UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 62

(February 1986)

Title:

EMAS-3: Writing Commands

Author: Tony Gibbons, Sandy Shaw

Contact: Advisory service

Software Support Category: n/a

Synopsis

This Note describes the features of EMAS-3 commands and gives details of how to write them. These features include automatic prompting for missing parameters, built-in help, type and access checking, and multiple invocation of commands from the expansion of wild-card filenames.

Keywords

Commands, Parameter Acquisition Mechanism (PAM)

Edinburgh Regional Computing Centre

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

Contents

This Note describes the features of EMAS-3 commands and gives details of how to write them. These features include automatic prompting for missing parameters, built-in help, type and access checking, and multiple invocation of commands from the expansion of wild-card filenames.

1. Calling Commands

The following examples illustrate the various ways in which a command can be called at the terminal. The facilities apply to **all** commands on EMAS-3.

1. *Command:* `analyse myfile`

   This shows the straightforward case, where the required parameters are specified on the same line as the command, as on EMAS 2900.

2. *Command:* `analyse`
   File: `myfile`

   Here, EMAS prompts for the required parameter *File:* and supplies defaults for the other parameters.

3. *Command:* `analyse ?`
   File: `myfile`
   Option():
   Output(.out):

   By specifying "?", the user requests that every parameter should be prompted for. Where the parameter is optional, its prompt indicates a default value in parentheses. Pressing RETURN causes that value to be used.

   a. *Output(.out):* `?`
      <help text about the Output parameter>
      Output(.out):

      Typing "?" in response to a prompt causes help information about the parameter to be displayed.

   b. *Option():* `<control+Y>`

      A response of `<control+Y>` to any prompt causes the call to be abandoned and a return to command level.

4. *Command:* `analyse ??`
   <help text about the command>
   File:

   This call causes help text for the whole command to be displayed, followed by prompts for each parameter.

5. *Command:* `analyse myfile , output=.lp`

   As an alternative to entering parameters in their fixed order, when prefixed `keyword=`, they can be given in any order. This is useful for commands with long parameter lists (the standard command OPTION is usually called with

keyword parameters for this reason, e.g. `OPTION RECALL=PERM`). After the first keyword parameter encountered, all the remaining parameters must also be specified by keyword. The keyword name is the same as the first word of the prompt, but may be typed in an abbreviated form. For example `ou=` is equivalent to `output=`.

The following examples show ways in which multiple calls of a command can be specified in a single command line.

6. *Command:* `analyse teach.*`

   Where a wild-card expression is given, the set of matching files is found and the command is called in turn for each file in the set. In this example, all the files and groups in group `teach` are analysed (the asterisk matches any sequence of zero or more characters).

7. *Command:* `rename tx* , text*`

   For commands with two or more filename parameters, a wild-card expression for the first parameter may require a corresponding expression for the others. In this example, all files and groups beginning with `tx` are renamed to begin `text`.

8. *Command:* `insert br1y & br2y & aby , mydirectory`

   Another way of generating multiple calls is to use ampersanding. The above call causes three files to be inserted in `mydirectory`.

9. *Command:* `insert br*y & aby , mydirectory`

   Ampersanding and wild-carding may be combined. This call is similar in effect to the example before it.

10. *Command:* `analyse @flist`

    In this example, the character file `flist` is read and each line in turn is used as the parameter list of a separate call of the command.

2. Command structure

A simple command which reads a file of input and generates a file of output is given:

```
%externalroutine make %alias "c#make"
   %externalroutinespec emas3 integer(%stringname vector, %integername value)
   %externalroutinespec emas3 set return code(%integername flag)
   %integer inchan, outchan, flag
   emas3integer("Input;route,in,char;?;The file of directives", inchan)
   emas3integer("Output;route,out;.out;?;Where to output the report", outchan)
   select input(inchan)
   select output(outchan)
      {the command does its work}
   emas3 set return code(flag)
%end
```

The points to note are:

- the entry point name begins `C#` to distinguish commands from other routines (although when called at the terminal, the `C#` prefix is never included). This is part of the general naming scheme in use for all EMAS-3 entry names, intended to remove some of the problems of name conflicts experienced on EMAS 2900. See section 11 for details of how to set this form of name in commands written in languages other than IMP.

- the parameters are obtained using (in this case) the routine EMAS3 INTEGER rather than by a `%string(255)` parameter to MAKE.

- The command does not need to perform any checking on the parameters.

- a return code is set before exit from the routine. This does two things:

  * allows job control statements to check the success or failure of command calls, and act accordingly;

  * prints an appropriate error message if the flag is positive.

  See section 9.

- although this example is written in IMP, the same framework is appropriate for commands written in other languages, such as Fortran.

3. The Parameter Acquisition Mechanism - PAM

A command gains the benefits of the features described in the above examples by making use of PAM. There are three PAM routines, used to acquire parameters of types *string*, *integer* and *longreal*. In calling a PAM routine, the command describes the parameter it requires and nominates a variable to receive the given value. A call is made for each parameter in turn, the order of call determining the order in which they are expected from the user.

No failure flag is returned to the command by any of the PAM routines. If a parameter is faulty, i.e. does not satisfy the qualifier in the vector supplied to the PAM routine, and the parameters are being acquired *by prompting*, the user will receive an error message and be prompted for a correct value. If the parameters are supplied on the command line, or if the command is running in background mode, then a failure message is printed and an immediate return to command level is made. This is all controlled by PAM; the command writer does not need to include any code to handle this.

The three PAM routines share a standard format:

```
(PAM vector, local variable)
```

In a simple case, a Fortran command which expects two parameters might make PAM calls:

```
call emas3string('Input file', cfile)
call emas3string('Operation', ops)
```

which simply nominate a prompt string and local variable name for each.
On executing each of these calls, PAM extracts the next parameter from the command line (or prompts the user if it is not in the command line) and then checks the validity of the value given.

In the example shown above, the PAM vector contains only a prompt field. Normally, it will contain additional fields to enable PAM to give help information, and to perform checking on the parameter. A complete PAM vector contains four *fields*, separated by semicolons, and is encoded as a text string:

```
prompt; qualifier; default; help
```

Examples of PAM vectors are given in the following section, and PAM fields are described in detail in section 5.

4. Typical PAM calls

To anticipate the detailed description of PAM vectors given in the following section, here are examples of calls for the most commonly required parameters.

1. `emas3integer("Input;route,char,in;?;The file to be processed", inchan)`

   returns a channel number associated with the file specified by the user, which can be used in an IMP `select input` statement or a Fortran `READ` statement. The PAM vector calls for a file which must exist, and have filetype "character"; the name of the file is not returned, only the channel number used to read it. The "?" in the vector indicates that no value default is specified.

2. ```
   emas3integer("Output;route,char,out;.out;". %c
            "The output file or device", outchan)
   ```

   returns a channel number associated with a file or device, valid for output. The `char` tag is an extra check that if there is an existing file of the specified name with a filetype other than character, it will not be overwritten. The tag `data` can replace `char` where appropriate. The default value specified, `.out`, causes output to be directed to the terminal if no parameter is specified by the user.

3. `emas3string("File;fileormem,read,char;?;The file of records", fname)`

   returns the full name of a file, or member of a partitioned file, checked for read access permission and filetype "character". This contrasts with the previous examples where a channel number is returned.

4. `emas3string("Outfile;file,write;xreport;The report file", freport)`

   returns a filename (`xreport` by default) for which write access is required.

5. `emas3string("Operation;word,make,replace,append", ops)`

   returns a string containing one of the three words specified. In this example no default is defined. The user can specify an abbreviated form of the word, so long as no ambiguity results. Hence the value `m` would match `make`. Matching is insensitive to the case of the letters of the words in the vector and of the word typed by the user. The matched word is returned to the command in upper case.

6. `emas3integer("Operation;word,make,replace,append;make", opi)`

   returns an integer value, 0 for `make`, 1 for `replace` and 2 for `append` depending on which the user specifies. In this case `make` is nominated as the default if the user does not specify a value.
   ```

5. The PAM vector

As outlined in section 3, the command writer describes the parameters the command requires by specifying a PAM vector for each. On each call of a PAM routine, the associated vector governs the behaviour of PAM in acquiring a value for the parameter. The PAM vector is encoded as a text string with up to four fields:

```
prompt; qualifier; default; help
```

The fields are described below.

5.1 Prompt field

The prompt field is displayed to the user when PAM requests input from the terminal for a parameter. This occurs when:

- the user fails to supply a value on the command line for a mandatory parameter (i.e. one with no default);
- the user explicitly requests prompting by calling the command with the single parameter "?" or "??".

The first word of the last line of the prompt is taken as the parameter's `keyword`, used where the form of call `keyword=value` is employed (see section 1.5). If the prompt field contains a newline, then the first time it is issued the whole text is displayed, but subsequently only the last line is given.

5.2 Qualifier field

The qualifier field is particularly important. It indicates which values are acceptable for the parameter. This enables checks to be made on file and device names; on filetype, access permission and existence; or, for integer values, checks on specific values or ranges of values. The qualifier can be extended by appending tags to indicate additional checks. For example, the qualifier `file` may be made to check that the file exists by adding the appropriate tag: `file,exist`.

As an alternative to the set of built-in qualifiers, you can nominate your own checking routine to perform non-standard checks; this is described in section 13.

5.3 Default field

If the command writer provides a value in the default field, then this is returned when the user does not offer a value. If the parameter is mandatory, then the default field should contain "?" to indicate that no default is available. When a user is prompted for a parameter and a default is available, it is shown in brackets:

Continue(yes):

Where the user omits a parameter and a default is available, then normally it is used directly without further reference to the user. However, it is occasionally desirable to have a parameter behave like a mandatory parameter while nevertheless possessing a default value. The effect is that a prompt is issued if no value is specified. The user can indicate acceptance of the default by pressing RETURN in response to the prompt. This conditional defaulting is selected by enclosing the default field in parentheses:

```
"Input file;file;(t#list);some help"
```

If a null value is acceptable for a parameter, the tag `ornull` should be given. If, in addition, the default value is itself null, this should be explicitly shown:

```
"Output;file,ornull;;some help"
```

In response to a prompt, a null value can be typed as "". This may occasionally be useful when a null value is acceptable for a parameter, but a non-null default is defined for it.

5.4 Help field

The help field is displayed when the user replies "?" in response to a prompt. Newlines may be included in the text of the field. An alternative to supplying the text of the help information explicitly, is to nominate a `helpfile` for the command. Indeed, this is the required method of providing help information about the whole command, when the user requests this by entering the single parameter "??" (see section 1.4). A helpfile is a character file constructed as a View file (see User Note 9). One helpfile may contain entries for several commands, with a "view section" for each. Within the section describing each command, there should be a subsection for each of its parameters. The help text for the ACCEPT command is as follows:

```
!<ACCEPT
   Command:ACCEPT file[,newname]

This command is used in conjunction with the OFFER command to
transfer a file from one user to another.  First the owner must OFFER
the file to you.  At any time after it has been OFFERed you can use
this command to ACCEPT it.  If the command succeeds the file will be
transferred into your file index.
!<FILE
The full name of a file which has been OFFERed to you by another user.
!>
!<NEWNAME
Use this parameter if you want to change the name of the ACCEPTed
file.  Otherwise the only part of the name that will change will
be the username.
!>
!>
```

To designate a helpfile, the text of the help field should appear in the following form within the vector:

```
call pamhelp(ercc90:myhelpfile)
```

In general, a helpfile need only be specified in the vector of the first PAM routine call made by a command. Its value is remembered for all subsequent PAM routine calls made for the command.

For a command with no parameters, no PAM routine call is made and therefore no help field is defined. The procedure EMAS3 HELPINFO provides an alternative method of conveying a help field to PAM. The help field given is used only if the user requests help about the command by calling it with the special parameter "??". The specification of the routine is:

```
%external %routine %spec EMAS3 HELPINFO(%string %name HELP FIELD)
```

It is also possible to nominate your own help routine to supply the necessary information. See section 12.

5.5 Keyword field

As described in section 5.1, the keyword of a parameter is taken from the first word (of the last line) of the prompt field. Exceptionally, where this value is unsuitable, an additional field can be appended to the PAM vector, following the help field and separated from it by a semicolon, to define the keyword. For example:

```
"File;fileormem;?;The type4 entries;type4"
```

5.6 Missing PAM vector fields

If a field is omitted from the PAM vector, a value is allocated as follows:

prompt	is set to	Integer (or String or Long Real, depending on the name of the PAM routine called)
qualifier	is set to	any
default	is set to	? (i.e. no default)
help	is set to	the qualifier field

6. Getting a string parameter

The routine for obtaining the value of a string parameter is EMAS3 STRING. Its specification is:

```
%external %routine %spec EMAS3 STRING(%string %name VECTOR, VALUE)
```

The qualifier field in the vector is used to determine whether the parameter value supplied by the user is valid. The possible qualifiers are:

qualifier	valid values
any	any non-empty string
group	a syntactically correct group name
fileormem	a syntactically correct file or pdfile member name
file	a syntactically correct file name
fileordev	a file name or a network device name
user	the name of a registered user
device	a device name
filelist	one fileormem, or several joined by '+' signs
date	a date, in the format dd/mm/yy

word,text0,text1,text2,...
: the parameter supplied is compared with text0, text1, etc. If an unambiguous match is found, ignoring case and spaces, the matching text in the qualifier is returned. For example, given the qualifier

```
word,no,yes
```

a parameter value of ye would yield the result YES. The result is

returned in upper case. A null value is defined by making one of the `text` options null, e.g. `word,,yes,no`.

`lcword,text0,text1,text2,..`
operates like `word` except that the result is returned in lower case.

6.1 Tags for string qualifiers

Qualifiers may be modified by `tags`. Tags follow the qualifier and are separated from it, and from each other, by commas.

String parameters are normally returned in upper case with spaces removed. This can be varied by use of the following tags:

`verbatim`	the result has case and spaces preserved
`lc`	the result is returned in lower case

For files, pdfile members and groups, the existence of the object and access permissions to it are checked by the following tags:

`exist`	the file, member or group must exist
`notexist`	the file, member or group must not exist
`read`	read access must have been permitted (implies `exist`)
`write`	write access must be allowed (if the file exists)

The following tags apply a filetype check to a file or member, if the file or member exists. In themselves, they do not imply an `exist` check.

`object`	an object file
`nonst`	a non-standard file (not `object`, `char`, `data` or `pdfile`)
`char`	a character file
`data`	a data file
`pdfile`	a partitioned data file
`dir`	a directory file (associating entry names with object file names)

If none of these tags is set, the parameter is merely checked for correct syntax.

The `join` tag applies to the `filelist` qualifier only:

`join`	concatenates the component files to a temporary file whose name is returned. If only a single file is specified, no copy is made of it. Only character files may be `joined`.

All parameters are subject to ampersand expansion (see 1.8). To forbid this, the following tag is used:

`noamp`	the parameter does not permit ampersand expansion

All file, member and group parameters are subject to wild-card expansion (see the examples in section 1). This can be controlled by the following tags:

nowild	the parameter does not allow wild-card expansion
cowild	if a wild-card expression is given for this parameter, then all other parameters with tag cowild require a corresponding wild-card expression.

Certain standard parameter values may be acceptable even though the value supplied does not pass the check implied by the qualifier, for example:

```
file,orequals
any,ornull
```

These parameters are catered for by the following tags:

ornull	a null value is acceptable
orequals	a value of "=" is acceptable
or.all	a value of .ALL is acceptable
or.end	a value of .END is acceptable
or.null	a value of .NULL is acceptable.

7. Getting an integer parameter

If the user is required to supply an integer value for a parameter, then the command writer should use the PAM routine EMAS3 INTEGER to retrieve it. However, if the outcome of a PAM check on a string value is most naturally represented as an integer, then EMAS3 INTEGER should again be used. For example, a parameter might have three valid values, YES, NO, DONTKNOW. By using EMAS3 INTEGER and setting up the qualifier part of the vector appropriately, the command writer can make PAM check for these values and return 0, 1 or 2 respectively. See the qualifier word, below, for details.

Another situation where EMAS3 INTEGER is appropriate is in the setting up of a route (in EMAS 2900 terminology, defining a file or device). When the user supplies a file name or device name, EMAS3 INTEGER can check the validity of this parameter and return a channel number, which the command could then select for input or output. See the qualifier route, below, for details.

The specification for EMAS3 INTEGER is:

```
%external %routine %spec EMAS3 INTEGER(%string %name VECTOR,
          %integer %name VALUE)
```

The qualifier in the vector is used to determine whether the parameter value supplied by the user is valid. The possible qualifiers are as follows:

qualifier	**valid value**
any	any integer value
m:n	a range of numbers. m and n are the bounds; a null value means no lower (or upper) bound; for example

6:10 means a number between 6 and 10

3: means a number greater than or equal to 3

:3 means a number less than or equal to 3

a sequence of ranges or values in numerical order
for example

0:3,5,8,10:12 means
0,1,2,3,5,8,10,11 or 12

word,text0,text1,text2,...
the parameter supplied is compared with text0, text1, etc. If a text matches the parameter value or starts with the parameter value, ignoring letter case, then the index of the text is returned.
For example, given the qualifier

word,no,yes

a parameter value of N or NO or n etc. would give 0, while y or ye or yes would give 1. A null value is defined by making one of the text options null, e.g. word,no,yes,.

route a file or pdfile member name, or a device name

7.1 Tags for the route qualifier

A number of tags are provided to modify the operation of the route qualifier. Tags follow the qualifier and are separated by commas. See the examples in section 4.

As described above, the route qualifier allows the user to specify a file or device name. PAM associates this with a channel number which it returns to the command as an integer value. This may be used in a subsequent call of SELECT INPUT or SELECT OUTPUT or a Fortran READ or WRITE statement.

The normal procedure is to allow PAM to choose a free channel number. It is possible, however, to preset a fixed channel number in the VALUE parameter of EMAS3 INTEGER, and indicate this by use of the following tag:

preset the file or device is associated with the channel number given in the VALUE parameter

Additional checks are performed when a tag is included which indicates whether the route is to be used for input or output:

in the file or device is to be used for input. Hence a check is made that either a valid input device, or an existing file with read access permitted, is specified.

out the file or device is to be used for output. A check is made that either a valid output device, or a filename which can be written to, is specified. The file need not exist, in which case it must be possible to create it; if it does exist however, it must have write access permitted.

If the file does exist, an additional check can be made on its filetype to prevent an attempt to read from a file of the wrong type, or to overwrite an existing file of a different type.

`char` a character file

`data` a data file

`object` an object file

`nonst` a non-standard file (mapped file)

Additional tags can be used to specify further details of the file definition. They have the same effect as the similarly named parameters of the DEFINE command.

`size= kb` where `kb` is a number in the range 1 to maxfilesize, giving the maximum allowed size for the (output) file

`Vn` defines V record format with a maximum of `n` bytes per record

`Fn` defines F record format with `n` bytes per record

`C` defines a character file (for Fortran only)

8. Getting a long real parameter

The routine for obtaining the value of a long real parameter is EMAS3 LONG REAL. Its specification is:

```
%external %routine %spec EMAS3 LONG REAL %c
        (%string %name VECTOR, %long %real %name VALUE)
```

The only qualifier provided is any.

9. Setting a Return Code

When a command has done its work it should set a return code before exiting. This is done by use of the routine EMAS3 SET RETURN CODE, which has the specification:

```
%external %routine %spec EMAS3 SET RETURN CODE(%integer %name FLAG)
```

If the command has failed, `FLAG` should be set to some non-zero value. If the failure is as a result of a Subsystem procedure failing (e.g. EMAS3 CONNECT or EMAS3 RENAME), then the return code should be set to the flag value returned by the Subsystem procedure. The effect of doing this is to cause an appropriate failure message to be printed on the user's terminal, of the form

command fails *failure message*

An arbitrary text may be placed in *failure message* by setting `FLAG=100` and setting the text required using the following routine:

```
%external %routine %spec EMAS3 SET FNAME(%string %name TEXT)
```

10. Additional PAM routines

A number of routines are provided to enable a command to perform manipulation of the command line.

- `%external %routine EMAS3 GETRESTOFLINE(%string %name LINE)`

  returns the list of parameters remaining on the line, or the complete parameter line if none have been read yet. The routine is not destructive, and can be followed by calls of EMAS3 STRING or EMAS3 INTEGER as required.

- `%external %routine EMAS3 LINE(%string %name LINE)`

  again returns the remainder of the parameter line, but clears it in addition.

- `%external %routine EMAS3 LCLINE(%string %name LINE)`

  operates like EMAS3 LINE, but returns the parameter line in the same form as specified by the user. Hence spaces and case are preserved.

- `%external %routine EMAS3 SETLINE(%string %name LINE)`

  sets the value of the current parameter line.

- `%external %routine EMAS3 LASTPARAM(%string %name PARAM)`

  returns the most recently acquired parameter in the form of an uninterpreted string. If the previous PAM routine acquired an integer parameter with qualifier `route`, then EMAS3 LASTPARAM would return a string containing the file or device name specified by the user. If the previous routine called for a string with qualifier `filelist,join`, which returns the name of a temporary concatenated file, then EMAS3 LASTPARAM would return the original filelist parameter value.

The routines can be used to preview the parameters before formally acquiring them. The preview may be required to guide the subsequent parameter acquisition process. Alternatively, the parameter line may be amended, and additional parameters added, before EMAS3 SETLINE is used to put it back in place. Finally, the standard PAM routines are called to acquire each parameter in turn.

11. Commands written in other languages

The routines described in this note may be called by a command written in any language available on EMAS-3. The only amendment required is to convert the name of the procedure to the corresponding `C#name` form, so that it is explicitly recognized as a command. This is done by using MAKECOMMAND. The parameters to MAKECOMMAND are the name of the object file followed by zero or more procedure entry names. A Fortran object file may contain one or more subroutines, but not a main program entry. If no procedure entry names are specified, then MAKECOMMAND converts all procedure entry names in the object file; otherwise MAKECOMMAND converts only the specified entry names.

Command: `makecommand fortobj`

Command: `makecommand object2 , mylist , mycount , ftest`

An example of a command written in FORTRAN which uses PAM follows:

```
      subroutine perman
      character*128 c
      character*130 d
      call emas3string('Filename;fileormem',c)
      d = c//',p'
      call emas3('analyse',d,iflag)
c     {other analysis of the file}
      call emas3setreturncode(iflag)
      end
```

12. User-provided help routines

A command writer can nominate his own routine to provide help information for a command and its parameters. The routine should be written with a specification:

```
%external %routine %spec MYHELP(%string %name COMMAND, KEY)
```

The name of the routine is placed in the `help field` of the PAM vector (section 5.4) as follows:

```
"File;fileormem;?;call myhelp"
```

If this routine is called by PAM (because the user asks for help), the first parameter gives the name of the command about which help is requested. The `C#` prefix is not included in the command name. If information on the whole command is required (i.e. the user typed "??"), then the second parameter is null; otherwise it contains the `keyword` of the relevant parameter.

13. User-provided checking routines

A command writer can choose not to use the standard qualifiers and tags in the qualifier field of the PAM vector, but supply his own routine to perform parameter validation. This routine should have one of the following specifications, depending on the type of the parameter:

```
%external %routine %spec MYSCHECK(%string %name COMMAND, KEY, INPUT
        %string %name OUTPUT VALUE, %integer %name FLAG)

%external %routine %spec MYICHECK(%string %name COMMAND, KEY, INPUT
        %integer %name OUTPUT VALUE, %integer %name FLAG)

%external %routine %spec MYLRCHECK(%string %name COMMAND, KEY, INPUT
        %longreal %name OUTPUT VALUE, %integer %name FLAG)
```

`command` gives the name of the command in which the parameter to be checked occurs

`key` gives the name of the parameter to be checked

`input` gives the text of the parameter typed by the user

`output value`
receives the result of evaluating `input`. The type of this parameter must agree with the type of the PAM routine called - EMAS3 STRING, EMAS3 INTEGER or EMAS3 LONGREAL.

`flag` conveys an indication of the success or failure of the check. Zero indicates success. Any other value causes a failure message to be printed.

The name of the user-supplied checking routine is placed in the `qualifier field` of the PAM vector (see section 5.2) as follows:

```
"File;call mycheck;?;The file of names"
```

14. Interim conversion of EMAS 2900 commands

Existing EMAS 2900 commands obtain their parameters directly:

```
%external %routine make(%string(255) param)
```

As an interim measure, the complete parameter line can be obtained on EMAS-3 as shown in the following example:

```
%external %routine make %alias "c#make"
%external %routine %spec emas3 line(%stringname line)
%string(255) line, in, out
   emas3 line(line)
   {analysis of parameters in EMAS 2900 style:}
   %unless line-> in.(",").out %start
      printstring("Output parameter missing"); newline
      %stop
   %finish
   {etc.}
%end
%endoffile
```

A command of this type is non-standard. It lacks the PAM features of prompting for missing parameters, built-in help, type and access checking, and multiple invocation from the expansion of wild-card filenames. Users are advised to convert to PAM.