To: Distribution

From: Steve Herbst

Subject: V2 exec_com extensions

Date: 04/02/81

Here are five extensions proposed for Version 2 exec_com as it was documented in MCR 4857. The first two have already appeared in an MTB several years ago (MTB-324, 01/09/77).

The five extensions are:

1. Do groups (&do, &end)

2. Condition handling (&on, &begin, &condition_name, &continue_to_signal, &restart, &revert, &signal)

3. Directing output (&output, &discard, &variable)

4. Directing &print statements (&print_switch)

5. Command escape (&execute)

----

1. Do groups

The exec_com &do and &end statements provide syntactic closure for a group of statements so that the group can be executed conditionally. They do not imply any scoping of variable names or values.

For the initial implementation, &goto's into a &do-&end block are not allowed (abort execution). Outward &goto's are allowed. The forthcoming command convert_ec, which among other things indents the text of &do-&end blocks, will also check for invalid &goto's. If no other problems are encountered with &goto's into blocks, they can be added later without affecting users.

There is no restriction on transfers within &do-&end blocks, or on the nesting of &do-&end blocks, as demonstrated in the example:

```
&if &[equal &1 tape] &then
     &if &[equal &2 test] &then &do
          &if &[not [mount_tape &3]] &then &do
               &print DX: Cannot mount tape &3
               &return false
          &end
          &return &[test_tape &f3]
     &end
     &else &do
          &set result true
          &set arg_index 4
        &label arg_loop
          &if &[ngreater &(arg_index) &n] &then
               &return &(result)
          &if &[not [read_tape &2 &3 &(&(arg_index))]]
               &then &set result false
          &set arg_index &[plus &(arg_index) 1]
          &goto arg_loop
     &end
&else &if &[equal &2 special] &then &goto special
&else &do
...
```

Note that since Version 2 strips leading white space from lines, blocks can be indented as desired for readability.

2. Condition handling

Condition handling is done with an any_other handler in the exec_com or absentee listener. Data as to which conditions are handled and which blocks of exec_com text are executed is modified by the statements:

```
&on(LIST OF CONDITIONS) STATEMENT
&revert LIST OF CONDITIONS
```

where LIST OF CONDITIONS is a list of condition names separated by white space. The &on statement can be followed by a single exec_com statement on the same line or by &begin, a block of statements, and &end:

```
&on(command_error active_function_error) &begin
     ec restore_access ([dirs **.lib])
     dl CP>temp_map
     &print MAP ABORTED
&end
```

The new &begin statement is only allowed in condition handlers.

The contents of the condition handler, either one statement or a &begin-&end block, is related to the containing exec_com in the same way as a &do-&end block. It references the same variable names and values as the containing ec. Any &goto's into a handler are not allowed. Handlers can be nested as in PL/I.

The effects of &attach and &detach statements inside a handler are local to the handler.

Three more statements are proposed:

```
          &signal CONDITION
```

causes the named condition to be signalled when it is executed.

```
          &restart
          &continue_to_signal
```

inside the text of a handler exit the handler; &restart restarts execution at the point where the condition was signalled, and &continue_to_signal propagates the condition. The end of the handler is an implicit &restart. A &goto statement inside the handler returns to the stack frame of the exec_com in which the handler was established. Both &quit and &return inside a handler quit out of the containing exec_com, as in:

```
          &on command_error &quit
```

The new expandable construct &condition_name expands inside a handler to the name of the condition being handled.

3. Directing output

The &output statement with various keywords is used to control where output is directed during exec_com execution. The available usages are:

```
          &output &discard {&osw SWITCHNAME}
          &output &var VARNAME {&osw SWITCHNAME}

          &output &revert {&all} {&osw SWITCHNAME}
```

The long names for &osw and &var are &output_switch and &variable. Multiple occurrences of "&osw SWITCHNAME" are allowed to direct the output of more than one switch.

The first two statements direct output on the specified switches or on the default switch user_output. The &revert usage reverts the last occurrence of either of the first two, or reverts all previous occurrences if "&all" is specified.

Two of these statements are vaguely similar to the discard_output and revert_output commands, but in no way interact with the commands. Since they are implemented within the exec_com language, they can appear at any stage of execution, for example, while inside a command's input loop. Whereas the discard_output command can only be used to execute a single command line, "&output &discard" discards output globally until the corresponding "&output &revert" statement.

The "&output &var" usage feeds all output until the corresponding "&output &revert" onto the end of a variable's value. The value does not change until the "&output &revert" statement is executed. Expansion of the variable can then be used to insert the output anywhere in the text, or return it as the value of the exec_com active function. Note that the translate active function may be needed to replace newlines in the value with spaces, as in the example:

```
&set line_numbers &""
&attach
&trace &command &input off
&output &var line_numbers
qx
r &1
g=/&f2/
q
&output &revert
&return &[translate &r(line_numbers) "&SP" "&NL"]
```

4. Directing &print statements

The new statements:

```
&print_switch SWITCHNAME
&print_switch &revert {&all}
```

affect which output switch the &print and &print_nnl statements print on. The "&print_switch SWITCHNAME" usage directs the output of all subsequent &print and &print_nnl statements (only in the current exec_com and independently of other types of output) to the specified single SWITCHNAME until the corresponding "&print_switch &revert". An example is using &print to print on the terminal while command output is directed to a file. The "&print_switch &revert &all" usage pops all previous &print_switch statements and redirects &print and &print_nnl output to user_output.

5. Command escape

The new &execute statement with short name &exec is used to execute a command line at any point within an exec_com. For example, it allows the user to invoke file_output, terminal_output, syn_output, and revert_output commands while inside input loops, and makes it unnecessary to duplicate the functions of these commands within exec_com.

Also, &exec statements are traced by "&trace control" independently of "&trace &command", and do not print ready messages regardless of the state of &ready or &ready_proc.

Summary list of new keywords:

&begin	&on
&condition_name	&output
&continue_to_signal	&print_switch
&discard	&restart
&do	&revert
&end	&signal
&execute, &exec	&variable, &var