Published: 11/08/67

Identification

Macro Processor
K. J. Martin

Purpose

A macro is in fact a data segment prepared using the context editor (BX.9.01). When a user invokes a macro a certain amount of preparation is required to transform this data segment into a usable form. The macro processor does this preparation and arranges for the necessary cleanup actions following the macro. See BX.18.00 for an overview of command language macros.

Environment

The macro processor is called in at least three different situations:

1) by the Shell after it recognizes that the command invoked is actually a macro data segment;

2) by the interactive procedures, such as the debugging aids, which wish to invoke macros,

3) as a procedure specifically called by the user.

In cases 1 and 3 the macro processor should not only set up the macro, but should also call the listener (BX.2.02) to start execution of the macro. In case 2 it should return immediately after setting up the macro. A special entry point to the macro processor exists solely for case 2. Case 3 is expected to occur relatively rarely, such as when a user wants to execute a macro only once and would prefer not to process it with the macro command (BX.18.02).

Usage

```
call macro_processor (macro_name, argument_array);
```

where

macro_name is the name of the segment prepared using the context editor. It consists of command lines including regular commands, macro control commands (detailed in BX.18.03 - BX.18.08), and possibly requests to any interactive commands included.

argument_array is an array of the arguments to the macro. When the user is invoking a macro at command level, he must type in the arguments of the macros as a list which the Shell passes on as an array. A list is necessary (rather than separate arguments) because the procedure macro_processor must have a known number of arguments.

The call for case 2 is:

```
call macro_processor$setup_only (macro_name, argument_array);
```

Implementation

The major jobs of the macro processor are

1) to insure that if the Listener is called it will not return to the macro processor until the macro is completed,

2) to cause the request handler (BY.4.01) to be called to read input lines,

3) to provide the request handler with a source of input lines, namely the macro data base,

4) to provide cleanup facilities when the macro is completed,

5) to store the arguments to the macro where the macro can access them,

6) if execution as well as setup is wanted, call the listener.

Insuring that the listener will not return until it is appropriate is easier than it sounds. A quick reading of BX.2.02 will reveal that if the user wants to housekeep after each command, the listener returns to its caller after each command. Conversely, if the user does not want to housekeep the listener does not return, but rather calls to read another command line. Although the macro is not operating at command level, we can use this feature of the listener. We merely set the housekeep option to off (default is no housekeeping at command level) after pushing the options stack (to preserve previous option values).

The calls involved are:

```
call push_opt;                          /* see BY.9.01 */
call modopt ("housekeep",0,"0"b," "): /* see BY.9.03 */
```

The next step is to alert the request handler to its imminent role. The I/O localattach call (BF.1.01) attaches an ioname to the request handler and conveniently passes the attachment on to the request handler. The macro processor wishes to attach the request handler to the ioname user_input, and also remember the current user_input attachment in order to restore it later. The macro processor calls unique_chars (BY.15.01) to obtain a unique character string. It then calls io_rename (BF.1.01) to rename the ioname of the current user_input attachment to the unique character string. Thus the ioname user_input is left free.

The macro processor can then safely issue the call:

```
call localattach ("user_input","request_handler",
                  default_stream,"r",status);
```

to attach user_input to the request handler. The I/O system calls request_handler$localattach to give the request handler a chance to prepare itself. See BY.4.01 on the request handler for further details.

The macro processor stashes away the unique character string to use when the macro has finished. It will then restore the ioname user_input to its previous attachment by renaming the attachment of the unique character string back to user_input.

Before providing the request handler with the macro text, part of the future cleanup facility must be prepared. The macro processor calls request_handler$insert_line to place the command line

```
modopt `housekeep' 0 1 ` ' <NL>
```

in the request queue. After the macro text has been exhausted, this will be the next command line "read" by the listener. When the listener receives control following execution of the command it will find the housekeeping option on and will return to its caller the macro processor.

The macro processor places the macro text in the request queue with a call to request_handler$insert_file.

The only task left before calling the listener is to store the arguments to the macro where the macro can access them. The macro_arg control command uses these arguments to set up the substitution list used by the request handler. The macro processor creates and initializes the segment macro_arguments if it does not already exist. The segment is a structure declared as:

```
dcl 1 arguments based (a_ptr),
    2 current_list bit (18),
    2 space area ((131071));
```

The macro processor then allocates the following structure into a_ptr → arguments.space.

```
dcl 1 args based (arg_ptr),
    2 previous_list ptr,
    2 n_args fixed bin (17),
    2 arg_array (arg_ptr→args.n_args),
      3 sub_index fixed bin (17), /* index of latest sub-
                                     stitution of this argu-
                                     ment in substitution
                                     list */
      3 count fixed bin (17),
      3 chars char (100);
```

where arg_ptr→args.previous_list points to the previously allocated set of arguments. (Argument lists are stacked so that only one data base is needed even when macros are nested.) arg_ptr→args.n_args is the number of arguments in the list of arguments where the count indicates how many of the chars are meaningful. The third-level element sub_index, is an index of the current substitution of this argument in the substitution list of the Request Handler (BY.4.01).

Having stashed away the arguments, the macro processor calls the listener. When the listener returns, the macro processor pops the option stack 1 frame:

```
call pop_opt (0);
```

Note that this changes the housekeep option back to its previous settings. The macro processor now pops the substitution list by a call to request_handler$pop_subst (BY.4.01). It also renames the ioname having the unique character string name to "user_input", then returns. If the macro processor was called at its $setup_only entry it returns before calling the listener and depends on someone else to pop the options stack.