the current screen line in the process of rewriting it.

5) (D)(L) causes the current line to be broken ("disjoined") at the cursor. In general the remainder of the screen has to be rewritten to accommodate the new line produced.

6) (A)(W) shifts the remainder of the current line about 15 places to the right and fills the gap with delete characters. These can then be overtyped as required. If the end of the line was moved off screen in the process, it reappears on (R)(L), although you do not have to follow (A)(W) with (R)(L). (Indeed, there is no command which must be followed by some other specific command - they are all entirely independent.)

7) (A)(L) causes the current line and all the lines below it to be replaced on the screen by null lines, the current line then to be rewritten at the bottom of the screen and finally the cursor to be moved back to the first of the null lines. These null lines can then be used as necessary. Null lines below the last line added will be discarded on typing (R)(S), but note that null lines traversed prior to text being added lower down will be turned into blank lines (i.e. they become part of the file).
If your terminal is operated by SCREED in page mode (e.g. the Lynwood DAD-1), then (A)(L) causes the screen to be blanked, the line previously containing the cursor to be written at the bottom of the screen and the line before that to be written at the top of the screen. The screen lines between are all null.

8) (A)(S) is similar to (A)(L) except that the current line is not rewritten at the bottom of the screen. Thus an arbitrarily large number of lines can be added simply by scrolling the screen when you get to the bottom. Use of a "clear screen" key (if your terminal has one) is equivalent to (A)(S) - the resulting screenful of null lines is understood to come before the line which previously contained the cursor.
If your terminal is operated by SCREED in page mode, (A)(S) causes the screen to go blank, as though you had used a "clear screen" key (as described above).

9) (R)(S) Rewrites the screen, removing all delete characters and null lines. If there are null lines above the cursor when (R)(S) is typed then file lines prior to those currently displayed are brought down onto the screen; if there are null lines below the cursor when (R)(S) is typed then file lines following those currently displayed are brought up onto the screen. The only exception to this rule is when there are insufficient previous file lines to bring down: in this case the first line of the file appears at the top of the screen. On

the other hand, if there are insufficient file lines following those displayed to bring up onto the screen, then null lines appear at the bottom of the screen.

10) (D)(S) means that the screen is to be "disjoined" at the current line. The current line becomes the first of the new screen; i.e. the screen scrolls until the current line is at the top. In the case of page mode terminals, the screen is rewritten with the current line at the top.

11) (C)(S) implies that this screenful and the next are to be "conjoined". The resulting screenful shows the join exactly in the middle of the screen; i.e. the effect is to scroll up by half a screen. Again, to achieve the required effect on page mode terminals, the screen is rewritten to display the appropriate text.

12) (O)(S) causes all the lines displayed on the screen to be removed ("omitted") from the file. (T)(S) is similar: it removes the current line and all lines below it on the screen from the file.

13) (N)(S) causes the file line following the bottom one currently displayed on the screen to become the top line on the screen.

Having read this, you should now experiment with the various combinations of action controls and text units. A summary of commands appropriate to your video type is given in Appendix B.

These facilities are useful, but they are not sufficient. The following questions make this clear:

* How do I terminate (or abandon) the edit session?

* How do I get to the middle of a large file (typing (N)(S) fifty times is rather tedious)?

* Can I save the editing done to date?

* How can I correct twenty occurrences of the same mis-spelled word?

* Can I change the delete character?

All these facilities are provided by means of another control character, (X) [(Z) on some terminals - see Appendix B]. If you type in some text and follow it with (X), SCREED rewrites the line as it was before you typed the text, and then examines the text typed. If it is one of the following, in upper or lower case, then a special action is implied:

CLOSE (or STOP or END or EXIT)
ABORT
SAVE
EDIT
ECCE
DEL=<char>

If the text was none of these then SCREED takes no action.

CLOSE
STOP
END
EXIT

Any of these causes the edit session to be terminated.

ABORT — The edit session is terminated without the output file being written to.

SAVE — The file is copied in its present state to the output file, the screen is blanked and then the current line and the line before and line after are written in the middle of the screen (the other screen lines are null).

EDIT — A branch is made to the Subsystem editor EDIT. The current position is preserved. You can terminate the edit session from within EDIT in the usual way, or return to SCREED if you wish. This is done by hitting the ESC key and replying "SCREED" (or "screed") to the "INT:" prompt. On hitting the "Return" key the "Edit:" prompt will appear as usual, but if you hit "Return" again SCREED will be reverted to. Three lines will be displayed, as for SAVE.

ECCE — A branch to the editor ECCE is made. The return to SCREED, if desired, is precisely as for the return from EDIT.

Note that you can start your editing in ECCE or EDIT and then branch to SCREED by the use of the "INT:". At present you must invoke EDIT from command level by the command TEDIT, and ECCE by TECCE.

DEL=<char> — The specified character becomes the delete character in future.

Thumbing

To move around the file you could invoke EDIT or ECCE by use of (X), then move the current position, then return to SCREED. However you can also do this without leaving SCREED, as follows: suppose that the top line of the screen corresponds to the first line of the file and that the bottom line of the screen corresponds to the last line of the file; then halfway down the screen ≡ halfway down the file, three quarters down the screen ≡ three quarters down the file, etc. If you position the cursor on the left hand margin of a screen line and then type (X), the effect is to move to the corresponding position of the file. The screen is blanked and three lines from the implied file position are written, at the specified screen position. If this is not in fact the right place then you can adjust your position, up or down, and type (X) again. This is known as "thumbing", as you are effectively thumbing through the file.

The reason for SCREED writing only two or three file lines, on entry or following a (X) command, is to save time - you might want to "thumb" to some other part of the file, and writing a complete screenful of text would thus waste time. You can always use (R)(S) if you want to see a screenful; in this case, note the significance of the cursor position, as explained in note (9) above.

Other notes

* The Recall file is not written to during the operation of SCREED.

* If the terminal seems to go dead and not even (X) has any effect, there is an "INT:" character which you can resort to: (]) with most terminals - see Appendix B. Note, however, that using it causes you to lose all your editing.

* All constructive comments and suspected faults should be directed to me.

John Murison
ERCC
(667 1081 ext. 2639)

On 2972 or 2980:

```
Command:INSERT(KNTLIB.SCREEDY)
Command:INSERT(KNTLIB.ECCE24)
Command:INSERT(KNTLIB.EDIT27)
```

The command to invoke SCREED is:

```
Command:SCREED(input file, output file)
```

Either parameter (but not both) may be omitted.

The use of the parameters is as for ECCE. The only difference between the SCREED and EDIT parameters is when specifying that a new file is to be edited:

```
Command:EDIT(FRED)
Command:SCREED(,FRED)
```

are equivalent in this case.

Terminals supported

SCREED indicates which terminals it can support when you call it. At the time of writing these are:

Perkin Elmer 550

Newbury 7000 Series

Lear Siegler ADM-3A

Lynwood DAD-1

Visual 200

Pericom 6801

Relevant details of each of these terminals are given in Appendix B.

Note that some older video terminals are not suitable for use with SCREED, for example the ITT 3210.

Please contact me if you have a terminal which is not in the above list and which you think might be suitable for use with SCREED.

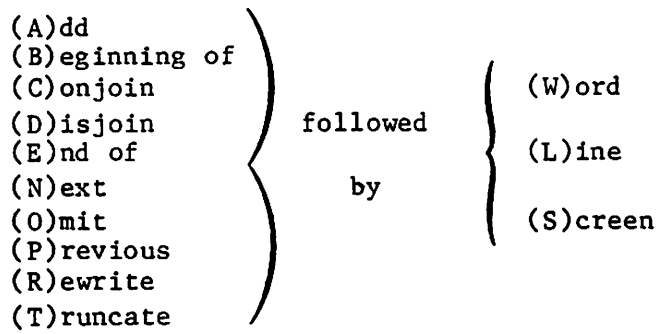
> WARNING
>
> This is a preliminary version of SCREED. It may be changed without warning. It may contain errors which corrupt files.

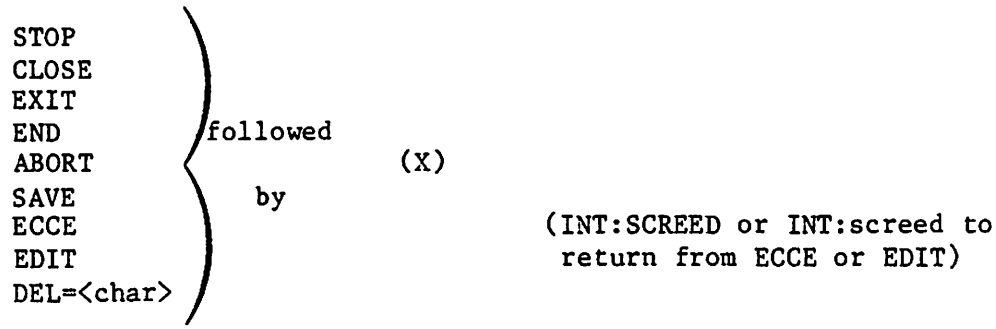
APPENDIX B-1: Perkin-Elmer 550

Command summary

Delete character is '^'. It can be changed (see "DEL=" below).

(A) below stands for Ctrl A (i.e. hold down the control button and type 'A'). Similarly (B), (C), etc.





You can "thumb" through the file by use of (X) on the left hand margin.

(]) in SCREED is the "INT:" character. All editing is lost if you use it.

Notes relating to the Perkin-Elmer 550

* There are no "arrow" keys. Thus the cursor has to be moved by the use of Tab, Backspace, Return, Line feed, and the appropriate SCREED commands (the (W) commands in particular).

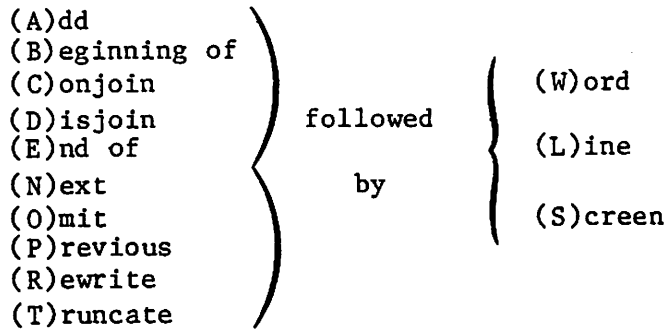
* The Clear key (which only operates when Ctrl is depressed also) does not transmit any code - the effect is entirely local to the terminal. Thus if you clear the screen by using it, SCREED is not aware of this. Therefore it should not be used.

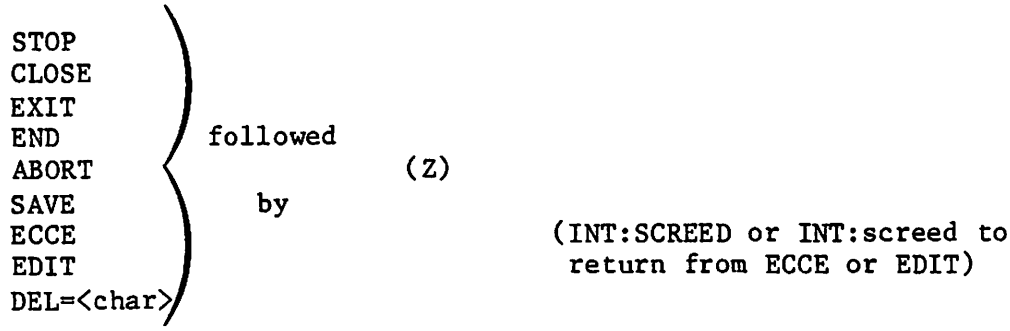
APPENDIX B-2: Newbury 7000 Series

Command summary

Delete character is '^'. It can be changed (see "DEL=" below).

(A) below stands for Ctrl A (i.e. hold down the control button and type 'A'). Similarly (B), (C), etc.





You can "thumb" through the file by use of (Z) on the left hand margin.

ESC in SCREED is the "INT:" character. All editing is lost if you use it.

Notes relating to the use of Newbury 7000 Series terminals

* As stated above, (Z) is used rather than (X), and ESC is the "INT:" character rather than (]).

* (With respect to the 7001:) The keyboard has a separate row of keys along the top. Most of these are either disabled or do not generate any codes. However the keys from Home rightwards are usable.

* If the terminal does not seem to be operating correctly when used with SCREED, it might not be set up appropriately. This can checked, and corrected if necessary, as described below. Note that during the procedure the terminal is effectively off-line, and so does not interfere with any on-line work in progress.

  Press the Edit key (which is not normally used), and then Ctrl and Home together. A screenful of text appears; it details various settings (which on most other terminals are determined by switches on the terminal). When you press Tab the cursor moves from one setting to the

next. If you press Return repeatedly the various possible values of the current option appear in turn.

The settings relevant to the operation of SCREED are as detailed below; the other settings should not be changed:

First line:

```
    ..............BELL=80;DELETE=N;NL=NO....     (BELL=80 turns off bell)

                    TELETYPE = NO
    :
    :
```

Under the heading ATTRIBUTES:

```
    PROTECT = NO
    BLINK = NO
    UNDERLINE = NO
    :
    :
```

Of the codes given under the heading FROM LINE, the following must be underlined:

            0C 1A

and the following must not be underlined:

            08 09 0A 0B 0D 16 18 19 1D 1F

The settings of the other FROM LINE codes is immaterial to the operation of SCREED.

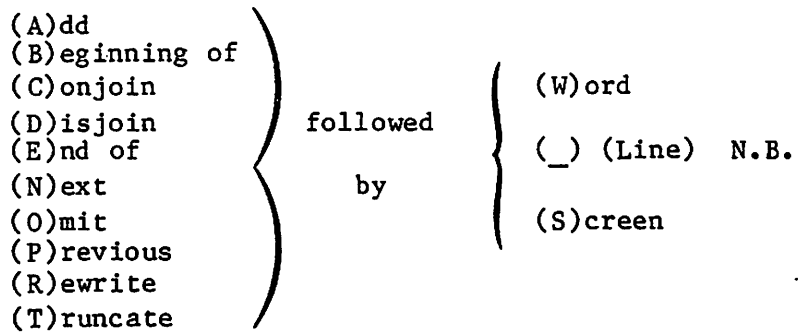
You should then finish by pressing ESC followed by Edit. This puts you back on-line to EMAS.

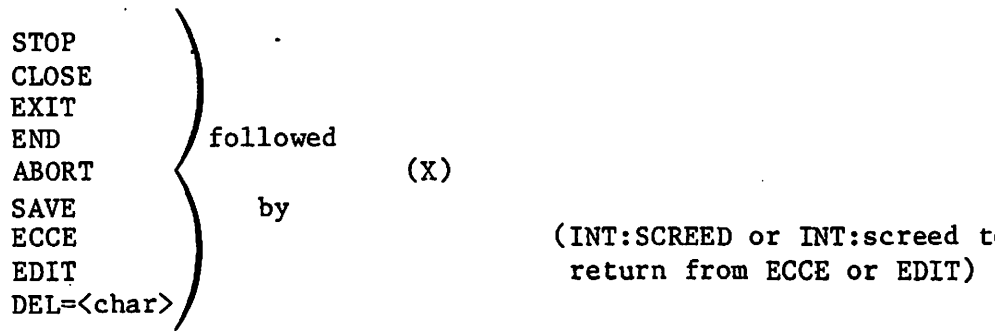
APPENDIX B-3: Lear Siegler ADM-3A

Command summary

Delete character is '^'. It can be changed (see "DEL=" below).

(A) below stands for Ctl A (i.e. hold down the control button and type 'A'). Similarly (B), (C), etc.





You can "thumb" through the file by use of (X) on the left hand margin.

(]) in SCREED is the "INT:" character. All editing is lost if you use it.

Notes relating to the Lear Siegler ADM-3A

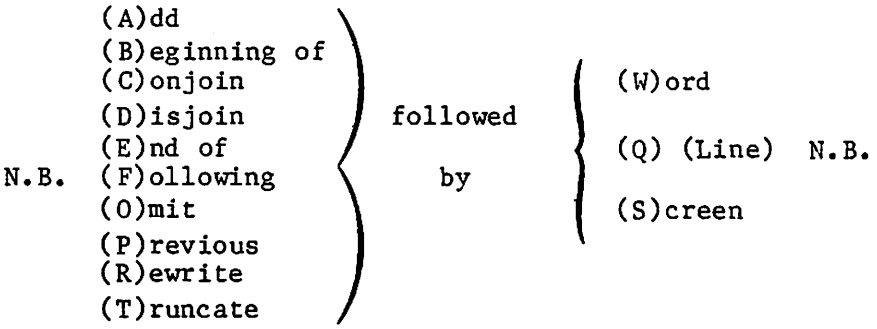
* As indicated above, (_) is used instead of (L).
* Although the terminal has a line width of 80 characters, SCREED (for good reasons) does not use the 80th column of any line, and will ignore any character typed in the 80th column.
* There is a (standard) option on the ADM-3A whereby typing the space bar does not always generate a space character - it merely moves the cursor right. This is not appropriate to the operation of SCREED and should therefore be changed. The person responsible for the terminal should be consulted, as the option switch in question is under the terminal casing. The instructions for making the change are described on page 2-3 of the terminal handbook. The switch in question is located at the top of the left-hand set of toggle switches; it is labelled "SPACE 6 / ADV"; it should be set to SPACE (by default it is set to ADV).

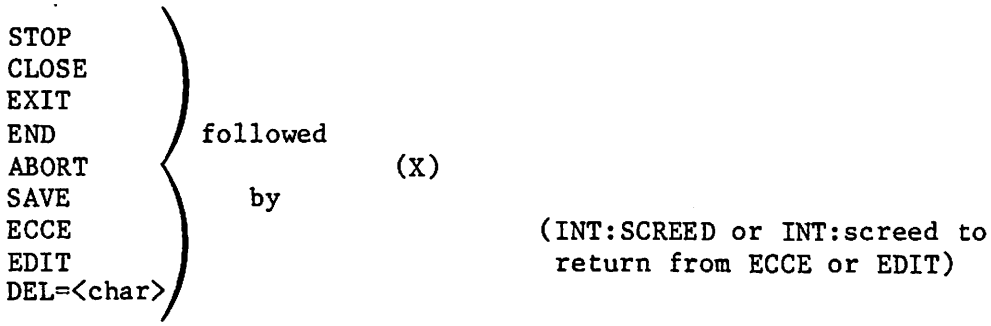
APPENDIX B-4: Lynwood DAD-1

Command summary

Delete character is '^'. It can be changed (see "DEL=" below).

(A) below stands for Ctl A (i.e. hold down the control button and type 'A'). Similarly (B), (C), etc.





You can "thumb" through the file by use of (X) on the left hand margin. (]) in SCREED is the "INT:" character. All editing is lost if you use it.

Notes relating to the Lynwood DAD-1 terminal

* Note that, as stated above, (F) (Following) is used instead of (N) (Next), and that (Q) is used instead of (L) (Line).

* SCREED operates the Lynwood in page mode, which it selects automatically. This affects the operation of (C)(S), (D)(S) and (A)(L), as explained in the description of SCREED.

* The Line Feed key has been disabled, as it has the effect of clearing any line that it causes the cursor to move onto.

* The two switches at the back of the terminal marked St.CR and St.LF should both be down (i.e. off).

* Sometimes on entry to SCREED, the first two lines of the file appear at the bottom of the screen rather than the top. This has to do with the terminal warming up (I think). SCREED will operate as though the two lines - and the cursor - were at the top of the screen. It is therefore best in this situation either to "thumb" or to use (R)(S) or (F)(S).

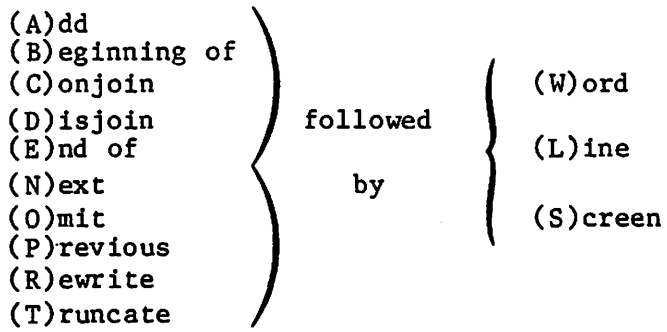
* Although the terminal has a line width of 80 characters, SCREED (for good reasons) does not use the 80th column of any line, and will ignore any character typed in the 80th column.

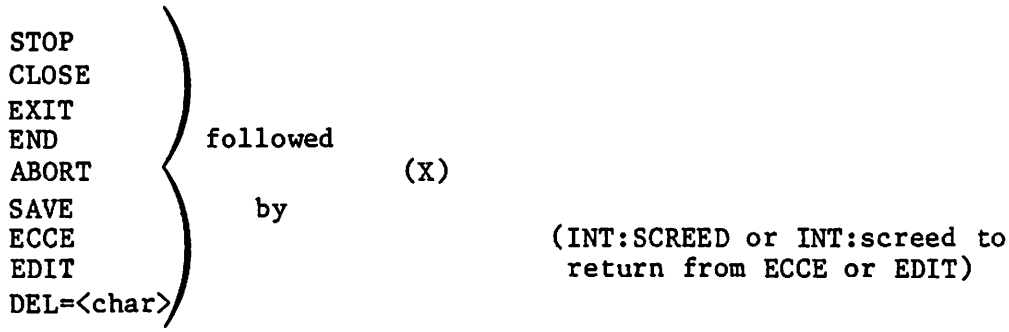
APPENDIX B-5: Visual 200

Command summary

Delete character is '^'. It can be changed (see "DEL=" below).

(A) below stands for Ctl A (i.e. hold down the control button and type 'A'). Similarly (B), (C), etc.





You can "thumb" through the file by use of (X) on the left hand margin.

(]) in SCREED is the "INT:" character. All editing is lost if you use it.

Notes relating to the Visual 200 terminal

* The Visual 200 terminal has a large number of features and settings; each is activated by the user generating - either explicitly or by use of a function key - a string of characters starting with ESC. SCREED can detect when this has been done but unless the feature happens to be one that SCREED uses (such as "Clear screen" or "Clear rest of line"), it will generate an error message and rewrite the screen. This however might not be enough, and to ensure the proper working of SCREED thereafter the user might have to reset the terminal; e.g. if he had issued the command to "clear all tab settings" SCREED would not be aware of this and would operate as though the default tab settings still applied.

* The "typo-matic" keys of this terminal make it particularly suitable for use with a screen editor.

APPENDIX B-6: Pericom 6801

Command summary

Delete character is '^'. It can be changed (see "DEL=" below).

(A) below stands for Ctl A (i.e. hold down the control button and type 'A'). Similarly (B), (C), etc.

(A)dd
(B)eginning of
(C)onjoin
(D)isjoin
(E)nd of
(N)ext
(O)mit
(P)revious
(R)ewrite
(T)runcate

followed by

(W)ord
(L)ine
(S)creen

STOP
CLOSE
EXIT
END
ABORT
SAVE
ECCE
EDIT
DEL=<char>

followed by (X)

(INT:SCREED or INT:screed to return from ECCE or EDIT)

You can "thumb" through the file by use of (X) on the left hand margin.

(]) in SCREED is the "INT:" character. All editing is lost if you use it.

Notes relating to the Pericom 6801 terminal

* SCREED requires the terminal to be in page mode. The user must set the page switch, beside the screen, at some stage before replying to SCREED's initial question about the terminal type.

* The Auto CR/LF and Auto LF wheels at the back of the terminal should both be rotated upwards to the limit, i.e. switched off.

* The terminal has arrow keys, on a separate pad. Unfortunately, although they move the cursor around the screen, they do not generate any codes and so SCREED is unaware that the cursor has been used. In other words, DO NOT USE THE ARROW KEYS.
(The cursor has to be moved by the use of Tab, Backspace, Return, Line Feed and the appropriate SCREED commands, the (W) commands in particular.)

* Although the terminal has a line width of 80 characters, SCREED (for good reasons) does not use the 80th column of any line, and will ignore any character typed in the 80th column.