Published: 01/09/67

Identification

Macro Facility for Command Language
G. Schroeder and D. B. Wagner

Reference

The user must be familiar with BX.1.00 Multics Command Language.

Introduction

When using an on-line system, one usually finds that there are certain sequences of commands that one issues over and over with only minor changes in arguments. The ability to define such command sequences as macros, then to invoke such a macro by typing the macro name and a list of arguments to be substituted into the command sequence, provides two major conveniences:

1) economy of typing

2) economy of thought; i.e., a complicated sequence of commands need be thought out only once; once defined, the sequence can be considered as a whole.

Perhaps the more obvious feature which must be included in a macro facility for a command language is the ability to declare certain arguments within the body of the macro to be bound variables and to substitute for these bound variables at the time the macro is invoked. Macros of commands should be definable such that the user can substitute for arguments to commands within the macro and for names of commands within the macro.

Another obviously useful feature is the ability to execute commands conditionally within a macro. One can easily envision a macro such that the failure of one command within it would cause the user to wish not to execute the rest of the sequence. It should be possible to express such a condition and have the macro behave properly when invoked.

The Control Commands

The above features are provided through the "control statements" of the macro package. These "control statements" are actually commands. Obviously the number and complexity of control statements can - and will - grow, but the following is a summary of a basic set:

macro_arg a b c ...

specifies names of the bound variables in the text following. Macro_arg causes all occurrences of the strings a, b, c, etc., which stand alone to be replaced by the corresponding arguments given in the macro call. A string is said to stand alone when it is delimited by ASCII characters that are not alphabetic or numeric and not the underscore (_).

create a b c ...

causes special symbols to be created and substituted for a, b, c, etc., in the text following. Created symbols are produced by concatenating the string "crs_" with a string obtained using the "unique identifier generator" described in BY.15.01. They are always distinct from each other and have a very good chance of being distinct from other file names, variables, etc. which are used in the macro.

Use of created symbols makes it possible to freely use temporary variables and files with names which are irrelevant outside a given macro, without worrying about naming conflicts.

The sequence

iterate x list `command sequence'

causes the text included in command sequence to be repeated several times over, once for each element of list. On each repetition the corresponding element of list is substituted for every occurrence of the string x standing alone as an element of a command. Iterate's can of course be nested to any depth.

The command

conditional x `command sequence'

causes the specified command sequence to be a part of the macro expansion only if the string x is not a string of zeros (x is normally the value of an immediate value command such as value, see below).

Two commands which will be useful in conjunction with the conditional command are setvalue and value, which maintain a data-base of variable-names and associated values.

setvalue x string

establishes x as the name of a variable with value given by string. Then the immediate value command

{value `expression'}

evaluates the arithmetic expression given, which may include variables established using the setvalue command, and returns a character-string representation of the value. In particular the command

conditional {value `a < b + c'} `line'

causes the line to be included in the macro expansion if and only if the values of the variables a, b, and c satisfy the expression a<b+c. Rules for evaluating expressions such as a<b+c will be established later.

A macro is defined by issuing the command

macro macro_name

where macro_name is the name of a file which contains a series of commands including macro control commands.

Macro will write a linkage section for the file being defined as a macro. This linkage section defines the symbol which is the name of the macro with a special class number. The macro file itself is not altered.

When the Shell attempts to build linkage to this macro the linker returns the class number of the symbol. The Shell thereby knows that a macro is being invoked.

The macro is invoked by typing

macro_name arg1 arg2 ...

Where macro_name is the name of a file containing a macro which the user has previously defined with the macro command.

As explained above, the Shell can tell that a macro is being called instead of an ordinary command. The Shell then calls the macro processor with command name (macro_name) and the arguments (arg1 arg2 ...). The Shell executes any interjected commands itself and does not pass them to the macro processor; in fact the Shell does a full syntax analysis of the command and passes the arguments to the macro processor.

The macro processor is divided into two procedures: the macro initiator and the macro executor.

The macro initiator sets up two data bases:

1) commands to be executed

2) a key to string substitutions to be made

These two data bases are shared with a procedure called the request handler (see BY.4.01), which actually performs string substitutions on demand and which also does various manipulations of these two data bases at the behest of the control commands.

The macro executor is called by the Shell when the Shell detects a macro. The macro executor calls the macro initiator to set up the two data bases described above. When the macro initiator returns, the macro executor calls the request handler to get a command line. The macro executor calls the Shell with the string returned by the request handler. The command which the Shell calls may be an ordinary command, or it may be a macro control command, or it may be another macro invocation. The Shell and the macro executor never know the difference; the macro control command simply calls the request handler to adjust its data bases. When the request handler returns, the macro control command returns to the Shell which returns to the macro executor. The macro executor then calls the request handler to get the next command, etc., until there are no more commands in the macro to be executed. The macro executor then returns to its caller, the Shell.

Figure 1 shows how the Shell, the macro executor, the macro initiator, and the request handler interact. The first data base (Q in the figure) contains a stack of "bunches" of commands. Each bunch in this stack is a list of commands - probably the text of one macro.

The second data base (Sub in the figure) contains a key to the string substitutions to be done in the macro. This is also a stack-pushed down on each occurrence of a new macro. The macro initiator sets this up from the macro_arg statements and the list of arguments given when the macro was invoked. After completing this task, the macro initiator calls an initialization entry (E1), in the request handler. When the request handler returns, the macro initiator returns to the macro executor.

The macro executor gets commands from the request handler (entry E2) until Q is empty. The macro executor calls the Shell to execute each command. When it is finished, i.e., Q is empty, the Macro executor returns to its caller, the Shell. Control commands (called by the Shell) call the request handler at another entry (E3) to make changes to Q and Sub.

There are certain commands such as the debugging commands and the context editor which are highly interactive. When executing, these commands accept requests from the user which are essentially subcommands to the command in control. The ability to include requests to a command within a macro appears to be a desirable feature. Although on the one hand, it is desirable to be able to include the requests to an interactive program in a macro, it is undesirable to be required to include the requests to an interactive program in a macro.

The user may include in the call to an interactive command in a macro the information that his requests are contained in the macro. (This could be controlled on an option.) In this case the interactive command will call the request handler for successive requests.

The request handler, of course, cannot tell the difference between commands that should go to the Shell and "subcommands" which should go to an interactive program. The request handler simply makes up the next command line from its data bases and returns it whenever it is called through its E2 entry (see figure 1). It is the responsibility of the interactive command to realize when it should get commands from the request handler and when it should not.

Interactive commands can use this facility only if they are documented in such a way that it is clear to the user that the requests to the command can be included in a macro. All commands obviously will not wish to provide the ability to include requests to the command in macros. This ability makes sense only for commands such as the debugging commands and the context editor.

Figure 2 shows how an interactive command fits in with the mechanism described in Figure 1.

It might sometimes be desirable to switch back and forth between sources of requests and commands - sometimes the macro file should be the source, then sometimes another

stream (probably that one connected with the typewriter). This can be done by an additional control command:

```
include stream_name
```

This command sets a switch in the request handler which causes it to read stream_name for successive lines instead of taking items from Q.

The control command

```
end_include
```

causes the request handler to again take items from Q.

An end-of-stream return on a call to read stream_name also causes the request handler to revert to reading Q.

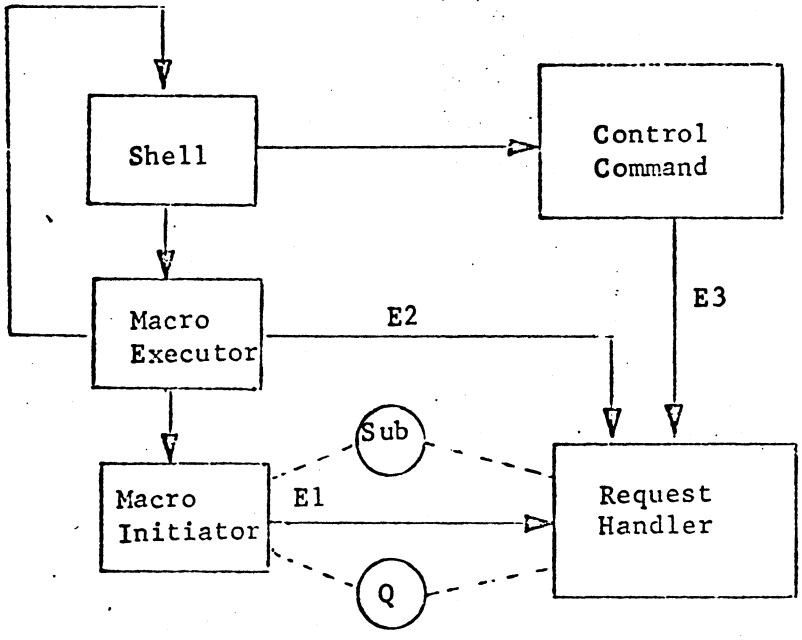
Shell
Control Command
Macro Executor
E2
E3
Sub
Macro Initiator
E1
Request Handler
Q

Figure 1

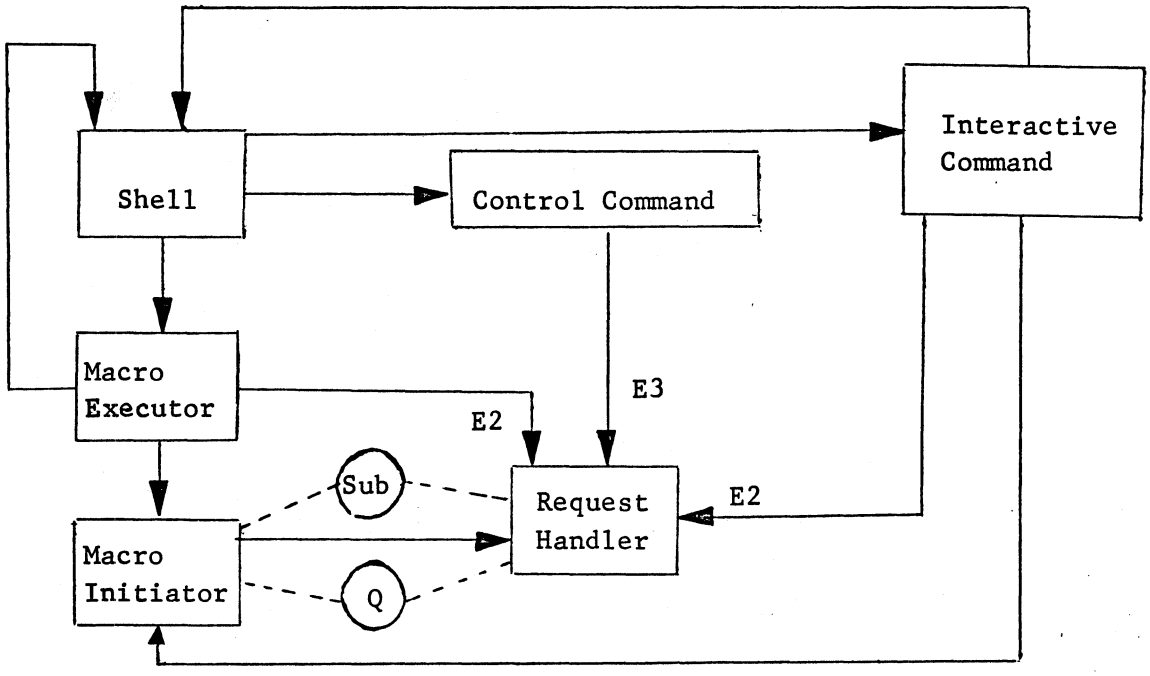
Shell
Control Command
Interactive Command
Macro Executor
E2
E3
Sub
Request Handler
E2
Macro Initiator
Q

Figure 2.