To: Distribution

From: Steve Herbst

Subject: Version 2 exec_com

Date: 09/13/78

Various new exec_com features have been proposed over the past two years and discussed at design review meetings. Three documents have appeared:

01/19/77	MTB-324	Proposed exec_com features
02/29/78	MTB-362	Variables in exec_com
03/29/78	MTR-147	Variables in exec_com

This MTB presents a consistent set of exec_com modifications intended for Release 7.5. It includes the most important and least controversial of the features suggested thus far.

The most important and widely requested feature is automatic variables. Users need a way to assign variable values that are independent of the context in which an exec_com is used. Two constructs, &set and &value, manage automatic variables with names of arbitrary length and values of arbitrary length. Values are substituted into each line before the line is executed or passed on to the reader of input.

Another important feature is the &return statement, which returns a value when the exec_com command is called as an active function. This construct enables users to write active function macros the way they currently write command macros.

Other features are the optional tracing of control lines and comment lines, a line continuation feature, a set of literals for explicitly inserting carriage motion, and the &(n) and &! features currently accepted by the do command and active function (these are the only do parameters that exec_com does not currently recognize).

Other potentially useful features, such as nested &if's, &then-&else's, and &do-&end's are omitted because of the design problems that they raise.

A NEW VERSION

Adding any new feature to the exec_com language necessitates a version change. This is unfortunately true since the installed abs_io_ passes on &constructs that it does not recognize, without complaining of a syntax error. Users can therefore have random &strings in their ec's. An example is the intentional use of &(n), currently not an exec_com feature, to pass argument parameters to the do command. Adding &(n) to exec_com would be an incompatible change since it would cause the values of exec_com arguments to be passed to do instead of "&(n)".

Similarly, any new &string in exec_com is an incompatible change since that string might already appear in ec's.

Freezing the exec_com language is a poor solution to the compatibility problem. Instead, a new version will be installed and optionally available. If and only if the first non-comment line of an exec_com is the string "&version 2", the new entry point abs_io_v2_get_line is attached in place of abs_io_get_line. This new entry point can implement arbitrary syntax, though it should support all of the existing features so that existing ec's can be converted.

Version 2 exec_com uses double ampersand (&&) as an escape feature for writing literal ampersand characters. Except for literal ampersands, & is always a syntactic operator. Any &string that is not recognized as part of the current syntax of the language is a syntax error and interrupts execution of the exec_com. Any new &construct added to the Version 2 language will be an upwards compatible change.

A convert_ec command will be installed at the same time as abs_io_v2_get_line. Since all of the valid constructs in exec_com will have the same meaning in the new version, convert_ec leaves these intact. Any other &'s that it encounters it replaces with &&'s. Finally, it prefixes the file with "&version 2".

If necessary, convert_ec can also convert Version 2 back to Version 1.

Assuming wide acceptance, Version 2 will one day become the standard version and "&version 1" will be required at the start of ec's in order to attach the old abs_io_get_line.

VARIABLES

Automatic variables are described in MTB-362 and MTR-147. Three changes are planned to the design that appears in those documents:

1. The original &af_value[] construct replaces the more general &cp_value(). The latter is too problematic to be handled in the first release since an arbitrary command language string can use iteration to expand into several values. It is not clear what iteration means inside an exec_com. The &af_value[] construct, with surrounding brackets required, can only expand into a single value.

2. Substitution of variables, active strings and argument parameters should not be allowed to alter the syntax of exec_com control lines. In particular, &'s and white space in expanded values should not affect exec_com syntax.

   For example, the line:

   ```
   &if &1 &then &2
   ```

   should have the same meaning whether or not exec_com's first argument contains the string " &then ". If this restriction is not observed, exec_com's are only understandable at run-time.

   MTB-362's proposal for variable substitution is consistent with exec_com's current practice of two-pass parsing. The first pass expands all substitutable constructs, and the second pass parses the line (if an exec_com control line) and executes it. The installed exec_com is prone to dynamic modification of syntax. No one in my knowledge has ever used this misfeature to advantage. On the contrary, it is a potential hazard.

   Version 2 has a three-stage parse. Control line keywords are recognized first. Then portions of the line between the keywords are expanded. Finally, the line is executed. (Arguments to &then and &else are only expanded if they are to be executed.) The syntactic function of a portion of the line is determined before that portion is expanded.

   A way to change syntax dynamically could someday prove useful. If so, it should be available by explicit request rather than by default. For example, a &rescan() construct can cause its argument to be expanded first and then scanned for keywords.

3. The &defined() predicate, which returns true if its argument is the name of a variable with an assigned value, is having its name changed to &test_value. Other useful predicates, for example &test_af which returns true if exec_com was called as an active function, use the same naming convention.

Variables are per-invocation of exec_com. Their names are hashed in a small table. Argument values, on the other hand, are stored by the exec_com command and cannot be changed.

The &default statement is used to assign values for exec_com arguments where no arguments are specified. For example, the line:

```
&default PCO tape -debug
```

causes the value of parameter &1 to be PCO if no first argument was specified to the ec command, and so on. The string &test_value(1) still evaluates to false if there is no first argument. The string &value(1) evaluates to "PCO", same as &1.

If the variable foo has the value "one two" containing a space, the statement:

```
&default &value(foo)
```

assigns the default value "one two" to the first argument rather than "one" to the first and "two" to the second. In general, space resulting from expansion is not the same as space in the original line.

The keyword &none is used in &default and &set statements to leave parameters and variables unbound. For example, the statement:

```
&default PCO &none tape_04
```

sets defaults for arguments one and three but none for argument two. The statement:

```
&set foo &none
```

leaves the variable foo unbound, even if it previously had a value.

If there is no nth argument to exec_com and no default is specified, the string &n evaluates to null string whereas the string &value(n) causes an error.

LABEL SEARCHES

Since control words cannot result from expansion in Version 2, lines beginning with &n's do not have to be expanded during label searching as is done currently. The search simply looks for <NL>&label.

The label name, however, can be the result of expansion. If &label is followed by an expandable string such as &value(label_var), that string must be expanded as part of the search.

Label search begins at the start of the exec_com and looks for the first occurrence of the label. This can be either a constant label or the result of expansion. Two lists are kept, one of constant labels with their positions, and one of expandable labels and their positions. The label search first searches the list of constant labels. If one is found, all expandable labels before the constant label's position are expanded to see if one of them intercepts control. If no constant label is found in the constants list, expandable labels in the expandables list are expanded and then previously unreached portions of the exec_com are searched for &label statements.

Labels are only stored in the tables when they are encountered in searches. Any portion of the exec_com segment has to be searched for labels only once.

EXPLICIT EXPANSION

Expansion is defined as the replacement of substitutable constructs by their values.

The values of substitutable constructs such as &value() are purposely not rescanned for substitutables. Therefore, any indirecting through variables must be asked for explicitly. An example is:

```
&set one two
&set two three

&value(&value(one))  ->
     &value(two)  ->
     three
```

Since &value(), &af_value[] and parameters rely on particular interpretations of their argument (&value() requires a defined variable name, &af_value[] requires an active string, and parameters require a number), nested expressions involving these constructs rely on the values of inner constructs to be of

certain types. For example, &(&value(foo)) relies on the value of the variable foo to be a number, and &value(&1) relies on the first exec_com argument to be the name of a bound variable. Therefore, the statement that evaluates an indirect expression must cooperate with the statements that set the values.

Indirecting is most useful if the assignment statements alone can specify whether indirecting is to take place through variables, active strings or parameters. The value of a variable can then be set to &value(), &af_value[] or &n and its value's value substituted in later, independent of which construct is used.

For this purpose exec_com requires a new construct, &expand(), that re-expands the value of its argument. Like &value(), &af_value[] and the various parameters, it first expands its argument. Then, instead of looking up the expansion in a table, it re-expands any substitutables that are there. The expansion of a string not containing any substitutables is defined to be the string itself. A summary of the substitutable constructs shows &expand's relation to the others:

&value(S)	expands S, then looks up the expansion in the table of variables.
&(S)	expands S, then takes the expansion as the number of an exec_com argument.
&af_value[S]	expands S, then evaluates its expansion as an active string.
&expand(S)	expands S, then re-expands its expansion.

Suppose that the variable foo is set to "&1" in one place, to "bar" at another place, to "&value(bar)" at another, and to "&af_value[plus &value(bar) 1]" at another. The expression &expand(&value(foo)) does whatever kind of indirecting is implied by the value of foo. In the first case it evaluates to the first ec argument, in the second case to "bar", in the third to bar's value, and in the fourth to bar's value plus one.

QUOTING AND REQUOTING

Since arbitrary strings can result from expansion, two more constructs are useful for quote-doubling and requoting the expanded values. These are &quote() and &requote(), each of which expands its argument and then adds quotes in the same way that the &q or &r parameter adds them to exec_com arguments.

A few examples are necessary:

```
(first arg is a"b c)
&set foo a"b c

&q1  ->  a"b c
&r1  ->  "a""b c"

&quote(&value(foo))  ->  a"b c
&requote(&value(foo))  ->  "a""b c"

"&q1"  ->  "a""b c"
"&r1"  ->  """a""""b c"""

"&quote(&value(foo))"  ->  "a""b c"
"&requote(&value(foo))"  ->  """a""""b c"""
```

LITERAL CHARACTER ESCAPES

White space and ampersands have special meaning in the exec_com language. The && feature allows a single ampersand character to be inserted without affecting the syntax of a line.

Eight new keywords are proposed for the insertion of arbitrary numbers of literal characters. Their primary function is to make exec_com's more readable by distinguishing parsable characters from non-parsable ones and to make counting of multiple characters unnecessary (Ec's that write ec's can require several layers of &'s). The new keywords are:

&SP(n)	n literal spaces
&HT(n)	n literal horizontal tabs
&VT(n)	n literal vertical tabs
&NL(n)	n literal newline characters
&FF(n)	n literal form-feed characters
&BS(n)	n literal backspace characters
&QT(n)	n literal double quotes
&AMP(n)	n literal ampersands

If the repetition factor n is omitted, 1 is assumed. For example, &SP() is replaced by a single space. The string &SP without parentheses causes a syntax error.

Note that exec_com does not process quotes, as does the command processor. If it did, the interaction between exec_com quote processing and command language quote processing would be very confusing.

DO PARAMETERS

The two parameters &(n) and &! recognized by the do command and active function should also be recognized by Version 2 exec_com. The parameter substitution performed by exec_com and do are in all other respects identical and should remain that way since they are used identically to insert command-line arguments.

The first, &(n), is equivalent to &n for 0<=n<=9 but also allows n to be 10 or greater. The second, &!, expands to a unique_chars_ string of 15 characters the first time it appears in an ec, and thereafter to the same 15-character string. It is useful for naming temporary entities unique to a particular invocation of exec_com.

The parameter &nm where n and m are digits (for example, &10), currently accepted by exec_com to mean the nm'th argument, is not recognized that way by the do command or active function. This string in a do command line expands into the value of the nth argument concatenated with m. For total compatibility with do, exec_com should interpret &nm the same way. The convert_ec command should map existing occurrences of &nm into &(nm).

CONTINUATION

The only way currently to put a long line into an exec_com is to run it off the end of the printing line. The lack of a continuation sequence is annoying when typing and makes ec's less readable than they could be. Since there is often no substitute for a long command line, a continuation feature has been added to Version 2.

The character sequence &+ at the beginning of a line identifies that line as a continuation of the preceding line. The abs_io_v2_get_line program has to look ahead when it is processing a line to see whether the next line begins with &+. White space at the end of the line being continued is flushed, therefore if necessary it must be included after the &+ on the continuation line.

The &+ character sequence is not a normal keyword in that it is not delimited on the right by white space or left parenthesis. Instead, it is followed immediately by the first character of the continuation.

The &+ sequence is also allowed at the end of a line to make it clear to people that the line is being continued. In this case, the next non-comment line must begin with &+ or else a syntax error occurs.

COMMENTS

A comment in the existing version of exec_com requires a line to itself. Version 2 allows them anywhere in the line, identified by &<white space>. The remainder of the line is ignored as a comment. Comments can therefore be used to explain the line-by-line operation of an ec.

White space before comments is flushed, allowing comments to be adjusted to a particular column position. The &+ sequence at the beginning of a line continues the portion of the previous line that precedes any comment.

WHITE SPACE

White space is arbitrary combinations of the four characters SP, HT, VT and FF. This definition is the same as the one used by abbrev and the command processor. White space is flushed before comments and after the keywords &then and &else. Control statements use white space to delimit their arguments, but otherwise ignore it. White space is stripped before expansion and can be added by means of &SP, &HT, &VT and &FF.

TRACING CONTROL

Two new statements are proposed for tracing control lines and comments. These are "&control_line on" ("&control_line off") and "&comment_line on" ("&comment_line off"). Their operation is analogous to "&command_line on" and "&input_line on". They complete a useful set of tracing modes.

ACTIVE FUNCTION USAGE

In order to write ec's that return values, as in:

```
sm [ec last_installer pl1_abs] I'm changing...
```

it is necessary to have an exec_com statement that returns text. The new &return statement is equivalent to &quit but does one of two things. If the exec_com command was called as an active function, &return returns the remainder of the (expanded) line as the active function return value. Any white space in the line is included in the return value. If exec_com was called as a command, &return prints the remainder of the line and quits.

The following is a summary list of new Version 2 features:

EXECUTABLE STATEMENTS

```
&control_line on/off
&comment_line on/off

&set            (for variables)
&default        (for missing exec_com arguments)

&return         (for exec_com active function)
```

SUBSTITUTABLE EXPRESSIONS

```
&value(...)     value of a variable
&af_value[...] expansion of an active string
&expand(...)    re-expansion of substitutables

&quote(...)
&requote(...)

&none           null binding for &set and &default

&test_value(...)          TRUE if variable defined
&test_af()                TRUE for ec active function

&!              unique string, as in do command
&(n)            argument n, as in do command

&SP(n)          n literal spaces
&HT(n)          n literal horizontal tabs
&VT(n)          n literal vertical tabs
&NL(n)          n literal newline chars
&FF(n)          n literal form-feed chars
&BS(n)          n literal backspace chars
&QT(n)          n literal quotes
&AMP(n)         n literal ampersands
```

OTHERS

```
&+              start a continuation line
&<SP>           comments anywhere in line
```