UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 58

(June 1986)

Title:

Moving IMP Programs to EMAS-3

Author: J.M. Murison

Contact: Advisory service

Software Support Category: See Note 15

Synopsis

This Note suggests a series of steps to follow in order to convert an IMP80 program on EMAS 2900 to an IMP program on EMAS-3.

Keywords

conditional compilation, EMAS 2900, EMAS-3, IMP, IMP80, SOAP

Edinburgh Regional Computing Centre

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

Introduction

When IMP programs are moved from EMAS 2900 to EMAS-3, a number of changes may have to be made. These are necessitated by changes in the implementation of the IMP language, in the operating system and in the machine hardware.

The first section of this Note suggests a series of steps to follow in order to convert an IMP80 program on EMAS 2900 to an IMP program on EMAS-3. In doing so it refers to later sections and to other documents. Although there are, in total, a large number of *potential* changes required to an IMP program, in practice it has been found that conversion for EMAS-3 is straightforward in most cases.

The last section of this Note describes the technique of *conditional compilation*, by use of which it is possible to produce a single 'master' source program which will compile on either EMAS 2900 or EMAS-3.

1. Making the move: steps to follow

This section describes the moving of an IMP program from the EMAS or BUSH services to the EMAS-A service at Edinburgh. In principle, it applies to any service using the EMAS 2900 system and any service using the EMAS-3 system.

The steps are as follows:

a) Convert to IMP80
b) Transfer source to EMAS-3
c) Get familiar with EMAS-3
d) Use SOAP
e) Compile program
f) Check changes in IMP & EMAS
g) Command writers: use PAM
h) Test program

a) *Convert to IMP80.* If you use the EMAS 2900 command IMP80 to compile your IMP program, go to step (b). Otherwise consult User Note 2 to convert to IMP80 on EMAS 2900.

b) *Transfer source to EMAS-3.* Get yourself a number on EMAS-A (User Note 29), and transfer your source IMP file(s) to it (User Note 60, Section 6). You can use the command TRANSFER, or TOEMASA. For example:

```
Command: OPTION SEARCHDIR=ERCLIB.GENERAL                    (Once only)
Command: TOEMASA PROG2900S, USERNO:PROG2900S, BACKPASS
```

This command would transfer file PROG2900S to EMAS-A. USERNO is your user number on EMAS-A; BACKPASS is your background password on EMAS-A. If your user number on the service from which the transfer is being made is the same as on EMAS-A then the USERNO: part can be omitted; and if in addition the file is to have the same name on EMAS-A (as shown here) then the second parameter can be omitted entirely, but not the comma following.

You cannot transfer object files, but it would be pointless to try as they would not run on EMAS-A anyway. You can, however, transfer partitioned files.

c) *Get familiar with EMAS-3.* If you have not already done so, get familiar with the main differences between the two systems (User Notes 60 and 61). All the conversion work, as recommended here, is to be done on EMAS-A.

d) *Use SOAP.* If your program does not contain any %EXTERNAL or %SYSTEM %ROUTINE %SPECs, go to step (e). Otherwise, assuming that your file is called PROG2900S, give the command:

```
Command: SOAP PROG2900S, PROG370S, TOEMAS3=Y
```

(SOAP is an IMP program formatter, described in User Note 14.) This produces, in file PROG370S, an equivalent source program in which those %EXTERNAL and %SYSTEM %SPECs which the SOAP utility knows about are converted to the correct EMAS-3 form. Warning messages are generated for %SPECs which it does not know about.

One important set of external routines are the standard EMAS commands (FILES, MAIL, HELP, etc.). In EMAS 2900 IMP80 these can be called either by providing a %SPEC of the form:

```
%EXTERNAL %ROUTINE %SPEC command(%STRING (255) PARAMS)
```

and then calling routine command directly, or by making use of CALL:

```
%EXTERNAL %ROUTINE %SPEC CALL(%STRING (255) PARAMS)
```

and then invoking the command by use of CALL(command, params). In EMAS-3 IMP the first, direct, method is not available; and in the second method routine CALL is replaced by routine EMAS3:

```
%EXTERNAL %ROUTINE %SPEC EMAS3(%STRING %NAME COMMAND, PARAMS,
                                %INTEGER %NAME FLAG)
```

This conversion is handled automatically by SOAP.

Have a look at PROG370S with your favourite EMAS editor. If you do not like what SOAP has done to your program, read User Note 14 to find out how to control SOAP; you should be able to make it format your program in the style you prefer.

e) *Compile program.* Give the command:

```
Command: IMP PROG370S, PROG370Y, PROG370LST
```

[Note the change of command name from IMP to IMP80.] If the compilation fails, read Section 2 of this Note to find out which changes are relevant to the failure. If none seems to be relevant, contact the ERCC Advisory service.

Modify PROG370S with an editor until it compiles successfully. In this context, note that the editor ECCE has an IMP syntax checker built in; this can be invoked within ECCE by use of the command %V. It will flag any syntactically faulty IMP statements, but will not detect that variables have not been declared, or have been declared twice, or that parameters to a procedure do not match the %SPEC, etc. None the less, it does speed up the edit-compile-edit-... iteration.

If the IMP compiler produces warning messages, do *not* disregard them: they can indicate that some features of IMP used in the program do not have quite the same effect on EMAS-3 as on EMAS 2900.

Do not proceed to the next step until your program compiles successfully.

f) *Check changes in IMP & EMAS.* If you have not already done so, you should *now* read Section 2 of this Note, because some of the language and system changes do not provoke compilation errors in unmodified IMP programs. Modify PROG370S accordingly.

g) *Command writers: use PAM.* If your program consists of one or several commands (in the EMAS sense) then you should now read User Note 62, and apply the necessary changes to PROG370S. Note: although the *Parameter Acquisition Mechanism* described in User Note 62 may take some getting used to, it is strongly recommended as it provides a uniform parameter-checking facility which can reduce the amount of code in IMP programs quite dramatically. It also removes the need to add code to handle group names, files within sub-groups, etc.

h) *Test program.* Try running PROG370Y. If it fails, read Section 2 again to see whether the failure is related to any item mentioned there. If it does not appear to be, contact the ERCC Advisory service. If it does (apparently) work, but contains constructions which you have had to change, then examine the logic of your new code very carefully, especially if the program is a large one or if you wrote it some time ago. It is surprising how a small change can have a large impact, especially in the area of Input/Output. In this context, it is well to bear in mind Dijkstra's dictum: *"Testing can show the presence of bugs, but can never show their absence"*.

2. Relevant differences in IMP, EMAS and the hardware

a) In EMAS-3 IMP, %ROUTINE READSYMBOL returns ISO codes 10 (NL) and 32-126 only, and ignores all other values; in particular it does not return ISO code 26 (SUB). For values greater than 127, it ignores the top bit (i.e. subtracts 128), and then treats the result as for values 0-127. READSYMBOL on EMAS 2900 ignores all values greater than 127.

b) In EMAS 2900 IMP80, %ROUTINE PRINTSYMBOL ignores the top bit on output. This does *not* happen on EMAS-3: PRINTSYMBOL can handle any value in the range 0-255 and is thus identical to PRINTCH.

c) In EMAS 2900 IMP80, access by use of CHARNO to a byte beyond the currently defined length of a string is permitted. This is not true in EMAS-3 IMP. However, in EMAS-3 IMP the length of a string constant or %CONSTANT %STRING can be found (but not changed!) by use of LENGTH(STR), and the form CHARNO(STR,N), where STR is a string constant or %CONSTANT %STRING and N is an integer constant or %CONSTANT %INTEGER, is also permitted on the right hand side of an expression. This is not true of EMAS 2900 IMP80.

d) In EMAS-3 IMP, jumping into %CYCLE/%REPEAT loops is faulted.

e) In EMAS-3 IMP, within a block no declarations of variables are allowed after *any* executable code.

f) In EMAS 2900 IMP80, the %ARRAY %MAP ARRAY can take the name of an existing array as its second parameter, the format of that array then being used by ARRAY. This is not permitted in EMAS-3 IMP: the second parameter of ARRAY must be an explicitly declared %ARRAY %FORMAT.

g) 2900 assembler code will not work on EMAS-3. The best advice is to translate the assembler into IMP. If this proves impossible, contact the ERCC Advisory service.

h) %LONG %INTEGER operations are provided by software emulation. Those involving multiplication or division are very slow. Exponentiation of a %LONG %INTEGER is not allowed in EMAS-3 IMP.

i) The keyword %SYSTEM is permitted in EMAS-3 IMP, but does not cause the prefix 'S#' to be added to the procedure entry point (see below). It is therefore advisable to remove *all* occurrences of this keyword from IMP programs, by use of %ALIAS; for example:

```
%SYSTEM %STRING %FUNCTION %SPEC ITOS(%INTEGER N)
```

should become

```
%EXTERNAL %STRING %FUNCTION %SPEC ITOS %ALIAS "S#ITOS"(%INTEGER N)
```

See also the next item.

j) On EMAS 2900 some Subsystem procedures require the keyword %SYSTEM in their specification, some require %EXTERNAL. On EMAS-3 this situation has been tidied up, as described in User Note 80. There are two sets of procedures: the *language-independent programming interface* (the procedure names start EMAS3...); and the *IMP-only interface* (the procedure entry points start S#...). The two sets of procedures are equivalent, although the parameter types are usually different, the former set using %NAME-type parameters exclusively.

When converting an EMAS 2900 IMP80 program, you thus have a choice as to which set of procedures you use (perhaps a mixture of the two). Generally it is simpler when converting an existing program to use the IMP-only interface, and this is what SOAP will do. But if you decide to use the language-independent interface, note that in EMAS-3 IMP a *value or expression* can be given as an actual parameter to an *external* procedure where the formal parameter is %NAME-type. For example:

```
%EXTERNAL %ROUTINE %SPEC EMAS3 PROMPT(%STRING %NAME PSTRING)
  :
EMAS3 PROMPT("Input:")
  :
```

This feature is *not* available in EMAS 2900 IMP80.

As indicated earlier, the text formatting utility SOAP (User Note 14) can make the necessary changes automatically. If you prefer to make these changes 'by hand', then file ERCLIB:SOAPSPECS, which contains the specs used by SOAP, may be of use to you.

k) In EMAS-3 full filenames can be up to 255 characters in length, and so all string variables intended to hold filenames should be declared with maximum length 255.

l) %HALF %INTEGERs are not implemented in EMAS-3 IMP. They are treated as %INTEGERs by the compiler and a warning message is printed in the listing file. For some programs this may be an adequate compromise, but it will not work for any programs that access %HALF %INTEGERs by mapping, e.g. those which access %HALF %INTEGERs in store mapped files. There is no mapping function HALF INTEGER.

In EMAS-3 IMP %SHORT %INTEGERs are implemented as described in the ERCC IMP80 Language manual, as they were on the System 4. Although both %HALF %INTEGERs and %SHORT %INTEGERs take two bytes to hold an integer value, the distinction between them is that %HALF %INTEGERs are treated as unsigned and %SHORT %INTEGERs are signed. Hence:

%HALF %INTEGERs *hold positive integer values in the range 0 to 65,535*

%SHORT %INTEGERs *hold integer values in the range -32,768 to 32,767*

Programs which use %HALF %INTEGERs for holding values within the range 0 to 32,767 can be converted by substituting %SHORT for %HALF throughout. In other cases it may be possible to use a jam transfer assignment (i.e. using the operator <-). However, if you do this you should be careful when making arithmetic comparisons between the resultant values. For example:

```
%SHORT %INTEGER VAL
  :
VAL <- X'FFFF';     ! VAL = X'FFFF' would fail (integer overflow).
%IF VAL=X'FFFF' %THEN  ....
  :
```

The test for equality will fail. This is because X'FFFF' is held as a 32-bit integer constant (X'0000FFFF') and to make the test the value in VAL has to be expanded up to a 32-bit quantity also. This is done by propagating the most significant bit (the sign bit) leftwards so that the numerical value is preserved. In this case the result is X'FFFFFFFF', which is not equal to X'0000FFFF'.

m) On EMAS-3, the %EXTERNAL %ROUTINE SET RETURN CODE prints a message appropriate to the integer value passed to it (see Section 5 of User Note 80), this message *including* the name of the routine from which SET RETURN CODE was called. On EMAS 2900 SET RETURN CODE does not print any message.

On EMAS-3, if SET RETURN CODE is called at the outer level of a command (%EXTERNAL %ROUTINE), then the change is normally of benefit. For example, if %EXTERNAL %ROUTINE WIDGET calls SET RETURN CODE(271) then the user receives the message

WIDGET fails - Attempt to write to PD member

However, if the programmer has called SET RETURN CODE in an inner routine, then the name of *that* routine will be included in the message. To get round this, the following approach can be used:

```
    :
%EXTERNAL %ROUTINE %SPEC PSYSMESS %ALIAS "S#PSYSMESS"(%STRING (255) CALLER,
                                                %INTEGER FLAG)
    :
PSYSMESS("WIDGET", 271)
%STOP
    :
```

PSYSMESS sets the return code to the value of its second parameter. If you do not want any message to be printed then call SET RETURN CODE(FLAG) with FLAG negative. This has the effect of inhibiting its message; the return code set is IMOD(FLAG).

If you want to print a system message without setting the return code, use routine FAILURE MESSAGE:

```
%EXTERNAL %STRING %FUNCTION %SPEC FAILURE MESSAGE %C
                                %ALIAS "S#FAILUREMESSAGE"(%INTEGER N)
```

n) A new routine, EMAS3 CLAIM CHANNEL, is provided on EMAS-3. While not strictly relevant to the conversion of an EMAS 2900 IMP80 program to EMAS-3 IMP, it is none the less a useful facility if you are revising your program while moving it.

Many programs have been written with channel numbers 'built in', and this often leads to clashes and inconvenience. The new procedure has been provided to eliminate this problem. Its specification is:

```
%EXTERNAL %ROUTINE %SPEC EMAS3 CLAIM CHANNEL(%INTEGER %NAME CHAN)
```

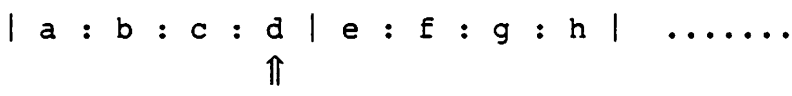
When you want to get a new I/O channel within an IMP program, call this procedure. If a free channel is available (it nearly always will be), its number will be set into the %INTEGER parameter passed; otherwise the value returned will be 0. The channel number supplied will normally be negative, though you need not be concerned with this fact. Negative channel numbers may only be used when supplied by EMAS3 CLAIM CHANNEL.

The channel number obtained can be used as a parameter to DEFINE. Within an IMP program the string function ITOS can be used to do this:

```
%EXTERNAL %STRING %FUNCTION %SPEC I TO S %ALIAS "S#ITOS"(%INTEGER I)
%EXTERNAL %ROUTINE %SPEC EMAS3(%STRING %NAME COMMAND, PARAMS,
                        %INTEGER %NAME FLAG)
%EXTERNAL %ROUTINE %SPEC EMAS3 CLAIM CHANNEL(%INTEGER %NAME CHAN)
    :
EMAS3 CLAIM CHANNEL(CHAN NO)
EMAS3("DEFINE", ITOS(CHAN NO).",T#TEMP1", FLAG)
    :
SELECT OUTPUT(CHAN NO)
    :
```

o) On EMAS-3 a 'T#..' filename is allowed a maximum of only 9 characters in its name. This is because the EMAS-3 system appends two 'hidden' digits to the name, as compared with only one digit on EMAS 2900.

p) Mapping functions: consider the situation where an %INTEGER %MAP returns a reference to an address which is not word-aligned,



such as the address of the byte labelled d in the above diagram. In EMAS 2900 IMP80, the hardware causes this to be modified to a reference to the integer made up of bytes a-b-c-d. In EMAS-3 IMP the reference returned is to the (unaligned) integer made up of bytes d-e-f-g; this does *not* cause an address error. A similar effect can occur with halfword-aligned objects.

q) The IBM-type machines on which EMAS-3 runs do not use *descriptors* to access items. This reduces the amount of checking that is done by the hardware, for example the types of parameters passed to %EXTERNAL procedures. This may give rise to problems, which can be avoided by checking that the specifications of %EXTERNAL items are correct. In particular, where a string is passed by value to an external procedure, the string maximum size given in the specification *must* match the procedure's description exactly. This is because the size given determines how many bytes are passed to the procedure. The problem does not arise on 2900 series machines.

Of course, the EMAS-3 IMP compiler checks within your IMP program that the actual parameters you supply are consistent with the specifications you provide. In addition, the program loader (User Note 85) checks that the number and *total* size of the parameters passed to an external procedure are consistent with the procedure's expectations.

r) The IBM-type hardware used by EMAS-3 does not provide such a sophisticated mechanism for store access control as the 2900 series. Although there is no greater risk of your information being read or altered illegally by other users, there is a greater chance of your program being able to read from incorrect addresses. For example, if your program computes an incorrect address and attempts to read from the data at that address then in the case of the 2900 it is likely that it would fail with an address error. On EMAS-3 there is a greater chance that it will read something at the incorrect address. This is obviously only a problem with faulty programs, but is something to bear in mind when investigating apparently intermittent behaviour of your program.

s) Programs should manipulate addresses with care. On EMAS-3 you should not assume that addresses will be positive values. Before comparing two addresses you should AND off the most significant bit. This will normally arise only where an IMP program calls (or is called by) a non-IMP program.

t) The serious limitation imposed on the size of the stack on the 2900 series machines does not apply, so there is no longer the complication of having a User stack and an Auxiliary stack, and of having to partition the available space between them.

3. Conditional compilation

When you have successfully completed the conversion of your IMP program to run on EMAS-3 you will probably notice that the differences between the two versions lie in three areas:

changes in %EXTERNAL ... %SPECs
changes as a result of other language and system differences
(for commands) changes in the acquisition and checking of parameters

These differences are often quite localized within a program, and the rest of the two versions may well be identical. There can be advantages in working with a single source file for both versions, particularly if the program is to remain in use on EMAS 2900.

A facility is provided with the EMAS 2900 IMP80 compiler and the EMAS-3 IMP compiler which is useful in this context, namely *conditional compilation.* Consider the following example:

```
  :
%CONSTANT %INTEGER TARGET=2900;    ! Possible values: 370, 2900.
  :
  :
%IF TARGET=370 %START
  %CONSTANT %STRING (1) NAMESEP=":"
  %EXTERNAL %ROUTINE %SPEC PROMPT %ALIAS "S#PROMPT"(%STRING (255) S)
   :
%FINISH

%IF TARGET=2900 %START
  %CONSTANT %STRING (1) NAMESEP="."
  %EXTERNAL %ROUTINE %SPEC PROMPT(%STRING (31) S)
   :
%FINISH
  :
  :
```

If a piece of code is bracketed by

```
%IF <condition> %START
  :
  :
%FINISH
```

where <condition> involves only constants and %CONSTANT %INTEGERs, then the IMP compiler will compile the code if the condition is satisfied and will ignore the code if it is not. In particular, the example above would *not* produce the fault "Name set twice".

[The use of the name TARGET is merely a convention which need not be followed. The use of the value 370 to represent EMAS-3 reflects the fact that EMAS-3 runs on IBM 370-type hardware; again, some other value can be used if preferred.]

Although the example above shows only declarations being the subject of conditional compilation, executable statements can also be handled in this way. However, the code

within such a %START/%FINISH block must be syntactically complete: it is not valid to have a %BEGIN or %CYCLE, etc. within the conditional block without also having the matching %END or %REPEAT, etc. in the same block.

One consequence of this requirement is that when you wish to produce a common source for a set of %EXTERNAL %ROUTINEs which are EMAS commands, it is necessary to use the following approach:

```
  :
%ROUTINE INNER ROUTINE
  :
  ! Your program goes here (apart from the parameter checking and analysis).
  :
%END; ! Of %ROUTINE INNER ROUTINE.

%IF TARGET=2900 %START
  %EXTERNAL %ROUTINE WIDGET(%STRING(255) S)
      :
      ! Decompose and analyse parameter string
      :
    INNER ROUTINE; ! This routine does the work.
      :
  %END; ! Of %EXTERNAL %ROUTINE WIDGET.
%FINISH

%IF TARGET=370 %START
  %EXTERNAL %ROUTINE WIDGET %ALIAS "C#WIDGET"
      :
      ! Decompose and analyse parameter string (using PAM of course)
      :
    INNER ROUTINE; ! This routine does the work.
      :
    SET RETURN CODE(FLAG)
  %END; ! Of %EXTERNAL %ROUTINE WIDGET.
%FINISH
```

It is necessary to do it this way because conditional compilation does not allow you to give an alternative for the first line only of an %EXTERNAL %ROUTINE.

Once you have dealt with all the system-specific parts of your program, all that is necessary to obtain a program suitable for one or other of the EMAS systems is to set the TARGET variable appropriately before compilation (on the correct system). Experience shows that it is best to keep the single source program *on one system only*. When a change is required, the source should be updated, copied (transferred) to the other system, and then compiled there with the appropriate value for TARGET (or whatever) selected. This copy of the source should then be *destroyed*.

References

The IMP80 Language, 1982
User Note 2: IMP80 on EMAS 2900: Differences from IMP9
User Note 14: SOAP80: A program for formatting IMP80 source programs
User Note 29: How to register as a user of ERCC services
User Note 60: EMAS-3: Changes from EMAS 2900 for all users
User Note 61: EMAS-3: Naming files
User Note 62: EMAS-3: Writing commands
User Note 80: EMAS-3: Subsystem language independent programming interface
User Note 85: EMAS-3: Program loader

Acknowledgements

Some of the text of Section 2 was taken from an earlier version of this Note, written by Roderick McLeod. I am also grateful to several colleagues for their comments before and after the writing of this Note, particularly Neil Hamilton-Smith, Andrew McKendrick, Sandy Shaw, Peter Stephens, Nick Stroud and Keith Yarwood.

Example

The first program below is the EMAS or BUSH IMP80 code for the transfer commands TOEMAS, TOEMASA, TOBUSH. The second program is the equivalent on EMAS-A.

EMAS or BUSH

```
%routine toemasx(%string (255) s, host)
   !Transfer a file to another emas machine
   !parameter = file,owner.newname,pass     (2nd param may be null)
   !optional last param "overwrite" allows overwriting of an existing file
   %external %string %fn %spec uinfs(%integer entry)
   %record %format rf(%integer conad, filetype, datastart, dataend)
   %system %routine %spec connect(%string (31) s, %integer access, maxbytes,
            protection, %record (rf) %name r, %integer %name flag)
   %system %string %fn %spec failure message(%integer flag)
   %system %routine %spec psysmes(%integer root, flag)
   %record (rf) r
   %string (255) newname, pass, file, overwrite
   %string (63) t1, t2, owner, this host
   %integer flag
   %external %routine %spec transfer(%string (255) s)

   %routine fail(%string (63) s)
      newline
      printstring("TO".host." fails - ".s)
      newline
      %stop
   %end;                                    ! OF FAIL

   this host = uinfs(16)
   %if this host->t1.(".").t2 %then this host = t2
   %if host=this host %then fail("this host is ".host)
   %unless s->file.(",").newname.(",").pass %then fail("Too few parameters")
   %unless pass->pass.(",").overwrite %then overwrite = ""
   %if ""#overwrite#"OVERWRITE" %and "MAKE"#overwrite#"FILE" %and overwrite#"REPLACE" %c
                     %then fail("OVERWRITE?")
   %if newname->owner.(":").newname %or newname->owner.(".").newname %then %start
      %if length(owner)#6 %then fail("Invalid owner ".owner)
   %finish %else %start
```

```
      owner = uinfs(1)
      %if newname="" %then %start
         %if file->t1.(".").t2 %then newname = t2 %else newname = file
      %finish
   %finish
   %unless 1<=length(newname)<=31 %and 'A'<=charno(newname, 1)<='Z' %then %c
                     fail("Invalid new name ".newname)
   %if newname->t1.("_").t2 %then fail("Invalid new name ".newname)
   %if length(pass)=0 %then fail("Invalid password ".pass)
   connect(file, 0, 0, 0, r, flag)
   %if flag#0 %then psysmes(8, flag) %and %return
   %if overwrite="" %then overwrite = "MAKE" %else %start
      %if overwrite="OVERWRITE" %then overwrite = "FILE"
   %finish
   transfer(file.",".host."(".owner.",".pass.")".newname.",".overwrite)
%end;                                          !OF TO70

%external %routine tobush(%string (255) s)
   toemasx(s, "BUSH")
%end

%external %routine toemas(%string (255) s)
   toemasx(s, "EMAS")
%end

%external %routine toemasa(%string (255) s)
   toemasx(s, "EMAS-A")
%end

%end %of %file
```

EMAS-A

```
%routine toemasx(%string (255) host)
   !Transfer a file to another emas system
   !parameters = file,owner:newname,pass     (2nd param may be null)
   !optional last param MODE controls whether an existing file may be overwritten
   !May be MAKE, FILE, REPLACE (or OVERWRITE) -  by default, MAKE
   %external %routine %spec emas3 string(%string %name vector, value)
   %external %routine %spec psysmess %alias "S#PSYSMESS" %c
                                    (%string (255) caller, %integer flag)
   %external %string %function %spec uinfs %alias "S#UINFS"(%integer n)
   %external %routine %spec emas3 set fname(%string %name text)
   %string (255) newname, pass, file, mode
   %string (63) t1, t2, owner, this host
   %integer flag
   %external %routine %spec emas3(%string %name command, params,
                                  %integer %name flag)
   %routine fail(%string (63) s)
      emas3 set fname(s);                  !the text of the error message
      psysmess("TO".host, 100);            !also sets return code
      %stop
   %end;                                  ! OF FAIL

   this host = uinfs(16)
   %if this host->t1.(".").t2 %then this host = t2
   %if host=this host %then fail("this host is " host)

   ! Now call PAM.
   emas3 string("Local file;fileormem,read,exist", file)
   emas3 string("Remote file;file;".file, newname)
   newname -> owner.(":").newname; !Full filename returned by emas3 string.
   emas3 string("Password;any,secret", pass)
   emas3 string("Mode;word,MAKE,REPLACE,FILE,OVERWRITE;MAKE", mode)
      %if mode="OVERWRITE" %then mode = "FILE"
      emas3("TRANSFER", file.",".host."(".owner.",".pass.")".newname.",".mode, flag)
      psysmess("TO".host, flag);             !to set return code
   %end;                                     !OF TOEMASX

   %external %routine tobush %alias "C#TOBUSH"
      toemasx("BUSH")
   %end

   %external %routine toemas %alias "C#TOEMAS"
      toemasx("EMAS")
   %end

   %external %routine toemasa %alias "C#TOEMASA"
      toemasx("EMAS-A")
   %end

   %end %of %file
```