UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 35

(June 1985)

Title:

EMAS 2900 Subsystem: data structures and 'hash commands'

Author: Neil Hamilton-Smith
John Wexler et al.

Contact: Advisory service

Software Support Category: See Note 15

Synopsis

This Note covers various aspects of the Edinburgh Subsystem on EMAS 2900. It is a selection of notes about those features which are most likely to be useful to the experienced user who wants to know more than is in the EMAS 2900 User's Guide. It deals with (among other things) the so called 'hash commands', and how to understand the information which the hash commands can provide for you. Much of what is described is not formally supported, but it is unlikely to be changed significantly during the life of the EMAS 2900 service. Most EMAS users do not need to use the information.

Keywords

COMREG, file definitions, file headers, hash commands, partitioned file index, registers

Contents

Edinburgh Regional Computing Centre

James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ. Telephone 031-667 1081

1 File Headers

In most files (and members of partitioned files) the first 32 bytes are reserved for a 'header'. This is laid out as 8 words, as follows:

FIRST WORD — Limit of information in file. This is a count of bytes, starting with the first byte in the file - i.e., the first byte of the header. No byte beyond this limit contains any useful information at all - neither data nor 'red tape' such as a partitioned file index.

SECOND WORD — Start of data. The offset (counted in bytes) of the first byte of data in the file, starting from the first byte of the file. The space between the start of the file and the start of data is used to hold the file header, and there are no other standard uses for it. Since a standard header is 32 bytes long, this word almost always contains the value 32.

THIRD WORD — Limit of space for file. This is the total size (in bytes) of the area allocated to hold the file. The information in the file - data, header and red tape - can be altered and expanded, provided they do not exceed this limit, without requesting the allocation of more space to hold the file. This size is a multiple of 4096 since file space is allocated in units of 4096 bytes (E-pages).
N.B. This word does NOT contain a useful value in a member of a partitioned file.

FOURTH WORD — File Type

- 0 - Character file (obsolete value)
- 1 - Object file
- 2 - Old format directory of object code
- 3 - Character file
- 4 - Data file
- 5 - Corrupt object file
- 6 - Partitioned file
- 8 - Journal file
- 9 - OPTION file

FIFTH WORD — Not used.

SIXTH WORD — Date and time last altered (see User Note 10).

SEVENTH WORD — Use depends on file TYPE. This may be of interest in data files (i.e., with TYPE = 4 in fourth word). For these files, it gives the FORMAT.

The least significant 2 bits are

1. for a file of fixed-length records
2. for a file of variable-length records
3. for an unstructured file

"Store mapped files" have TYPE = 4 and FORMAT = 3.
Direct access files have TYPE = 4 and FORMAT = 1.

The bit X'00000020' may be set in FORMAT to indicate that it contains EBCDIC characters.

The 16 most significant bits of format give the record length (in bytes) for fixed-length records, or the maximum record length for variable-length records. They are not used for unstructured files.

The seventh word is also significant in partitioned files (TYPE = 6). For those, it is an offset, relative to the first byte of the file (i.e., of the header), counted in bytes, which points to the first byte of the partitioned file's index.

For object files (TYPE = 1) the seventh word is the offset in bytes of the start of the "load data" which is not described in this Note.

In directory files (TYPE = 2), the seventh word is also an offset in bytes relative to the first byte of the (header of the) file, which points to the start of the "PLIST" of the directory. This Note does not describe the structure of directories.

For all other file types, the seventh word is either not used or else unlikely to be of interest.

EIGHTH WORD For a structured data file (TYPE = 3, FORMAT = 1 or 2) this gives the number of records in the file.

For a partitioned file (TYPE = 6) this is the number of members. It is also the number of entries in the index.

For object files (TYPE=1) this is the offset in bytes of the "object file map" which is not described in this Note.

For all other types of file, it is unlikely to be of interest.

1.1 Partitioned file index

The seventh word of the partitioned file's header points to the index. The eighth word indicates how many members there are. The index has exactly one entry for each member; the entries are laid out consecutively; there are no spare or empty entries.

Each entry is 32 bytes long.

The first four bytes are an integer which is the offset, in bytes, relative to the first byte of the (header of the) file, of the first byte of the header of the member.

The next twelve bytes contain a string, in normal IMP format, of ISO characters. This is the member's name. The first byte of the string is an integer I where $0<I<11$: I is the length of the string: the characters of the string occupy the next I bytes. If I is less than 11, then the remainder of the 12 bytes are not used.

The last sixteen bytes of the entry are not used.

Each member of a partitioned file is laid out exactly like a file, with a header (in the format already defined) in its first bytes. In the header of a member, offsets are given relative to the start of the header of the member, not relative to the start of the containing partitioned file. Space is allocated for members in small units, so that they do not have to occupy a multiple of 4096 bytes. The third word in the header of a member is not significant and does *not* indicate the size of the member.

2 File definitions

For every i/o stream which has been DEFINEd, there is a record of the following format:

```
        %RECORD %FORMAT                                                    %C
FDF(%INTEGER LINK, DSNUM,                                                  %C
        %BYTEINTEGER STATUS, ACCESSROUTE, VALID ACTION, CUR STATE,         %C
        %BYTEINTEGER MODE OF USE, MODE, FILE ORG, DEV CODE,                %C
        %BYTEINTEGER REC TYPE, FLAGS, LM, RM,                              %C
        %INTEGER ASVAR, AREC, RECSIZE, MINREC, MAXREC, MAXSIZE,            %C
        LASTREC, CONAD, CURREC, CUR, END, TRANSFERS, DARECNUM,             %C
        CURSIZE, DATASTART, %STRING (31) IDEN,                             %C
        %INTEGER KEYDESC0, KEYDESC1, RECSIZEDESC0, RECSIZEDESC1,           %C
        %BYTE %INTEGER F77FLAG, F77FORM, F77ACCESS, F77STATUS,             %C
        %INTEGER F77RECL, F77NREC, IDADDR,                                 %C
        %BYTE %INTEGER F77BLANK, F77UFD, SPARE1, SPARE2)
```

The record corresponding to stream N can be located by using the %SYSTEM %INTEGER %FN FDMAP (%INTEGER CHAN), thus: FDMAP(N) will return the address of the relevant record, if there is one, and otherwise it will return zero. The record can also be examined from command level, by using the #LISTFD command (see Section 4.3.5 below). This will print out any of the following fields which are non-zero:

LINK
DSNUM
STATUS
ACCESS ROUTE
VALID ACTION
CUR STATE
MODE OF USE
MODE
FILE ORG
DEV CODE
RECTYPE
FLAGS
ASVAR
AREC
RECSIZE
MINREC
MAXREC
MAXSIZE
LASTREC
CONAD
CURREC
CUR
END
TRANSFERS
DARECNUM
CURSIZE
DATASTART

and it will also print the %STRING IDEN.

LINK Used internally by the Subsystem.

DSNUM — Stream number.

1 to 80 can be defined by the user.
81 - input stream used by standard commands for reading control data, etc.
82 - output stream used by standard commands when user requests output to a file.
83 - default input stream via a 'pipe' (not available in standard subsystems).
84 - default output stream via a 'pipe' (not available in standard subsystems).
87 - output stream used by compilers for error listings.
88 - default input while in OBEY.
89 - default output while in OBEY.
90 - normal default input stream and command input stream.
91 - normal default output stream (i.e., console output or job journal).

SELECT INPUT for stream 0 or 98 is taken to refer to the current default input stream, whichever that may be, and NOT to any file definition record which may exist for stream 98 (there cannot be one for stream 0).
FORTRAN READs on logical unit 5 refer to the current default input stream if there is no file definition record for channel 5.
Similarly, SELECT OUTPUT for stream 0, 99 or 107 refers to the current default output stream, and FORTRAN WRITEs to logical unit 6 use the current default output stream if channel 6 is not defined.

STATUS — 0 - not OPENed; 3 - OPENed.

ACCESS ROUTE
1 for .IN
2 for .OUT
3 for store-mapped file
5 for magnetic tape
6 means "use definition for stream ASVAR"
8 for file
9 for console
10 for .NULL
11 for alien data in background JCL

VALID ACTION
1 read
2 write
4 rewind
8 backspace
16 endfile
32 close
64 seek

Normally several actions will be valid, and VALID ACTION will be the sum of the values corresponding to those actions.

CUR STATE	0 - CLOSEd 1 - last action was OPEN 2 - last action was READ 3 - last action was WRITE 4 - last action was REWIND 5 - last action was BACKSPACE 6 - last action was ENDFILE 7 - end-of-file detected by last attempted READ
MODE OF USE	1 - character; 2 - sequential; 3 - direct access; 13 - FORTRAN direct access.
MODE	Set to 11 by DEFINE; otherwise not used.
FILE ORG	Not used.
DEV CODE	0 for .NULL or .TEMP 127 for .LP For a device, SPOOLER's queue identification number (may be increased by 128 to indicate 'special treatment').
RECTYPE	1 - Fixed length (includes Direct access) 2 - Variable length 4 - Character
FLAGS	8 - Append data (-MOD) 16 - Insert format effectors for printer 32 - EBCDIC 64 - Write permit ring needed (for magnetic tape).
ASVAR	For FORTRAN(E) direct access, this is the 'associated variable'. If ACCESS ROUTE=6, this is the number of another definition to be used instead of the present one.
AREC	Used internally by the Subsystem.
RECSIZE	Size of current record.
MINREC	Minimum record size in bytes.
MAXREC	Maximum record size in bytes.
MAXSIZE	Maximum size in bytes to which file may expand.
LASTREC	Used internally by the Subsystem (normally, the address of the last record transferred).
CONAD	Connect address of file.
CURREC	Address of current record, or of first character of current line.
CUR	Address of next byte to be read or written.
END	Address of first byte beyond end of data in file.
TRANSFERS	Count of transfers since file was OPENed.

DARECNUM	Used internally by the Subsystem. (Normally the number of the record most recently transferred in a direct access file.)
CURSIZE	Current size of file, from start of header to end of data, in bytes.
DATASTART	Start address of data in file.
IDEN	File name.

3 Use of COMREG

The Edinburgh Subsystem declares an

```
%OWNINTEGERARRAY SSCOMREG(0:60)
```

which is generally accessed via the

```
%SYSTEMINTEGERMAP COMREG(%INTEGER I)
```

The uses of the elements of this array are given herein. An array and map of the same name are used for a similar purpose by the Scientific Jobber; all such use is also recorded here, and the Jobber is mentioned explicitly in the relevant cases.

COMREG(1)

Used for 'SSLEVEL' in the Scientific Jobber. Values for this entry are as follows:

-1 - Aborting
0 - Initialising
1 - Command processor
2 - Trusted facility (e.g. a compiler)
3 - User program

Not used by the Edinburgh Subsystem.

COMREG(2)

Used to indicate batch termination in the Scientific Jobber. Set up in module DIAG22, but never actually accessed.

Not used by the Edinburgh Subsystem.

COMREG(3)

Head of chain of loaded procedure entries in the Scientific Jobber.

Not used by the Edinburgh Subsystem.

COMREG(5)

Head of chain of unsatisfied procedure references in the Scientific Jobber.

Not used by the Edinburgh Subsystem.

COMREG(7)

Count of unsatisfied references; used by Edinburgh Subsystem loader.

In the Scientific Jobber, head of chain of unsatisfied data references.

COMREG(8)

Set up by Subsystem loader - ADDR(BASE(1)) for use by simulator.

COMREG(9)

Used by the BCPL compiler for internal communication.

Since %SYSTEMROUTINE COMPILE does not pass a parameter string to the compiler, this entry is used to hold the address of any auxiliary parameter string generated by the BCPL command itself.

COMREG(10)

"Monitor called" flag; used by the CLI module of the Job Control package.

COMREG(11)

Address of the public table used for EBCDIC to ISO code conversion. This table is 256 bytes long, and is in a form suitable for use by the hardware 'table translate' operation.

Used for the same purpose in the Scientific Jobber.

COMREG(12)

Address of the public table used for ISO to EBCDIC code conversion. This table is 256 bytes long, and is in a form suitable for use by the hardware 'table translate' operation.

Used for the same purpose in the Scientific Jobber.

COMREG(13)

Used by the BASIC compiler. Contains the address of the table within the Subsystem which is used by the ICL mathematical library routines, to enter the language-dependent diagnostic routines. BASIC alters the descriptor in this table to point to its own routine, and restores the previous value (pointing to the ICL MATHS ERROR ROUTINE in the Subsystem) when it exits.

Note that *Int*:A, etc. are intercepted by BASIC; this guarantees that the old value is always restored on exit.

COMREG(14)

The address at which the compiler workfile (T#WRK) is connected.

Used for a similar purpose in the Scientific Jobber.

COMREG(15)

Used to set up CODEBASE in LPUT, in both the Edinburgh Subsystem and the Scientific Jobber. However, this variable is never used for any meaningful purpose.

COMREG(17)

Used by the Scientific Jobber to hold the address of the file descriptor map (FDMAP).

Not used by the Edinburgh Subsystem.

COMREG(19)

Used by the Scientific Jobber (EFILE).

Used by VME 2900 to point to a list of file definition tables.

Also used by magnetic tape support software: count of tapes allocated for utility use.

Not used by the Edinburgh Subsystem.

COMREG(21)

Used by the Scientific Jobber for the address of BASICFDS(0) in module EFILE.

Not used by the Edinburgh Subsystem.

COMREG(22)

Current input channel number. Used for the same purpose in the Scientific Jobber.

COMREG(23)

Current output channel number. Used for the same purpose in the Scientific Jobber.

COMREG(24)

Subsystem 'return code'. This is set to indicate the success or failure of a subsystem command. Non-zero values have the usual meanings as Subsystem

error codes. A value of zero indicates success. This entry may be set by a call of

%EXTERNALROUTINE SET RETURN CODE(%INTEGER N)

and examined by use of

%EXTERNALINTEGERFUNCTION RETURN CODE

if required.

COMREG(25)

If this item is non-zero, then when an error is detected within the subsystem itself, diagnostics include routines within the subsystem rather than just the calling user-level routines.

Used for a similar purpose in the Scientific Jobber.

COMREG(26)

Used by various OMF (ICL object module format) processing utilities, to pass options from command level. Bits in this entry are set and cleared by the command OMFPARM.

Also used for option flags in the Scientific Jobber.

Meaning of option bits

2**0 - MAP
2**1 - FIXUPS
2**2 - MAXKEYS
2**3 - NOCASCADE
2**4 - LIBPROC
2**5 - SHARE

2**21 - NOLOCALS

2**24 - __CODE properties
2**25 - /

2**26 - __GLA properties
2**27 - /

2**28 - __SST properties
2**29 - /

2**30 - __STACK properties
2**31 - /

Meaning of property bits

0 - DENSE
1 - LOCAL
2 - SPARSE
3 - SERIAL

COMREG(27)

Part of current PARM settings. Used for the same purpose in the Scientific Jobber.

Meaning of bits in COMREG(27)

2**0 - QUOTES
2**1 - NOLIST
2**2 - NODIAG
2**3 - STACK
2**4 - NOCHECK
2**5 - NOARRAY
2**6 - NOTRACE
2**7 - PROFILE
2**8 - IMPS
2**9 - INHIBIOF
2**10 - ZERO
2**11 - XREF
2**12 - LABELS
2**13 - LET
2**14 - CODE
2**15 - ATTR
2**16 - OPT
2**17 - MAP
2**18 - DEBUG
2**19 - FREE
2**20 - DYNAMIC
2**21 - 'Diag stream set'
2**22 - EBCDIC
2**23 - NOLINE
2**24 - 'Stack size set'
2**25 - Not used
2**26 - PARMZ
2**27 - PARMY
2**28 - PARMX
2**29 - MISMATCH
2**30 - 'Jobber mode'
2**31 - Not used

COMREG(28)

Part of current PARM settings. Used for the same purpose in the Scientific Jobber.

Meaning of bits in COMREG(28)

2**0 - F77 use on VME; 0 on EMAS
2**1 - I8
2**2 - L8
2**3 - R8
2**4 - OPTEXT
2**5 - NOCOMMENTS
2**6 - NOWARNINGS
2**7 - STRICT
2**8 - MAXDICT
2**9 - Not used
2**10 - Not used
2**11 - Not used
2**12 - Not used
2**13 - Not used
2**14 - MINSTACK
2**15 - Not used
2**16 - \
2**17 - \ Diagnostic level
2**18 - / F77 on VME;
2**19 - / 0 on EMAS
2**20 - OPT1
2**21 - OPT2
2**22 - OPT3
2**23 - OPT4
2**24 - \
2**25 - \
2**26 - \
2**27 - \ F77 use on VME;
2**28 - / 0 on EMAS
2**29 - /
2**30 - /
2**31 - /

COMREG(29)

Address of COMREG(0) in the Scientific Jobber.

Not used by the Edinburgh Subsystem.

COMREG(30)

Head of virtual file chain in the Scientific Jobber.

Not used by the Edinburgh Subsystem.

COMREG(31)

In VME 2900 it is initialized to the area handling data area.

COMREG(33)

Used by the Scientific Jobber; holds the address of SIGDATA(0) in module DIAG.

Not used by the Edinburgh Subsystem.

COMREG(34)

Signal level.

Used for the same purpose in the Scientific Jobber.

COMREG(35)

Address of the start of the base GLA.

Used for a similar purpose in the Scientific Jobber.

COMREG(36)

Saved LNB for contingency trapping.

Used for the same purpose in the Scientific Jobber.

COMREG(37)

Address of the base of the auxiliary stack in the Scientific Jobber.

Not used by the Edinburgh Subsystem.

COMREG(38)

Address of the base of the user GLA (T#UGLA).

Used for the address of SS#GLA in the Scientific Jobber.

COMREG(39)

Used by new loader to hold LOADPARM settings.

COMREG(40)

Set by %SYSTEMROUTINE COMPILE to the channel number which the compilers are to use for their error message file. A value of -1 indicates that no error messages are required by the user, save those which are placed in the compiler listing file.

An additional indication about the error channel is given by a bit in COMREG(27). If set, it indicates that the contents of COMREG(40) are valid for use as an output channel.

COMREG(41)

Address of a descriptor to the auxiliary stack (T#AUXST).

Used for the same purpose in the Scientific Jobber (SS#AUXST).

COMREG(42)

Used in FORTRAN I/O module to set value of variable OPEH MODE during FIO initialisation. Values are as follows:

-1 - Jobber mode
 0 - Edinburgh Subsystem mode
 1 - OPEH mode (VME systems)

COMREG(44)

Contains the address of the first free byte in the user GLA; manipulated by LOAD and UNLOAD.

COMREG(45)

Used for DAP Fortran. Holds the DAPARM word, analogous to COMREG(27) and (28) for PARM.

DAP PARM Settings (COMREG (45))

BIT	VAL	PARM	EFFECT	AFFECTS
0	1	MAP	Give Load map	LOADER
1	2	REGS	Give DAP registers on completion	LOADER
2	4	DIAG	Disable diagnostics	LOADER
3	8	DUMP	Enable DPB dumping	DUMMY DIRECTOR
4	X'10'	NOLIST	Produce no listings	FORT, ASS, CON
5	X'20'	NOSOURCE	Do not produce source listing	FORT, ASS, CON
6	X'40'	XREF	Produce XREF listing	FORT, ASS
7	X'80'	ERL	Produce External Reference Listing	FORT, ASS
8	X'100'	ATTR	Produce attribute listing	FORT
9	X'200'	OBJECT	Produce AMF listing	FORT
10	X'400'	DIRECTIVES	Source directives in listing	FORT
11	X'800'	NOCODE	Do not produce code	FORT, ASS, CON
12	X'1000'	AMF	Compiler produces only AMF ASSEMBLER assumes AMF, not APAL	FORT, ASS
13	X'2000'	NORTCHECKS	No Run Time checks performed	FORT
14	X'4000'	NODOLOOPS	DO loop checks not performed	FORT
15	X'8000'	NONORMALISATION	Normalisation checks not performed	FORT
16	X'10000'	NOSUBSCRIPTS	Subscripts checks not performed	FORT
17	X'20000'	LISTALL	Full listings	FORT, ASS, CON
18	X'40000'	ISO	Produce AMF in ISO	FORT
19	X'80000'	LOADMON	Switch on Load monitoring	LOADER

COMREG(46)

Set by %SYSTEMROUTINE COMPILE to the connect address of the compiler source file. If the compiler source is not a file, this item is set to zero; in this situation, the compiler should use the currently selected input channel.

COMREG(47)

Number of statements or faults; set by compilers.

COMREG(48)

Stack limit. Initialised to X'3D000' in the Edinburgh Subsystem. The validity of this value is indicated by a value of 1 in a certain bit in COMREG(27).

Used for the same purpose in the Scientific Jobber.

COMREG(49)

Used for the address of the transfer count record in the Scientific Jobber.

Not used by the Edinburgh Subsystem.

COMREG(50)

Used by DAP software.

If the left-hand (most significant) bit is set, then the DAP has been used by this process in the current session, and the rest of the word is a cumulative total of DAP time used at the start of the current session. It is used as a base figure which METER then subtracts from the current amount.

In VME 2900 it points to the array of CALL descriptors, and to the various language dependent diagnostic routines.

COMREG(51)

In VME 2900 it is set to the number of array elements per array to be printed during diagnostics.

COMREG(52)

Set by %SYSTEMROUTINE COMPILE to the address of the name of the compiler output object file. This name is held as a normal IMP string; the pointer is to the zeroth (length) byte of this string.

COMREG(53)

Reserved for development use by the ERCC Compiler Group.

COMREG(54)

Reserved for development use by the ERCC Compiler Group.

COMREG(55)

Reserved for development use by F77PARM.

COMREG(56)

Reserved for development use by F77PARM.

COMREG(57)

Used by LPUT; if this entry is non-zero, it is used as an address at which to store the name of the main entry point (as an IMP string) or, failing that, the first other entry point in the object file being generated.

COMREG(58)

Contains the address of the %OWN variable MAINDR1 in the Edinburgh Subsystem; needed for special versions of the RUN command which need to load and enter programs themselves.

COMREG(59)

Used by the Scientific Jobber; address of COMPS(1) in module EFILE.

Not used by the Edinburgh Subsystem.

COMREG(60)

Used by LPUT; bit 2**1 in this entry is set to indicate the presence of a main entry point.

4 The EMAS Hash Commands

4.1 Introduction

The commands described in this Note are provided for low-level diagnostic purposes; as such, they are of interest mainly to anyone concerned with low-level facilities in the Edinburgh Subsystem.

A few commands are of wider interest; for instance, those concerned with loader monitoring and timing.

These commands have no formal support category and so could (in principle) be changed or withdrawn without notice, and there is no guarantee that errors in them would be corrected. However, they have been in use for some time and have proved to be useful.

4.2 Command Structure

Each command has a '#' as its first character. Note that 'hash commands' (as they are known) do not invoke the loader, and there is no one-to-one correspondence between each hash command and an external routine. This means that hash commands cannot (usually) be called from programs, and cannot be aliased (but see Section 4.5 below).

The parameters to a hash command often include numeric constants. These can be typed as decimal numbers in the normal way, or as hexadecimal numbers, in which case they must be preceded by an 'X'. Commands may be typed in upper or lower case, or a mixture of both. For example, the following two commands would have the same effect:

```
#SNAP X840000,256

#snap x840000,x100
```

Note that, in all relevant cases, lengths are expressed in bytes.

4.3 The EMAS Hash Commands

The commands are grouped according to their function, and those that are most likely to be of general interest are described first. All the examples assume that OPTION(NOBRACKETS) has been selected.

4.3.1 Representation of integers

#DEC Xnn
Prints the decimal equivalent of the hex number 'nn'; the 'X' is vital.

```
Command:#DEC XFF
N= 255
```

#HEX nn
Prints the hexadecimal equivalent of the decimal number 'nn'.

```
Command:#HEX 255
X=000000FF
```

4.3.2 Monitoring commands

#MON
Sets a flag so that the CPU time and Pageturns used by subsequent commands are printed on completion of each command.
(Cancel #MON with #N.)

Example:

```
Command:#MON            (Sets mon CPU and Pageturns bit).
 2 MS 36 PT

Command:USERS
Users = 67
 4 MS 39 PT
```

#MONLOAD n
"MONitor the LOADer", at a depth 'n' from 1 to 31. Prints information about all the object modules being loaded to satisfy a command, with store locations and timings. Currently the lowest 5 bits of n are significant:

- 2**0 - requests minimal loading information and some important but non-critical warnings.
- 2**1 - request information on object files which are being loaded and unloaded and on the location and layout of areas in loaded files.
- 2**2 - request information on names and locations of code and data entry points as they are loaded and information on common areas set up by the loader.
- 2**3 - requests information from the loader search module on which entry points are being sought, which directories are being searched, how aliases are handled, etc.
- 2**4 - requests information on which unsatisfied references are being made dynamic when LOADPARM(MIN) is set. LOADPARM(MIN) suppresses cascade loading and the loader will only load the file which contains the required entry point. Any common areas required are created and all unsatisfied references are made dynamic.

In using #MONLOAD it is generally better to use integer parameters such as 1,3,7,15,31, which have successively more bits set, rather than values such as 2,4,8,16 in which only one bit is set. This is because some information given by higher bits amplifies or expands that given at lower bit setting and the information is no longer seen in context. Note that #MONLOAD(-1) will generate all possible monitoring. Loader diagnostic monitoring settings will remain in force until another call on #MONLOAD.

#MONLOAD(0) or #N will turn off monitoring.

Failure messages and some critical warnings are always generated regardless of the #MONLOAD settings.

Examples:

```
Command:#MONLOAD 1


Command:HELP CHERISH
0.00        HELP Found in SUBSYS.HELP21Y
0.08        S#ZVIEW Found in SUBSYS.NEWVIEWY
0.14        S#ZCOPY2 Found in SUBSYS.COPY2_COPY2
0.19        VDUI Found in SUBSYS.SCREENCPY
0.25        READPROFILE Found in SUBSYS.SYSTEM_PROFILEY
0.26        ENTER called
Searching on "*CHERI*"
[screenful of information]


Command:#MONLOAD 7

Command:LAYOUT LI/LO
0.00        LAYOUT Found in ERCLIB.GENERALY_ELAYOUT
0.00        Starting to load ERCLIB.GENERALY_ELAYOUT
0.06        CODE      01049A70      00003858
0.06        GLA       00EC0000      00000110
0.06        SST       0104D2C8      00000760
0.06        UST       00EC0110      00000208
0.07        Code Entry LAYOUT at 01049AB0
0.07        Finished loading ERCLIB.GENERALY_ELAYOUT
0.07        ENTER called
```

4.3.3 Errors

#PMESS fn
Prints the Subsystem Error Message for fault number 'fn'.

Examples:

```
Command:#PMESS 103
 Negative sign incorrect

Command:#PMESS 104
 Invalid format

Command:#PMESS 105
 Decimal field too wide

Command:#PMESS 20
 Array inside out

Command:#PMESS 21
 No result

Command:#PMESS 22
 Param not destination
```

4.3.4 Obtaining information from the Virtual Memory

#PVM
Prints the layout of Virtual Memory, i.e. the table of connected files together with their connect address, read/write mode, size, and FSYS location.

Example:

```
Command:#PVM
SEG HOLE CONAD      MODE USE  K   FSYS      FILE
32    2  00800000     0    8  372   0       S#DISC.SITEX380
34    1  00880000     0    8  256   0       EKLD91.T#BGLA
35    4  008C0000    19    1   12   10      EKLD91.T#LOAD0
39    1  009C0000     1    0    4   10      EKLD91.SS#OPT
40    1  00A00000    19    8    4   10      EKLD91.T#IT0
41    1  00A40000     3    8   64   10      EKLD91.SS#JOURNAL
43    1  00AC0000     1    0    4   10      EKLD91.SF
44    1  00B00000     1    1    4   10      EKLD91.SS#DIR
45    1  00B40000     1    2  172   28      SUBSYS.SYSTEM
46    1  00B80000     1    1    8   64      CONLIB.GRAPHICS
47    1  00BC0000     1    1   12   24      ERCLIB.GRAPHICS
48    1  00C00000     1    8   16   64      PLULIB.PACKDIR
49    1  00C40000     1    1    4   24      ERCLIB.SORTMERGE
50    1  00C80000     1    1    4   29      KNTLIB.GENERAL
51    1  00CC0000     1    1    4   24      ERCLIB.GENERAL
52    1  00D00000     1    1   32   64      CONLIB.GENERAL
53    2  00D40000     1    0  424   24      ERCLIB.GENERALY
55    4  00DC0000    19    1   64   10      EKLD91.T#UGLA0
59    1  00EC0000    19    1  256   10      EKLD91.T#ASTK0
118   2  01D80000   147    1  252   10      EKLD91.T#USTK0
```

See Section 6 below for an explanation of this table.

#DUMP addr, length, out
Dumps the specified area of Virtual Memory to file/device 'out' (.LP by default). 'addr' and 'length' can be in either hexadecimal (preceded by X) or decimal form.

Example:

Command:#DUMP X00900000,60;FILENAME

Command:LIST FILENAME,.LP23

If you want to dump small areas use #SNAP (see below) or Supersnap (described in User Note 36) since their output looks better on a terminal.

#SNAP addr,length
Dumps the specified area of Virtual Memory to .OUT.

Command:#SNAP XD985B0,64

```
(00D985B0)  020002D0 08544553 5446494C 45000000
(00D985C0)  000185AC 020002C8 0A4D4552 47454649
(00D985D0)  4C455300 000185C0 020002A8 08534F52
(00D985E0)  5446494C 45000000 00000000 020002C0
```

#SNAPCH addr,length
Translates the specified area of Virtual Memory into ISO and prints it on .OUT.

```
Command:#SNAPCH XD985B0,64

(00D985B0)        TESTFILE            MERGEFI
(00D985D0)   LES          SORTFILE
```

#SNAPCODE addr,length,out
Decompiles code from the specified area into a form of 2900 assembly language and prints it on device/file 'out'.

```
Command:#snapcode x00900020,16

00020      81818181
00024      81818181
00028          7E84    LXN       (LNB + 4)
0002A          1804    PRCL      4
0002C          1C04    JLK       TO  X'2E'
0002E          5D98    STLN      TOS
```

4.3.5 Obtaining information about files

#CONNECT file
Connects 'file', in WRITE mode if possible, otherwise in READ mode, and prints the connect address as a hexadecimal number. If 'file' is a member of a Partitioned file, connection is always in READ mode.

```
Command:#CONNECT FFCO
FFCO CONNECTED IN WRITE MODE AT 00900000
```

Remember that for most files the first 32 bytes will be a header, and the actual data of the file will start 32 bytes further on from the address returned by #CONNECT: in this example, at X00900020. See Section 1 above for a description of file headers.

#DUMPFILE file,offset,length,out
Dumps the specified area of 'file' to file/device 'out' (.LP by default). 'offset' and 'length' can be in either hexadecimal (preceded by X) or decimal form.

Example:

```
Command:#DUMPFILE FFCO,0,60,FILENAME

Command:LIST FILENAME,.LP23
```

You might find the utility Supersnap (described in User Note 36) helpful for examining a file.

#LISTFD n
Prints all known facts about the file or device DEFINEd on channel 'n'.

Example:

```
Command:DEFINE 1,sf
Command:#LISTFD 1
DSNUM:          00000001          1
ACCESS ROUTE:00000008             8
VALID ACTION:  0000007F         127
MODE:          0000000B          11
RECTYPE:       00000002           2
MINREC:        00000001           1
MAXREC:        00000400        1024
MAXSIZE:       00040000      262144
IDEN:      SF
```

See Section 2 above for an explanation of these lines of output.

4.3.6 Altering values within the Virtual Memory

#SBYTE addr,val
Sets the byte at address 'addr' to the value 'val'. 'addr' and 'val' can be in either hexadecimal (preceded by X) or decimal form.

#SSTRING addr,"string"
Sets the string at the specified address to the given string, which *must* be enclosed in double quotes.

#SWORD addr,val
Sets the word at address 'addr' to the value 'val'.

The utility Supersnap, described in User Note 36, is useful for altering values within the virtual memory.

4.3.7 Contents of Registers

#REGS n
Prints the contents of the CPU registers at the time of the most recent failure if 'n' is omitted, otherwise at one of the three previous failures, as 'n' is -1, -2 or -3.

Example:

```
Command:#REGS
CONTINGENCY AT 11.07.30
CLASS: 00000004
SUBCLASS: 00000000
LNB: 01D9904C
PSR: 00A04009
PC: 011800D4
SSR: 05800000
SF: 01D99074
IT: 0000C857
IC: F6000000
CTB: 00883E10
XNB: 01000000
B: 00000008
DR: 58000000 01D99074
ACC: 00000000 00000000 28000001 00000001
FPC: 011800D2
SPARE: 00000000
```

See Section 5 below for the significance of these registers.

#PCOM n
Prints the contents of location 'n' of COMREG. n must be in the range 1 to 60.

```
Command:#PCOM 35
SSCOMREG( 35)=00880000
```

See Section 3 above for details of COMREGs.

#SCOM n,v
Sets location 'n' of COMREG to value 'v'. For most users, there is only one use of #SCOM which is likely ever to be useful. That is

```
Command:#SCOM 25,1
```

which affects the diagnostic trace printed when a program error or fault is detected. This trace normally includes information about the program or package or user-written command being executed, but not about routines within the Subsystem. Setting COMREG(25) to 1 ensures that the trace of routine calls will include any routines within the Subsystem (but the variables for those routines will not be printed, since the Subsystem is compiled with PARM OPT). If you have diagnostics for a failure which indicate that the program failed on a call to the Subsystem - typically, for example, an i/o statement - and if you need to ask Advisory for help with the problem, then you can use #SCOM 25,1 before trying to reproduce the failure in order to get fuller diagnostics which may be useful for the advisor. The effect of #SCOM 25,1 is cancelled by #SCOM 25,0.

#ACR
Gives the ACCESS CONTROL REGISTER value, or "degree of privilege", for the process: 0=most privileged, 15=least privileged, 10=normal.

```
Command: #ACR
ACR = 10
```

4.3.8 How to stop

#N
Cancels all #MON and #MONLOAD settings.

4.4 Diagnostic aids

If you are afflicted by some problem or failure and you suspect that the diagnostics are inadequate or misleading (and there certainly can be such cases), you can help Advisory by using the # commands to add useful information to the evidence which you bring with your query.

Some useful commands are:

QENV *,filename	to put extensive diagnostics into that file, which you may then LIST to a printer (see User Note 27)
#SCOM 25,1	see Section 4.3.7 above.

4.5 Using # commands from a program

%SYSTEM %ROUTINE %SPEC HASH COMMAND(%STRING(255) COMMAND,PARAM) can be used. COMMAND should be the name of one of the # commands but without the prefixed "#". PARAM should be the parameter string which you would type if you were using the command at the terminal.

5 Registers

#REGS prints out the contents of the 2900's hardware registers at the time of the last hardware-detected program failure. #REGS -1 will produce the registers for the previous fault, and so on back to #REGS -3.

The registers are, with their jargon names:

CLASS	} to identify the nature
SUBCLASS	} of the fault
LNB	Local Name Base (stack frame pointer)
PSR	Program Status Register
PC	Program Counter
SSR	System Status Register
SF	Stack Front pointer
IT	Interval Timer
IC	Instruction Counter
CTB	Cross reference Table Base
XNB	Extra Name Base
B	Index accumulator
DR	Descriptor register
ACC	Accumulator
FPC	Failing Program Counter
SPARE	

CLASS values (with SUBCLASS values where significant).

0 Floating point overflow.

1 Floating point underflow - i.e., arithmetic produces a non-zero result

which is too small to be represented in the 2900 floating point binary format.

2 Fixed point (i.e., integer) overflow.

3 Decimal overflow (hardly ever arises, since very few languages apart from COBOL ever exercise the decimal arithmetic facilities of the hardware).

4 Division by zero attempted.

5 Bound check - typically, "array bounds exceeded" or similar faults. SUBCLASS is significant, and its values are given here *for reference only* with no attempt to explain the jargon.

- 0 operand accessed via a descriptor with modifier, descriptor does not have "bound check inhibit" bit set, modifier is not less than the "bound" field in the descriptor.
- 1 MODD (modify DR) instruction attempted with operand not less than bound field of DR.
- 2 to 7 DVM (dope vector multiply) instruction attempted -
  - 2 (index - lower bound) gives overflow
  - 3 (index - lower bound) is negative
  - 4 multiplier is negative
  - 5 upper bound is negative
  - 6 result (displacement) > upperbound or > 2^{31}
  - 7 DR bound becomes negative
  - 8 TCH (Table check) or TTR (table translate) instruction attempted: the value of one of the bytes to be checked or translated is too large for the table supplied.

6 Size error - location too small for operand. Usually arises when an area of store is addressed via a descriptor, subclass is significant.

- SUBCLASS 0 addressed item is too small (e.g., attempt to store the contents of a 32-bit register in a two-byte location)
- 1 addressed item is too large (e.g., attempt to fetch an 8-byte area of store into a one-word register)

7 Overflow in B register.

8 Stack error: subclass is significant

- 0 attempt to take more data off the stack (in the current stack frame) than has been put on it.
- 1 not used.
- 2 attempt to change stack segment by "Load LNB" or "EXIT": typically due to corruption of return linkage information for exit from subroutine.
- 3 attempt to make LNB, the "stack frame pointer", point beyond the limit of data currently on the stack (i.e., LNB > SF): instruction was "Load LNB" or "EXIT".
- 4 "Raise LNB" (used to prepare stack for subroutine entry) used with operand < 0.
- 5 "Raise LNB" used with an operand which would actually decrease the value of LNB.
- 6 "Adjust SF", used to declare or release space for local variables, used to clear the whole of the current stack frame.
- 7 "Adjust SF" attempts to acquire more space than is available in the stack segment.

9 Privilege: various kinds of "address error". These arise when a program tries to access an address which is a valid address in the virtual memory, but the desired kind of access is forbidden by the access protection mechanism. Subclass is significant.

- 0 Attempt to read inaccessible data
- 1 Attempt to overwrite protected data
- 2 Attempt to execute non-executable data
- 3 Attempt to use "image store"
- 4 Attempt to misuse "image store" (can only arise when access to image store is permitted)
- 5 Attempt to evade protection mechanism (detected by the subroutine EXIT instruction)
- 6 Privileged instructions attempted.

10 Descriptor error

- 0 Wrong type of descriptor for "JUMP"
- 1 Wrong type of descriptor for normal operand access
- 2 Wrong type of descriptor for "CALL"
- 3 Wrong type of descriptor for subroutine "EXIT"
- 4 Wrong type of descriptor for "Dope Vector Multiply"
- 5 Wrong length in string descriptor
- 6 Wrong type of descriptor in DR for store-to-store operation
- 7 Wrong type of descriptor in ACC for store-to-store operation
- 8 Wrong type of descriptor for Table Check or Table Translate
- 9 Invalid "size code" in descriptor
- 10 Invalid "subtype" in descriptor
- 11 Attempt to "modify" a system call descriptor
- 12 Not used
- 13 Wrong type of descriptor for Semaphore instruction

11 String operation failure

- 0 Length > bound of DR
- 1 Length > bound of descriptor in ACC

12 Instruction error

- 0 Invalid instruction code
- 1 Literal operand for "store" instruction
- 2 Valid form of address, but not acceptable for a particular instruction
- 3 Attempt to jump from one segment to another using wrong type of jump.
- 4 Invalid operand address.
- 5 Attempt to read an item from the stack segment at a higher address than the current stack top.
- 6 Attempt to fetch data using address format which implies that the data should be in the code segment, but with a virtual address which is in fact in another segment.
- 7 Executed the last instruction in a segment with no provision to jump back or into another segment.

13 Accumulator incompatible with instruction

0 Accumulator size 128 bits for integer or logical operation.
1 Accumulator size 64 bits for "Add/Subtract logical".
2 Accumulator size 128 bits for "Float".
3 Accumulator size 32 bits for "Floating divide double".
4 Accumulator size 128 bits for "Load upper half"
or 32 bits for "Store upper half".
5 Accumulator size 128 bits for "floating multiply double",
or 64 bits for "integer multiply double".
6 Accumulator size not 64 bits when it should contain a descriptor.
7 Attempt to make accumulator size 0 bits.
8 Accumulator size 128 bits for "compress/expand ACC".

LNB or "Local Name Base" or "Stack Frame Pointer" points to the area of storage reserved for the current program or command or routine or function. This area includes

a) information to allow the "calling routine" to be resumed when the "current routine" has finished - this is known as the "return linkage information".

b) an address pointing to an area known as the GLA (General Linkage Area) in which the current routine keeps its "static storage" - e.g., %own variables - and where it may also expect to find information to allow it to call further routines (PLT or Procedure Linkage Table).

c) parameters passed from the "calling routine" to the "current routine".

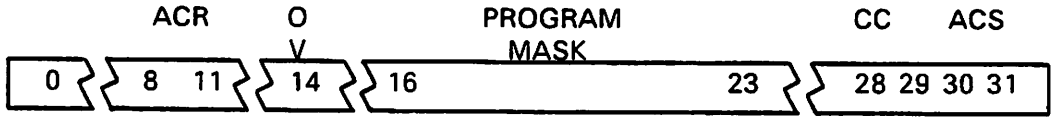
d) addresses of the stack frames of other routines whose variables the current routine may need to use. This is known as the "Display Vector".

e) information for diagnostics and fault trapping.

f) local storage (e.g., for variables which are not %own)

g) work space used in evaluating expressions.

PSR Program Status Register



The 'interesting' fields are:

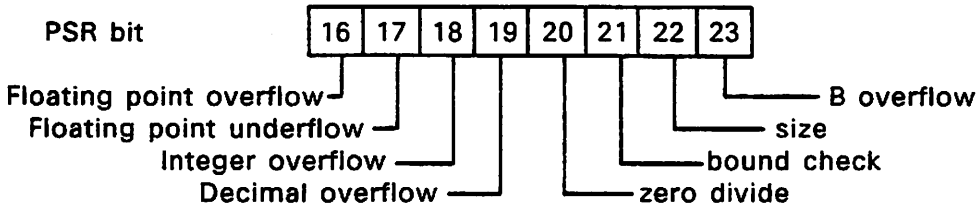
ACR - Access Control Register. Values range from 0 (most privileged) to 15 (least privileged). 10 is normal for user processes on EMAS. The ACR appears as the third digit in the hexadecimal representation of PSR.

OV - non-zero if and only if OVerflow has been detected by a recent instruction. Reset to zero by most arithmetic instructions if overflow is not detected.

PROGRAM MASK

8 bits, each of which corresponds to a program error condition.
If the error occurs and the bit is non-zero, then the normal interrupt action will be inhibited and the error will be ignored.

These bits correspond to program error classes 0 to 7 in order:



The Program Mask appears as the fifth and sixth digits in the hexadecimal representation of the PSR, and a normal value is X'40' - i.e., only floating point underflow will be masked (the effect is that a zero value will be supplied whenever an impossibly small value is generated by floating point arithmetic).

CC - Condition Code - a 2-bit unsigned integer which is set by some instructions to indicate the kind of result achieved, and which can subsequently be tested by conditional jump instructions. In #REGS output, it may reveal something about the result of a recent arithmetic operation or comparison before the failure occurred. It appears in the last digit X in the hexadecimal representation of the PSR: CC is simply X divided by 4.

N.B. The 2900 has a range of conditional jump instructions which do not need to test CC, as well as those which do test it. Hence it is not necessary for all arithmetic operations to set CC, and many of them do not in fact set it. This is quite different from the situation with IBM mainframe architecture (and many other machines) where practically all conditional jumps depend on the condition code, so that all arithmetic operations have to set the condition code. In these architectures, the great majority of condition code settings represent the result of arithmetic operations, and the condition code settings can be made very uniform for those instructions by classifying results as zero, positive, negative or "exceptional". On IBM hardware, for instance, all zero results set a condition code of zero, and all positive results set it to be 1, and so on; and a condition code of one nearly always means that the last arithmetic operation produced a positive result. This is NOT the case with the 2900, since few arithmetic operations set the condition code. A fairly high proportion of those instructions which do set it assign values to represent conditions that would be meaningless for any other instruction. The only uniformity arises with comparison instructions, and for these, typical condition code values are:

0 : equality
1 : register < operand
2 : register > operand
3 : not used

ACS - ACcumulator Size - a 2-bit unsigned integer indicating how many bits of the 128-bit accumulator contain significant data (see ACC, below). ACS can be derived from the hexadecimal representation of PSR as the remainder when the least significant digit is divided by 4.

Values are:

0 : invalid
1 : 32-bits
2 : 64-bits
3 : 128-bits

For values 1 and 2, it is the less significant end of the Accumulator which contains significant data.

PC - Program Counter - the address of the next instruction to be executed. By comparing this with the CONAD addresses revealed by #PVM, you can see which object file was being executed at the time of failure.

SSR - System Status Register - unlikely ever to be of interest.

SF - Stack Front - the limit address of valid data on the stack, and hence of the current stack frame (see LNB above). SF points just beyond the last item of data on the stack.

IT - Interval Timer - of no interest for diagnostic purposes.

IC - Instruction Counter - of no interest for diagnostic purposes.

CTB - Cross reference Table Base - so called, but usable for many other purposes. Should always contain an address, probably of some area of store that has recently been referenced. Very often points to a GLA or PLT (see LNB above), as it is convenient to use it in calling external routines or accessing static (%own) data.

XNB - Extra Name Base: very similar to CTB. Often points to the stack frame of another routine than the "current routine", having been loaded from the "display vector" - see LNB above.

B - May hold an address, but often used for simple arithmetic.

DR - Descriptor Register. This is used for addressing data in store. The left-hand byte gives the "type" of the descriptor, and typical values are

X'18' byte vector
X'28' word vector
X'30' or X'B0' double word vector
X'38' quad word vector
X'B1' used to point to GLA or PLT
X'58' string
X'E1' code - used for subroutine entry and exit
X'E3' system call descriptor (for protected subroutine calls)
X'E5' escape descriptor, mostly used for "dynamic routine calls".

The rest of the first word is a "bound" indicating the size of the area of store. It is not necessarily a count of bytes: for a word vector descriptor, for instance, it will be a count of words.

The second word is the address of the first byte of the area of store.

IMP strings are sometimes addressed by "string descriptors",

but more often by "byte vector descriptors".

ACC - ACCumulator, used for general arithmetic etc.

If ACS (see PSR, above) is 32 bits, then only the fourth (right-hand) word in ACC is significant. If ACS is 64, then the third and fourth words are significant. If ACS is 128, then all four words are significant.

FPC - "Failing PC". For some failures, will point to the instruction which caused the failure.

The layout of a stack frame is as follows:

First word: Address of the stack frame of the routine which called the current routine.

Second word: A copy of the PSR at the time of the call, which will normally be restored on exit from the current routine.

Third word: The address of the instruction in the calling routine to which control will return on exit from the current routine - i.e., the instruction just after the CALL which invoked the current routine.

Fourth word: For object code conforming to normal EMAS standards, the least significant 24 bits of this word are used to locate a "diagnostic record". Information about the diagnostic tables, and about the structure of object files, are beyond the scope of this Note.

Fifth word: The address of the GLA (see LNB above) of the current routine.

Sixth and subsequent words:

Parameters (if any) passed by the calling routine to the current routine.

Then come the display vector (see LNB above) plus any diagnostic information - the format varies from one compiler to another. For IMP, the first word after the parameters is often a "current line number". The display vector can be recognized as a sequence of words (or, for some languages, descriptors) all pointing into the stack.

Then come locally declared variables (but local arrays may be held off the stack).

and finally work space used in evaluating expressions.

The second and third words, and usually also the fourth and fifth words, will form descriptors.

6 Interpretation of #PVM output

The following notes apply to files as recognized by Director. They are not applicable to members of partitioned files.

Virtual store is divided into segments. The maximum size of a segment is 262144

bytes. Addresses are 32 bit quantities and the most significant 14 bits of an address are the segment number. The least significant 18 bits are the "address within segment". When a file is connected, it occupies one or more consecutive segments in virtual store. The first byte of the file is at address zero in its segment. All but the first (or only) segment will be full-sized, i.e. 262144 bytes long. No other file is connected in any of the segments.

The connect address CONAD of a file is the address of its first byte in virtual store. This must be the first byte of a segment, as described above, and SEG is the number of that segment. Thus CONAD is the same as SEG followed by 18 binary zeros. It is convenient to print CONAD as a hexadecimal number and SEG in decimal. Thus a file with a SEG of 51 will have CONAD 00CC0000.

When a file is connected, a number of consecutive segments are allocated for it in virtual store. There may be more segments than are needed for the size of the file; this allows the file to be expanded without being disconnected from virtual store. The number of segments is given as HOLE.

The current size of the file is given as K in units of 1024 bytes. This will always be a multiple of 4, since file space is allocated in units of 4096 bytes known as "E-pages".

FSYS is the number of the "file system" - i.e., disc - on which the files resides.

MODE indicates what actions are permitted on the file.

Mode bits are:

X'00000001'	read
X'00000002'	write
X'00000004'	execute
X'00000008'	accept shared write
X'00000010'	newcopy
X'00000020'	comms mode
X'00000040'	disc only
X'00000080'	new stack segment
X'00000100'	disallow DISCONNECT, CHANGE ACCESS, CHANGE SIZE
X'80000000'	non-slaved segment

Common valid combinations are (in decimal):

1	read	
2	write	{*}
3	read/write	
4	execute	{*}
5	read/execute	
9	read (accept shared write)	
10	write (accept shared write)	{*}
11	read/write (accept shared write)	
18	write newcopy	{*}
19	read/write newcopy	

where {*} means that Director grants read access in addition to the requested modes.

FILE gives the name of the file. Some files whose names you do not recognize are workspace of various kinds set up by the system to support your work.

USE gives the number of reasons why a file is connected. So long as it is non-zero, the file should not be disconnected from your virtual memory. Some files are never to be disconnected until the end of the session: these may be marked by an asterisk, or they may have a USE count of 8. USE is concerned only with usage within one process. If a file is being shared by two processes and is connected in both, that will not be shown by USE. USE is commonly 0 or 1. 0 means that the file is not being used by the process and may be disconnected at any time. A USE value greater than one may arise when there is more than one reason why the file should be connected.