To: Distribution

From: Steve Herbst

Subject: Variables in exec_com

Date: 2/29/78

This MTB proposes a variable substitution feature to be added to exec_com. Hopefully, the &value syntax described here is unambiguous and will not interact adversely with other features of the language.

Each exec_com line is processed in two distinct stages:

1. Variable, parameter and active function substitution.

2. Execution of control lines and delivery of other lines to the input switch.

Only the first stage is concerned with variables.

A variable name is defined and assigned a value by the &set control line:

```
&set variable_name value_string
```

The string variable_name cannot contain ampersands, parentheses, brackets or white space. The variable keeps its value until the exec_com returns or until the value is explicitly changed by another &set statement. Variable names and their values are stored in a per-stack frame data base and are local to an invocation of exec_com. If an exec_com A sets the value of a variable, that value is not known to any exec_com that A calls or to any exec_com that called A, including other invocations of A.

Reference is of the form:

```
&value(variable_name)
```

and can appear anywhere inside exec_com lines. Reference to an unset variable is a semantic error and aborts the exec_com.

Active functions can be evaluated in the substitution stage by saying:

```
&af_value[active_function args]
```

The substitutable constructs in exec_com are &value strings, parameters, and the && escape sequence. Examples:

1. &value(foo)
2. &af_value[plus [divide 6 4] 2]
3. &1, &q1, &r1, &f1, &n
4. &&f2

The last evaluates to the literal string &f2 and no parameter substitution is performed.

All of these constructs have equal priority. Substitutables are expanded from left to right. Substitution is also recursive and iterative.

1. Recursion: When substitutables are nested, the innermost one is expanded first.

```
(where args = 2, a, b, c, d)
&set arg_index_3 2

&r&value(arg_index_&af_value[plus &1 1]) ->
&r&value(arg_index_&af_value[plus 2 1]) ->
&r&value(arg_index_3) ->
&r2 -> "a"

&f&value(arg_index_3) ->
&f2 -> a b c d

&&value(arg_index_3) ->
&value(arg_index_3)
                (not expanded)
```

2. Iteration: After substitution, the string is re-scanned. Expansion continues until there are no substitutables.

```
&set one &&value(two)
&set two MIT

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

3. Both:

```
(args = tape, map, debug)
&set one &&1
&set tape 50207

&value(&value(one)) ->
&value(&1) ->
&value(tape) -> 50207
```

It is an error to have a looping definition, and users have to be warned in the documentation. Two examples of looping definitions are:

```
&set one &&value(one)
```

and:

```
&set one &&value(two)
&set two &&value(one)
```