UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 81

(February 1986)

Title:

EMAS-3: Command macro scheme

Author: John Maddock, Keith Yarwood

Contact: Advisory service

Software Support Category: B

Synopsis

Command macros provide execution of a series of command lines by typing a macro name instead of a command name, and allow textual substitution of parts of the command lines as well as conditional and repetitive execution of command lines.

The scheme currently implemented is provisional. Changes in syntax may be made during the coming months, but broadly equivalent facilities will continue to be available. Changes will be notified by ALERT on EMAS-A and by reissue of this User Note.

Keywords

Command macros

Edinburgh Regional Computing Centre

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

Introduction

Command macros provide:

- execution of a series of command lines by typing a macro name instead of a command name;
- textual substitution of parts of the command lines according to the values of the macro parameters and macro variables;
- conditional execution of command lines according to the values of the macro parameters, macro variables and system variables;
- repetitive execution of command lines under control of macro variables and system variables.

The scheme currently implemented is provisional. Changes in syntax may be made during the coming months, but broadly equivalent facilities will continue to be available. Changes will be notified by ALERT on EMAS-A and by reissue of this User Note.

Macro definition and textual substitution

Macros are character files, usually being members of a partitioned file :SS#MACRO. (":" is an abbreviation for "username:" for the current process owner). The first line of a macro is the macro header, comprising the name of the macro followed by formal parameters. The formal parameters are simply alpha-numeric names. They are separated from the macro name by one or more spaces, and from each other by commas. Each formal parameter may be followed by a phrase `=text` where `text` is the default text to be used during macro interpretation when the call of the macro does not provide an explicit value for the parameter. For example, the macro header of macro FORT (being partitioned file :SSMACRO_FORT) might be:

```
fort sourcenumber , outputfile=.lp
```

The remaining lines of the macro, called the macro body, are EMAS command lines in which text of the actual parameters given in the call of the macro is substituted in the command lines where the parameter names appear preceded by `%%`.
For example:

```
fortran sourcepd_member%%sourcenumber , object , listing
define 6 , %%outputfile
run object
```

During macro execution, the `%%`-parameter text is replaced by the text of the corresponding parameter in the call of the macro. Thus the macro given above might be invoked by typing

```
Command: fort 25 , .out
```

which would lead to the execution of the commands

```
Command: fortran sourcepd_member25 , object , listing
Command: define 6 , .out
Command: run object
```

Some examples of macro definitions and executions are given in file SUBSYS:EXAMPLES.MACROS

Lines of the macro body may in addition be macro variable declarations, labels or conditionals, as described below.

Macros may be nested within macros to any depth. A macro defined within a macro is local to the macro containing it, and is *not* stored as a separate member of :SS#MACRO.

Creating and altering macros

A macro may be created and optionally placed in :SS#MACRO by using an editor. It may also be created directly from typed input by using one of the following forms, in which case the macro header is created automatically, the file is placed in :SS#MACRO and the macro name is inserted into the current active directory. The member may subsequently be edited in the ordinary way.

The commands INSERT, REMOVE may be used to enable macro names to be inserted explicitly into a directory if necessary. The forms of the commands are:

```
Command: INSERT macrofilename , directory
Command: REMOVE macrofilename , directory
```

The directory parameter may be omitted, in which case the macro is inserted into the current active directory. These commands can take effect even if the specified directory is currently connected in the virtual memories of other processes.

Direct creation of a macro - short form

The short form of a macro definition is

```
(MACRO macroname param1 , param2 ,... paramN) macrobody
```

For example

```
Command: (MACRO PDEXTRACT PDFILE , MONTH ,
            LPNO) LIST %%PDFILE_%%MONTHPAY , LP%%LPNO
```

Then the command line

```
Command: pdextract data86 , january , 15
```

causes execution of the command line

```
Command: list data86_januarypay , .lp15
```

(Note that on EMAS-3, command-lines may be typed on several input lines by breaking the "command-line" after ",".)

Direct creation of a macro - long form

The full form, providing for the creation of multi-line macros is

```
(BEGIN MACRO macroname param1 , param2 , ... paramN)
   :
macrobody
   :
(END MACRO)
```

In this form, the earlier example FORT would be typed

```
Command: (begin macro fort sourcenumber , outputfile=.lp)
Command: fortran sourcepd_member%%sourcenumber , object
Command: define 6 , %%outputfile
Command: run object
Command: (end macro)
```

Instead of the line (end macro), Ctrl+Y (end-message) may be typed.

Invocation of a macro

A macro is a character file with internal format as described in this document. The file is usually but not necessarily a member of partitioned file :SS#MACRO. A macro is called by typing the macro name followed by its actual parameters, in the same form as for an EMAS command. The loader search for the macro name is analogous to that for a command entry point or alias. If the macro name has been inserted into one of the directories in the process' SEARCHDIR list, the loader returns the name of the file containing the macro and macro execution commences. If the macro name is not found during the directory search, then the partitioned file :SS#MACRO is searched and if a member exists having the same name as the typed macro name, again macro execution begins.

There are thus two ways in which the search for a macro may be successfully completed: *either* from a macro entry in a directory *or* because it exists as a member of :SS#MACRO where the member name is the macro name. If the macro was created directly from typed input by entering

```
Command: (MACRO macroname etc.
```

or

```
Command: (BEGIN MACRO macroname etc.
```

then both search criteria can be met. Having the macro name in a directory allows a more efficient search (because the directories will always be searched *before* :SS#MACRO is searched) and in addition other users can access the inserted macros using the standard SEARCHDIR mechanism via your permitted libraries. It is necessary, of course, to PERMIT the :SS#MACRO or other file containing the macro, if your macros are to be executed by other users.

Macro variables

Variables of type integer may be declared within macros. The form of declaration is:

```
.DECLARE name , name ,..., name
```

The variable names are subject to the normal rules: alphanumeric, first character alphabetic. Values may be set into these variables using the .SET macro instruction, thus:

```
.SET name=number
```

For example:

```
.DECLARE COUNT
.SET COUNT=5
```

These variables may form part of binary expressions and may be used in conditional statements (q.v.). Binary expressions may appear on the right-hand side of .SET statements; operators available are add(+), subtract(-), multiply(*) and divide(/). Thus:

```
.SET COUNT=COUNT+1
.SET COUNT=INIT*3
```

Variables may also be concatenated with text, in the same way as parameters: there is an implicit conversion to string. Thus

```
TEST%%COUNT
```

would yield TEST5 if COUNT had value 5.

.SET may also be used to set the value of a macro parameter. Note, however, that parameters are implicitly string-type variables. Whenever an assignment is made to a variable or a parameter, the appropriate conversion (to integer or string respectively) is made. It does not make sense to assign a non-numeric string into a variable, which is implicitly type integer, and such constructions will be faulted.

System variables

One system variable is currently available, being the return code from the previously executed command. Its name is .RCODE and it may be tested in a statement such as

```
.IF .RCODE=0 .GOTO NEXT
```

Read the section on Conditionals for further details. A value may not be *assigned* to .RCODE.

Jumps and labels

A macro label takes the form

```
name;
```

where name obeys the usual rules, thus:

```
LABEL;
```

Jumps are of the form .GOTO label, thus:

```
.GOTO LABEL
```

Conditionals

Conditionals take the form

```
.IF parvar comparator parvar .GOTO label
```

where parvar may be a parameter or a variable, comparator is equals(=), not equals(#), greater than(>) or less than(<). Type conversion takes place where necessary, and the comparison is *either* arithmetic *or* (IMP-) string comparison,

according to the type of the left-hand parvar.

For example:

```
.IF PARM1=VAR1 .GOTO LABEL
```

%% does *not* appear in front of the parameter name: this would imply pre-substitution with the parameter value. If the characters PARM1 or VAR1 do not correspond to parameter or variable names, the literal string(s) PARM1, VAR1 are used in the comparison.

Character data for commands

Character data may be supplied as part of the macro body. Character data are preceded by a line .INPUT and terminated by a line .ED (end data). When a command from the macro body requires input from the current default input stream, lines from the following macro data are supplied. The lines of macro data are subject to the same text-substitution as the command-lines of the rest of the macro body, if required. For example, a macro CLEANOUT might be defined:

```
cleanout bad,file
edit %%file
.input
(m/%%bad/dl)*
e
.ed
```

Then typing

Command: `cleanout warning , listfile`

would delete from file LISTFILE all lines containing the word WARNING.

Comments in macros

Comments, introduced by "--" (double minus sign), may be incorporated into the macro body. All characters from "--" to the end of the current line are ignored by the macro interpreter, except within blocks of command data (between .INPUT and .ED).

Messages in macros

The construction

```
.MSG message
```

causes printing of the text message at the terminal during macro execution.