To: Distribution

From: Steve Herbst

Subject: Proposed exec_com features

Date: 1/19/77

1. Syntax Checking and && Escape Feature

Exec_com parameters and control statements all begin with the character &. Any string beginning with & that is not currently defined as a parameter or control statement represents itself. For example, the statement:

```
&print &t3 &5
```

prints:

```
&t3 <value of 5th argument>
```

Exec_com's toleration of undefined &strings becomes a problem when we want to define new &strings, for example if we want to make &tn stand for the first through nth arguments. Any such new feature requires an incompatible change.

This problem would not exist, currently, if undefined &strings were all rejected as syntax errors. The installation of an exec_com that diagnoses syntax would be the last incompatible change for a long time. The &print line above would cause this new version of exec_com to call sub_err_:

```
exec_com: Syntax error on line n.
          Undefined control string &t3
```

and return.

The syntax-checking version of exec_com must provide an escape feature for writing arbitrary &strings. The pair && represents the single character &. The &print line above must be converted into:

```
&print &&t3 &5
```

The && escape feature allows the parameter &(n) to take on the same meaning in exec_com's as in the do command and still let the user to invoke do from inside an ec. Currently, this is done as follows:

```
do "copy &(1)>&(2) &(3)" [wd] &f1
```

relying on the fact that exec_com does not replace the &(n)'s. When exec_com is changed to understand &(n)'s and &&'s, the same effect can be achieved by:

```
do "copy &&1>&&2 &&3" [wd] &f1
```

The incompatible change toward syntax checking should be optional per exec_com segment. That is, only ec's that have been explicitly converted should run with syntax checking and the new interpretation of && and &(n). The new tool convert_ec replaces &'s in undefined strings with &&'s. For example, it replaces &(n) with &&(n). It also inserts the control line "&syntax on" at the beginning of the converted ec segment. Eventually, the new version will be assumed for all ec's and &syntax lines will be ignored.

Once this incompatible change has been made, other &features can be added compatibly.

2. &control_line or

Today there is no way to make exec_com print control statements. The user would often like to see, for example, the expression tested by &if to determine what happens next. The control statements:

```
&control_line on
&control_line off
```

turn on and off a mode that prints all control lines after expanding them and before executing them.

3. &set and &value

The current method of defining variables is inadequate. The value,set command and the value active function keep a data base in the user's home directory. Variables are static from process to process and from exec_com to exec_com. With automatic exec_com variables, one ec would not have to worry about the variables used in another.

The &set control statement can appear anywhere that commands or control statements such as &print can appear. The control line:

```
&if [query continue?] &then &set(go,once more)
```

conditionally assigns the value "once more" to the variable go. Any previous value of go is replaced.

The string &value can appear anywhere, for example:

```
&goto &value(foo)
```

Its argument is enclosed in parentheses to allow concatenation of the value:

```
create a.&value(date).log
```

As a further useful extension, the string:

```
&value[active function]
```

is replaced by the active function return value. For example:

```
&set qpath &value[wd]>q.ms
```

4. &default

An exec_com cannot anticipate how many arguments it will be called with. The &default statement assigns default values for argument positions, in place of the current default "". These defaults are used in evaluating the parameters &n, &qn, &rn and &fn. Usage is:

```
&default default1 ... defaultj
```

For example, a start_up.ec designed to be called in a variety of ways can use the statement:

```
&default login interactive
```

and at some point say:

```
&if [equal &1 login] &then &goto set_io
```

Called with no arguments, start_up.ec assumes a new interactive process. After new_proc, the answering service says:

```
ec start_up new_proc interactive
```

giving &1 the value "new_proc".

The &default statement makes the default active function unnecessary in ec's. The default active function was designed to be used inside exec_com's and do command lines, for example:

```
dprint -queue [default 2 &2] &1
```

in an exec_com whose optional second argument is a queue number. If there is no second argument, the queue number is 2. The above statement can be replaced by:

```
&default "" 2
:
dprint -queue &2 &1
```

5. &return(value)

This control statement allows an exec_com to be called as an active function. It causes the exec_com command to fill in the specified value as its active function return value.

If exec_com is called as a command and executes a &return statement, it prints the value and returns. In this case, &return(value) is equivalent to the sequence:

```
&print value
&quit
```

If exec_com is called as an active function and executes a &quit statement, the null value "" is returned.

6. Extensions to the &(n) Parameter

When incompatible change number 1 has been made, the parameter &(n) will represent the nth argument to exec_com. One extension to the use of this parameter is to allow an expression inside the parentheses, for example:

```
        &(&value(foo))
or:     &(&value[active function])
```

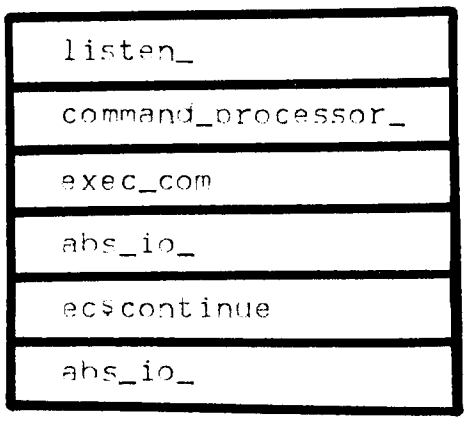
as long as the expression yields a legal parameter string, for example "f3".

7. &on condition_name

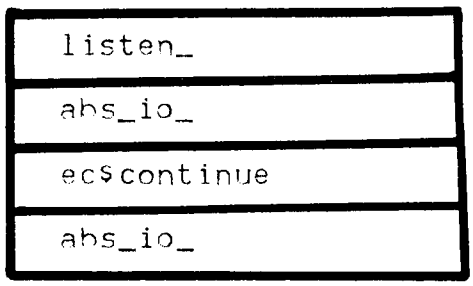
This control statement is followed by a command line, another control statement, or a block of statements surrounded by &do and &end described below. Its effect is to set up a handler for the specified condition.

The utility of this feature is clear, but its method of implementation is open to discussion. The module abs_io_, called to interpret each line of an ec, returns to the exec_com command when it is done with the line. Any handler set up by abs_io_ is therefore reverted automatically after the &on line.

One solution to this problem is for abs_io_, after setting up a handler, to call exec_com instead of returning. It calls exec_com$continue, which in turn starts calling abs_io_ beginning with the next line. In an interactive process, the stack looks like this:



or in absentee:



and the remainder of the ec is run with &on's handler still in effect.

The corresponding statement:

```
&revert condition_name
```

causes the handler to be reverted where it was initially set up. It does not cause ec$continue to return, since the order of &revert's is unpredictable.

8. Comments Inside Lines

Exec_com lines beginning with & and one or more blanks are comments. There is no particular reason why comments have to begin lines. Exec_com should be changed to allow:

```
&print this stinks      & complain about j<0
cr [date].[time]        & create a temporary
```

Comments alongside the text can improve readability.

9. &do and &end

These control statements allow better-structured exec_com's by replacing a lot of &label's and &goto's. The string &do can

by replacing a lot of &label's and &goto's. The string &do can appear anywhere that commands and control statements such as &print can appear. When &do is encountered, the corresponding &end is found by counting &do's and &end's in the text. This search for &end happens after parameter substitution and before the remainder of the line is executed.

10. Multiple Nesting

For even better structure in ec's, we can nest &if's, &then's, &else's, &do's and &end's to an arbitrary depth. Example:

```
&if [...] &then &if [...] &then ...
     &else ...
&else &if [...] &then &do ... &end &else
     ...
```

11. Line Numbering

Exec_com is good for generating command lines to test commands and the command processor. The printed output of two exec_com runs can be compared by the compare_ascii command. Comparison is especially useful if lines are numbered. By putting the input line number on each output line, we can make it easy to refer to a numbered listing of the exec_com being run.

A comparison test where the same exec_com is run with specifically different versions of programs is a good way to automatically find out whether the programs' operation has changed. Commands that print can be used to show the results of programs that do not print. The test exec_com can be modified and extended as new features are added and new bugs discovered and fixed.

The new command ecln is the exec_com command with line numbering turned on. Output from each run of the test exec_com is directed to a temporary file. An exec_com can be used to run the comparison test:

```
&label compare_output
&
&  &1 - pathname of test ec
&  even-numbered args - pathnames of old versions
&  odd-numbered args - pathnames of new versions
&
&if [exists segment [pd]>temp1] &then tc [pd]>temp1
&if [exists segment [pd]>temp2] &then tc [pd]>temp2
in (&2 &4 &6 &8 &10 &12 &14 &16 &18 &20)
fo [pd]>temp1 -osw user_output -osw error_output
ecln &1
co -osw user_output -osw error_output
tm (&2 &4 &6 &8 &10 &12 &14 &16 &18 &20)
```

```
in (&3 &5 &7 &9 &11 &13 &15 &17 &19 &21)
fo [pd]>temp2 -osw user_output -osw error_output
ecln &1
co -osw user_output -osw error_output
tm (&3 &5 &7 &9 &11 &13 &15 &17 &19 &21)
cpa [pd]>temp1 [pd]>temp2
```

The -osw (-output_switch) control argument to file_output and console_output is a new feature that allows the user to specify which I/O switches are redirected. In the above exec_com, both user_output and error_output are directed to the temporary files.

The ecln command implements numbering of output lines by calling abs_io_$control with the new order "line_numbers" and by splicing the new I/O Module abs_io_ln_ after user_output and error_output.

When abs_io_ intercepts an input line, it increments an internal static counter named input_line. The I/O Module abs_io_ln_ prefixes every output line that it intercepts with the value of input_line.

A sample follows:

```
38   &  Finally, test -rb on queues.
39   la 1.ms -rb
39o  4, 4, 4
39o  rw   *.SysDaemon.*
40   la ?.ms
40o    >udd>m>Herbst>test_dir>1.ms
40o  rw   *.SysDaemon.*
40o
40o    >udd>m>Herbst>test_dir>2.ms
40o  rew  Orange.Juice.*
40o  rw   *.SysDaemon.*
41   &quit
```

I have been using test exec_com's personally for several months to test commands and have found that they save me a lot of time and uncertainty.