EDINBURGH REGIONAL COMPUTING CENTRE

EMAS 2900 SERVICE NOTE

No: 1 (version 5)
Date: April 1980

Conversion to EMAS 2900 for 4-75 Users

N.B. THIS NOTE REPLACES SERVICE NOTE 1 DATED AUGUST 1979.

Introduction

This note is intended for existing users of EMAS on the 4-75 who are planning to move to EMAS 2900. It primarily concerns differences in the appearance of the Subsystem, but other areas are mentioned where relevant. It reflects the state of the System as running on the ERCC 2970 at the time of writing. More up-to-date information can be found in the ALERT and HELP information.

Essential background information

It is assumed that the reader has access to

* EMAS 2900 User's Guide, Second Edition, January 1980

* EMAS 2900 Information Card, Third Edition, February 1980

* For IMP users, the IMP Language Manual, and particularly its Update 1 (February 1978)

Design intention

The user interface has been based on that used for EMAS on the 4-75. The re-implementation for ICL 2900 Series computers has been used as an opportunity to tidy up some of the inconsistencies in the command language, in the light of experience of the problems which users have encountered on the 4-75. The intention is that it should be possible to carry out all the same operations - often using identical commands.

The rest of this note should be read in conjuction with the EMAS 2900 User's Guide.

Chapter 1

File names are constructed as on the 4-75 except that both file names and partitioned file member names can be up to 11 characters.

Chapter 2

The device code .TT is no longer valid. In its place the more general forms .IN and .OUT must be used. They refer respectively to the current primary input and output. For foreground sessions these are normally both the interactive terminal. Hence:

```
          IMP(A,AY,.OUT)
```

Use HELP(DEVICES) to obtain an up-to-date list of output devices.

Chapter 3

When logging on, respond to the "HOST" prompt with "2970" or "2980", not with "EMAS".

Chapter 4

Note the distinction between SS# and T# files. Note also that the name of the default compiler listing file is T#LIST.

Chapter 6

The command FILEANAL is replaced by ANALYSE, which takes three parameters:

```
          ANALYSE (file, option, out)
```

See page 6-1 for details.

COPYFILE becomes COPY.

Chapter 7

The distinction between Character files and Data files is more significant on the 2900. In particular the text editors only operate on Character files. The provision of a format "C" in DEFINE makes it possible to generate Character files as output from a FORTRAN program.

The first parameter for DEFINE is now an integer; hence:

```
          DEFINE(3,FILEAB)   not   DEFINE(FT3,FILEAB)
```

Note that DDLIST is replaced by DEFINE(?).

Chapter 11

The program loading mechanism is significantly different. However, the majority of users will not be greatly affected.

INSERT replaces INSERTFILE. It must be called explicitly for object files, other than main programs, before they are first used.

The function of APPENDLIB is now replaced by a facility provided by the OPTION command. Briefly:

```
APPENDLIB (CONLIB.GRAPHICS)   becomes

OPTION (SEARCHDIR=CONLIB.GRAPHICS)
```

There is a new command ALIAS which makes it easy to modify the names of commands.

Chapter 12

The first 31 characters of entry names are significant.

TESTINT is replaced by INTERRUPT.

SSINFO is replaced by two functions UINFI and UINFS.

Chapter 16

The implementation of BACKSPACE is incomplete. In the case of Character files there is no restriction. In the case of Data files consecutive calls of BACKSPACE are faulted.

Chapter 17

The OPTION command provides more facilities than its counterpart on the 4-75. In particular it replaces USERLIB, APPENDLIB and REMOVELIB. See page 17-5.

Command changes

The list on the next page gives the names of 4-75 EMAS commands, their EMAS 2900 equivalents (where different) and an indication of which commands are significantly different on EMAS 2900.

EMAS 4-75 command	EMAS 2900 equivalent (if different)	Significant differences	Notes
ACCEPT			
ALERT			
ALGOL			
APPENDLIB	OPTION	*	See Chapters 11 and 17
ARCHIVE			
CHERISH			
CLEAR			
CONCAT			
COPYFILE	COPY		
CPULIMIT			
DDLIST	DEFINE(?)		
DEFINE		*	
DEFINEMT			
DELETEJOB	DELETEDOC		
DELIVER			
DESTROY			
DETACH		*	See Chapter 16
DISCARD			
DISCONNECT			
EDIT			
FILEANAL	ANALYSE	*	
FILES			
FINDJOB	DOCUMENTS		
FORTE			
HAZARD			
HELP			
IMP			Changes in language
INSERTFILE	INSERT		
LIBANAL	ANALYSE	*	
LINK			
LIST			
LOOK			
METER			
NEWPDFILE			
NEWSMFILE			
OBEYFILE	OBEY		
OFFER			
OPTION		*	See Chapter 17
PARM			
PASSWORD			
PERMITFILE	PERMIT		
PERMITLIB	PERMIT	*	No direct equivalent
PROJECT			Not available
QUEUES	DOCUMENTS		
RECALL			
REMOVEFILE	REMOVE		
REMOVELIB	OPTION		See Chapters 11 and 17
RENAME			
RESTORE			
RUN			
SEND			
STOP			
SUGGESTION			
TIDYLIB	TIDYDIR		
USERLIB	OPTION	*	See Chapters 11 and 17, and command NEWDIRECTORY
USERS			

R.R. McLeod

EMAS 2900 SERVICE NOTE

No: 2 (revised)
Date: 18/6/80

Transferring Files between the 2970 and 2980: T080 and T070

The commands T080 and T070 are available for transferring a user file from one mainframe to another. If it is a character file it is dispatched immediately along communications lines to the target machine. Otherwise it is sent along communications lines as a background job, so that in due course a line printer report of the execution of the job (on the target machine) will be received. There is no restriction on the type of file which may be transferred, but the size of the file cannot be greater than 1023 Kbytes.

Establish access to the commands as follows:

```
Command:OPTION(SEARCHDIR=CONLIB.GENERAL)      (2970)

Command:OPTION(SEARCHDIR=ERCLIB.GENERAL)      (2980)
```

There are two forms of the command: T070 and T080, where the suffix indicates the target mainframe. The parameters for both commands are as follows:

```
Command:TOxx(file, newowner.newfile, back, overwrite)
```

file	-	the file or partitioned file member to be transferred (may belong to another user so long as read access is permitted).
newowner.newfile	-	if newowner is omitted the owner of the process issuing the command is used. If newfile is also omitted the file is given the same name on the target machine.
back	-	the background password of the receiving owner on the target machine.
overwrite	-	OVERWRITE may optionally be specified: if file "newfile" already exists in the target machine process, it will then be overwritten.

Please contact your local Advisory Service if any difficulties are encountered in using the above facilities.

A. Shaw

EMAS 2900 SERVICE NOTE

No: 3 (revised)
Date: 10/6/80

Accounting and Charging Routines

These routines are held in object file ACCNTS.PDCHARGE_CHARGEY on the 2970 and 2980; the corresponding source file is ACCNTS.PDCHARGE_CHARGEnS (2970 only).

The routines are described below.

externalroutine CHARGES (test)

This routine takes an optional parameter, either a specific username or the word TEST; in either case a test run is implied. The effects are described below.

If no parameter is given, CHARGES reads and clears the file index accounting information for every accredited user in the System. The information read is put into a store map file called JUSE which is then sent to the JOURNL process via the SPOOLR JOURNL queue.

Before CHARGES does this, however, it makes use of the file created for JOURNL to update a weekly accounting file and a "cumulative use" file.

The current accounting week is established and a file for the week's accounting is created unless it already exists. The form of the file name is A29nnmmmn, e.g., A2980JUN2.

The format of the file is as follows:

(1) "ddmmyyddmmyy"

the dates (inclusive) of the accounting week

(2) #n dd/mm/yy

n	specifies the charge code 1, 2, 3, 4 ......
dd/mm/yy	gives the date on which the charges following were incurred

(3) hh.mm.ss AAAAnn ppp

hh.mm.ss	is the time of day at which the charge was calculated
AAAAnn	is the relevant user number
ppp	is the charge (in pence) incurred

Line (1) appears once only, at the start of the file. Following each occurrence of a line with format (2) there are a number of lines with format (3). The charge code determines what sort of charge is being made in the lines following:

1	use of 2980 or 2972
2	file space on 2980 or 2972
3	use of 2970
4	file space on 2970

On each call of the CHARGES routine (with the exception of test runs), two sets of charges, use and file space, are calculated for the relevant machine and appended to the current accounting week file. This file is then normally accessed during the following accounting week (i.e. when it is complete).

The "cumulative use" file is called CUSE. It is created by CHARGES if necessary. It can be listed, in whole or in part, by use of the command LISTCUSE, described below.

When CHARGES is called with a parameter, a test run is carried out. In this case the file indexes are examined but not cleared, the weekly charge filename is not prefixed by A29nn, and the names of the files generated (JUSE, CUSE, MMMn) all have "TEST" appended. JUSETEST is not transferred to the JOURNL process; it will be overwritten in any subsequent test run.

If the parameter is a user name then only the charges for that user are calculated. If the parameter is TEST then the charges for all accredited users are calculated.

Notes

* If file JUSE is found by CHARGES to exist, CHARGES starts by sending it to the JOURNL queue. This should not happen, since CHARGES should have done this the previous time it was called.

* The structure of the JUSE file is described below.

* The structure of the "cumulative use" file is described below.

* At the time of writing, CHARGES is invoked nightly by a batch job which causes itself to be run the next day also. Thus the charging scheme is automatic and no operator intervention is required.

* The processing of the completed weekly accounting files is carried out as a separate stage.

---

externalroutine LISTCUSE (string(255) S)

This routine either creates and initialises the "cumulative use" file added to by CHARGES (see above), or, if the file exists, lists the contents.

The form of a call of the routine is as follows:

```
Command:LISTCUSE(usermask,out,file)
```

All three parameters are optional.

usermask	can be null (the default), meaning that information on all users is to be output, or a six character username, meaning that information on that user only is to be output, or a name including one or more '?' signs, specifying a group of users about whom information is to be output; e.g. ERCCI?? would give information on all users whose names started ERCI.

out	can be null (the default), meaning that output is to be sent to the primary output device (normally the user's terminal), <u>or</u> an output device (e.g. .LP), <u>or</u> a filename.
file	by default this is CUSE. If a test CUSE file is to be examined (see above), this parameter must be specified as CUSETEST.

If usermask is non-null and does not contain six characters, an error is flagged.

<u>externalroutine</u> LISTJUSE (<u>string</u>(255) S)

This routine lists the complete contents of the store map file JUSE (or JUSETEST). The form of a call of the routine is as follows:

```
Command:LISTJUSE(file,out)

file  is JUSE by default

out   is .OUT by default
```

Refer to the description of DSFI in the EMAS 2900 Subsystem Writer's Manual for explanations of the various items listed.

<u>externalroutine</u> GETUSE(<u>string</u> (6) USER, <u>integer</u> FSYS, RESET, ADR)

This routine returns accounting information in a record array starting at ADR, for all accredited EMAS users (if FSYS is -1), or for all users on the disc pack specified by FSYS, or for a single user (if USER is non-null). In the latter case FSYS can be used to specify where the user's file index resides, if known.

The record array starting at ADR is assumed to be (0:n) (STATSF):

```
recordformat STATSF(string(6) USER, c
   integer NEXT, FSYS, KINST, PTURN, KBTSLDEV, KBFSLDEV, c
   MSOCP, CONNECT, AFILE, AKB, DFILE, DKB, CFILE, CKB)
```

The records are held in a list structure to give alphabetic order (_NEXT giving the next item in the list). This list is terminated by a dummy record held in array element 0, with (0)_USER="ZZZZZZ". The genuine last user record thus has _NEXT=0.

The 0th record, on return from GETUSE, holds two items of information:

(0)_NEXT gives the FIRST record in the list structre
(0)_FSYS gives the TOTAL number of (genuine) user records in the structure.

If on entry (0)_NEXT is non-zero, this indicates that a record array structure already exists. In this case each user's accounting information is added in to the corresponding record (unless no such record currently exists, when on is created). When (0)_NEXT is non-zero on entry, it is assumed to point to th start of the structure, and (0)_FSYS is assumed to give the total number of user records in the structure.

If all the file systems are searched and more thasn one file index is found for a particular user, then the accounting information for each is stored in a separate record of the record array.

If RESET is 0 on entry, the accounting information is merely read from the user's file indexes. If it is 1, the file indexes are also reset.

It is expected thet GETUSE will be called on a regular basis with RESET=1, and that either all the discpacks will be searched in a single call, or that GETUSE will be called for each discpack in turn, the calling program having used GET AV FSYS beforehand.

The user of GETUSE must be running at an ACR level of 5 or less.

Structure of JOURNL File JUSE[TEST]

This file has a standard Edinburgh Subsystem file header. It is a store map file.

Following the file header, the first two words contain the packed date-and-time and the pattern X'FFFFFF03' respectively.

The remainder of the file is used to hold a record array of the structure described in GETUSE (above).

Structure of Summary File CUSE[TEST]

This file has a standard Edinburgh Subsystem header. It is a store map file.

Following the header, there are two 8-byte strings (each allocated ten bytes) which give the start and finish dates corresponding to the information held.

Thereafter a record array is mapped onto the file, from byte 20 after the end of the Subsystem header. The record array is defined as folows:

```
recordformat CRECF(string(6) USER, integer NEXT, USEP, FILEP)

recordarray CREC(0:1024) (CRECF)
```

where USER is a user name NEXT points to the next user (in alphabetical order) USER is the accumulated use charge in pence for that user FILEP is the accumulated file charge in pence for that user.

Note that:

```
     CREC(0)_USER = "ZZZZZZ"
     CREC(0)_NEXT = array element containing first genuine user
                    alphabetical order)
     CREC(0)_USEP = total number of genuine users in array.
```

The 0th element of the array is used to terminate the list. Thus CREC(i)_NEXT=0 if the user in element 1 is the last one in the alphabetical list.

J.M. Murison

No: 3
Date: 23/6/78

EMAS 2900: Accounting and Charging Routines

These routines are held in the object file ERCI05.CHARGE (the corresponding source file is ERCI05.CHARGES), and are as follows:

externalroutine CHARGES

This routine takes no parameters. It reads and clears the file index accounting information for every accredited user in the System. The information read is put into a store map file (currently called JUSE) which is dispatched to the JOURNL process via the SPOOLR JOURNL queue.

Before CHARGES does this, however, it makes use of the file created for JOURNL in 3 ways:

1) A character file (currently called AddmmU, where ddmm is part of the date of the run of the program) is created and dispatched to System 4 EMAS process ACCNT4. The format of each line of this file is:

```
        hh.mm.ss AAAAnn    ppp
```

where	hh.mm.ss	is the time of day
	AAAAnn	is a user number
	ppp	is the charge in pence attributed to the user because of his use of the machine (hence the U at the end of the filename).

The contents of the JOURNL file are used to calculate the charges, the current charging formula being used. This charging formula is implemented in integerfn USECH, which is local to CHARGES, and an explanation of the constants involved is given there in comment statements.

2) A character file (currently called AddmmF) is created and dispatched to System 4 EMAS process ACCNT4. The format of each line of the file is identical to that for the previous file. The charge (ppp) is that attributed to the user because of the file space currently belonging to him (hence the F at the end of the filename).

The contents of the JOURNL file are used to calculate the charges, the current file space rates being used. These rates are implemented in integerfn FILECH, which is local to CHARGES.

3) A store map file (currently called CUSE) is updated if it exists at the time CHARGES is invoked. This is a "cumulative use" file whose contents can be output, in whole or in part, by use of the command LISTCUSE, described below.

Notes

* If a JOURNL file is found by CHARGES to exist, CHARGES starts by sending it to the JOURNL queue. This should not happen, since CHARGES should have done this the previous time it was called.

* If either of the character files of charging information is found by CHARGES to exist, then CHARGES appends the charges calculated in its current call to these files. Again, this should not happen.

* The structure of the JOURNL file is described below.

* The structure of the "cumulative use" file is described below.

----

externalroutine LISTCUSE (string(255) S)

This routine either creates and initialises the "cumulative use" file added to by CHARGES (see above), or, if the file exists, lists the contents. In the latter case it also gives the user the option to re-initialise the file.

The form of a call of the routine is as follows:

```
       COMMAND:LISTCUSE(usermask,out)
```

Both parameters are optional.

```
usermask               can be null (the default), meaning that information
                          on all users is to be output
                       or a six character username, meaning that information on
                          that user only is to be output
                       or a name including one or more '?' signs, specifying a
                          group of users about whom information is to be
                          output; e.g. ERCCI?? would give information on all
                          users whose names started ERCI.

out                    can be null (the default), meaning that output is to
                          be sent to the primary output device (normally the
                          user's terminal)
                       or an output device (e.g. .LP)
                       or a filename.
```

If usermask is non-null and does not contain six characters, an error is flagged.

In all cases however the user is prompted:

```
           Reset CUSE?
```

to which he should reply

```
           Y <CR>               if the file is to be cleared

        or N <CR> or just <CR> if it is not to be cleared.
```

```
externalroutine GETUSE(string (6) USER, integer FSYS, RESET, ADR)
```

This routine returns accounting information in a record array starting at ADR, for all accredited EMAS users (if FSYS is -1), or for all users on the disc pack specified by FSYS, or for a single user (if USER is non-null). In the latter case FSYS can be used to specify where the user's file index resides, if known.

The record array starting at ADR is assumed to be (0:n) (STATSF):

```
       recordformat STATSF(string(6) USER, c
          integer NEXT, FSYS, KINST, PTURN, KBTSLDEV, KBFSLDEV, c
          MSOCP, CONNECT, AFILE, AKB, DFILE, DKB, CFILE, CKB)
```

The records are held in a list structure to give alphabetic order (_NEXT giving the next item in the list). This list is terminated by a dummy record held in array element 0, with (0)_USER="ZZZZZZ". The genuine last user record thus has _NEXT=0.

The 0th record, on return from GETUSE, holds two items of information:

```
   (0)_NEXT gives the FIRST record in the list structure
   (0)_FSYS gives the TOTAL number of (genuine) user records in the
            structure
```

If on entry (0)_NEXT is non-zero, this indicates that a record array structure already exists. In this case each user's accounting information is added in to the corresponding record (unless no such record currently exists, when one is created). When (0)_NEXT is non-zero on entry, it is assumed to point to the start of the structure, and (0)_FSYS is assumed to give the total number of user records in the structure.

If all the file systems are searched and more than one file index is found for a particular user, then the accounting information for each is stored in a separate record of the record array.

If RESET is 0 on entry, the accounting information is merely read from the users' file indexes. If it is 1, the file indexes are also reset.

It is expected that GETUSE will be called on a regular basis with RESET=1, and that either all the discpacks will be searched in a single call, or that GETUSE will be called for each discpack in turn, the calling program having used GET AV FSYS beforehand.

The user of GETUSE must be running at an ACR level of 5 or less.

Structure_of_JOURNL_File_JUSE

This file has a standard Edinburgh Subsystem file header. It is a store map file.

Following the file header, there is a standard identification string. This is inserted by use of externalroutine JHEAD, which is passed the address of the start of the user data and a code string indicating the use of the file. JHEAD inserts the identification string in the file, and updates the address to point to the next free space in the file.

The remainder of the file is used to hold a record array of the structure described in GETUSE (above).

Structure_of_Summary_File_CUSE

This file has a standard Edinburgh Subsystem header. It is a store map file.

Following the header, there are two 8-byte strings (each allocated ten bytes) which give the start and finish dates corresponding to the information held.

Thereafter a record array is mapped onto the file, from byte 20 after the end of the Subsystem header. The record array-is defined as follows:

```
        recordformat CRECF(string(6) USER, integer NEXT, USEP, FILEP)

        recordarray CREC(0:1024) (CRECF)
```

```
where USER  is a user name
      NEXT  points to the next user (in alphabetical order)
      USER  is the accumulated use charge in pence for that user
      FILEP is the accumulated file charge in pence for that user.
```

Note that:

```
   CREC(0)_USER = "ZZZZZZ"
   CREC(0)_NEXT = array element containing first genuine user (alphabetical
                  order)
   CREC(0)_USEP = total number of genuine users in array.
```

The 0th element of the array is used to terminate the list. Thus CREC(i)_NEXT=0 if the user in element i is the last one in the alphabetical list.

J.M. Murison