IBM VM/System Product: EXEC 2 Language Reference Summary

SX24-5124-0

First Edition (July 1980)

This card applies to the initial release of IBM Virtual Machine/System Product, Program Number 5664-167 and to all subsequent versions and modifications unless otherwise indicated in new editions. However, the basic document is the authoritative source and will be the first to reflect changes. Information herein is extracted from *IBM Virtual Machine/System Product: EXEC 2 Reference*, SC24-5219.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below; requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

This publication has been produced by IBM Corporation, Programming Publications, Dept. G60, P.O. Box 6, Endicott, New York. U.S.A. 13760.

Control Statements

Control statements begin with a control word, which is usually followed by one or more additional words. The control words and the rules for their use follow:

```
&ARGS          [word1 [word2  . . . ]]
```

Assign "word1", "word2", . . . , to the arguments &1, &2, . . . , and discard any other EXEC 2 arguments that were previously set. The number of arguments now set is the number of words given in the &ARGS statement, which may be less or greater than the number of arguments previously set.

```
&BEGPRINT      [ n        [ k ] ]
&BEGTYPE       [ *        [ * ] ]
               [ label          ]
               [ 1              ]
line1
line2
. . .
```

Print at the console "line1", "line2", . . . , truncated if necessary at column "k", without removing surplus blanks or replacing any EXEC 2 variables. If the truncation column is not given, or is given as "*", the lines are not truncated by the EXEC 2 interpreter. (CMS truncates at 130 characters.)

The number of lines to be printed is determined by the first argument, as follows:

n, 1
: Print the given number of lines; or, if there are insufficient lines in the file, print all lines to the end of the file.

*
: Print all lines to the end of the file.

label
: Print down to, but not including, a line that contains the given label and nothing else; or, if such a line does not exist, print all lines to the end of the file. The label, to be recognized, must be wholly contained within the columns that would otherwise be printed, and it must be the only word within these columns. The first character of a label must be a hyphen.

After the lines have been printed, execution continues on the line following the last one printed, or, if printing is terminated by a label, on the line following the label.

This and "&BEGSTACK" are the only statements that occupy more than one line, and are the only statements that permit the lines of an EXEC 2 file to be handled literally, that is without removing surplus blanks or replacing EXEC 2 variables.

```
&BEGSTACK      [ n        [ k    [ FIFO ] ] ]
               [ *        [ *    [ LIFO ] ] ]
               [ label                      ]
               [ 1                          ]
line1
line2
. . .
```

Place in the console stack "line1", "line2", . . . , truncated if necessary at column "k", without removing surplus blanks or replacing any EXEC 2 variables. If the truncation column is not given, or is given as "*", the lines are not truncated. The lines are by default stacked FIFO (first in, first out), but this can be changed by giving "LIFO" (last in, first out) as the third argument.

The number of lines to be stacked is determined by the first argument, as follows:

n, 1
: Stack the given number of lines; or, if there are insufficient lines in the file, stack all lines to the end of the file.

*
: Stack all lines to the end of the file.

label
: Stack down to, but not including, a line that contains the given label and nothing else; or, if such a line does not exist, stack all lines to the end of the file. The label, to be recognized, must be wholly contained within the columns which would otherwise be stacked, and it must be the only word within these columns. The first character of a label must be a hyphen.

After the lines have been stacked, execution continues on the line following the last one stacked, or, if stacking is terminated by a label, on the line following the label.

This and "&BEGPRINT" are the only statements that occupy more than one line, and are the only statements that permit the lines of an EXEC 2 file to be handled literally, that is, without removing surplus blanks or replacing EXEC 2 variables.

```
&BUFFER        n        [comment]
               *
```

Discard the lookaside buffer (if any) together with its contents; and then, if "n" is given, and is positive, or if "*" is given, create a new lookaside buffer. If "n" is given, and is zero, a new lookaside buffer is not created. The value of "n" must not be negative. (In CMS the initial buffer size is 32 lines.)

The lookaside buffer is a contrivance that enables the EXEC 2 interpreter to remember the location of labels to which reference has already been made, and to keep a private copy of some of the more recently executed lines of the file. It can thereby improve the performance of EXEC 2 loops, in which the same labels and lines are used repeatedly.

If "n" is given, it defines the maximum number of lines that can be kept in the buffer; if "*" is given, there is no fixed limit. For maximum effect, the buffer should be capable of keeping the longest loop in its entirety and should be set up before entering the loop. An even larger buffer may be advantageous if user-defined functions or subroutines are invoked from within a loop.

A lookaside buffer should not be used if the EXEC 2 file is subject to modification during execution; if it is, the results are unpredictable.

```
&CALL          line-number [arg1 [arg2  . . . ]]
               label
```

Create a new generation of the EXEC 2 arguments &1, &2, . . . , initialized to "arg1", "arg2", . . . , and invoke the specified subroutine by transferring control to the given line, or to a line starting with the given label, in such a way as to allow control to be returned with the &RETURN statement.

The new generation of arguments supersedes the arguments that were previously set, making the previous values, and the number previously set, temporarily inaccessible. On entry to the subroutine, the values of the arguments, and the number set, are as given in the &CALL statement. Their values, and the number set, can be changed inside the subroutine in the same way as outside, such as by assignment or with the &ARGS or &READ statement.

On return, the new generation of arguments is discarded, making the previous values, and the number previously set, again accessible. Execution resumes on the line following the &CALL statement.

The first character of a label must be a hyphen. The search for a label starts on the line following the &CALL statement; then, if a match is not found before the end of the file, the search resumes at the top. If a matching label does not exist, execution stops abnormally with an error message.

```
&CASE          [ U [comment] ]
               [ M           ]
```

Translate to uppercase any lowercase alphabetic characters that are read in response to subsequent &READ statements, or do not translate them (allow "mixed" cases), or (if no argument is given) do not change the setting. Initially the translation is set to "U".

```
&COMMAND       word1 [word2  . . . ]
```

Issue to the host system (CMS) the command comprising "word1", "word2", . . . , separated from each other by a single blank. When it is finished, its return code is obtainable from the predefined EXEC 2 variable &RC. The statement normally has the same effect as:

```
word1  word2  . . .
```

There are, however, the following differences:

- A command, the first word of which begins with an asterisk, a hyphen, or an ampersand can be issued by giving it as the argument to &COMMAND; otherwise it is interpreted as a comment, a labeled statement, an assignment, or a control statement. (Note however, that these characters are not acceptable to CMS command mode.)
- &COMMAND overrides any presumption of a subcommand environment and always issues the command to the host system (CMS).

```
&DUMP          ARGS
               VAR[S]  [var1 [var2  . . . ]]
```

Print lines at the console of the form:

```
var = VALUE
```

either:
: one for each EXEC 2 argument &1, &2, . . . which is set;

or:
: one line for each of the variables "var1", "var2", ...

The lines are truncated if their length exceeds the implementation limit for printed output. (In CMS, the line is truncated if its length exceeds 130.)

```
&ERROR         action
```

Set the action which, until further notice, is to be invoked automatically on return from any commands (and subcommands) that yield an error return code (that is a return code that is not zero). The action may be any executable statement, including a null statement.

The action is not inspected at the time the &ERROR statement is executed. Instead, the search for and replacement of any EXEC 2 variables takes place each time the action is executed. The action is executed as if it occupied the same line in the EXEC 2 file as the command (or subcommand) that yielded the nonzero return code.

What happens after the action depends upon the type and consequences of the action. If it is itself a command (or subcommand) which also yields an error return code, execution stops abnormally with an error message; otherwise (unless the action causes a transfer of control), execution resumes at the line following the command that caused the action to be invoked.

Initially, the error action is set to the null statement.

```
&EXIT          [ return-code [comment] ]
               [ 0                     ]
```

Stop execution of the EXEC 2 file, and yield the given return code, which must be numeric. If the given return code is not within the range of return codes acceptable to the host system, the result is defined by the implementation. (In CMS the range is -2,147,483,648 to +2,147,483,647.)

```
&GOTO          line-number   [comment]
               label
```

Transfer control to the given line, or to the line starting with "label".

The first character of a label must be a hyphen. The search for a label starts on the line following the &GOTO statement; then, if a match has not been found before the end of the file, the search resumes at the top. If a matching label does not exist, execution stops abnormally with an error message.

```
&IF            word1      =|EQ [word2  executable-statement]
                          ¬=|NE
                          <|LT
                          <=|¬>|LE|NG
                          >|GT
                          >=|¬<|GE|NL
```

If the condition is satisfied, execute the given executable statement; otherwise proceed to the next statement. The comparative may be given in any of the forms shown (for example "=" or "EQ"). The comparison is numeric if both comparands are numeric; otherwise both comparatives are treated as character strings, and the shorter one is (for the purpose of the comparison) padded on the right with blanks. If "word2" is absent, a null string is used in its place.

```
&LOOP          n          m
               label      *
                          WHILE  condition
                          UNTIL
```

Loop through the following "n" lines, or down to (and including) the first line starting with "label", for "m" times, or indefinitely (*), or "WHILE" (or "UNTIL") the given condition is satisfied.

The values of "n" and "m" (if given) must be numeric; also "n" must be positive, and "m" must not be negative.

The first character of the label (if given) must be a hyphen, and the label must be attached, as the first word of the line, to an executable statement that lies below the &LOOP statement.

The form of the condition (if given) is similar to that of the &IF statement previously described:

```
word1       =|EQ            [word2 [comment]]
            ¬=|NE
            <|LT
            <=|¬>|LE|NG
            >|GT
            >=|¬<|GE|NL
```

The condition is evaluated before each iteration of the loop, including the first. If "word2" is absent, a null string is used in its place. The comparison is numeric if both comparands are numeric; otherwise, both comparands are treated as character strings, and the shorter one is (for the purpose of the comparison) padded on the right with blanks.

If the condition is invalid, execution stops abnormally with an error message that identifies the line containing the &LOOP statement.

```
&PRESUME       [ &COMMAND                 ]
               [ &SUBCOMMAND  environment ]
```

Presume that any executable statements that have the syntax of a command (that is the first word of the statement does not begin with an ampersand) are to be issued to the host system (CMS); or presume that they are to be issued to the given subcommand environment.

The name of the subcommand environment is not checked when the &PRESUME statement is executed. If, when a subcommand is subsequently issued, the environment does not exist, the only effect is to set a special return code. (In CMS it's -3.)

The "&PRESUME" control statement with no arguments is equivalent to "&PRESUME &COMMAND".

By convention, the presumption is initially set to "&COMMAND" if the EXEC 2 file has a filetype of EXEC; otherwise it is set to "&SUBCOMMAND filetype", where "filetype" is the filetype of the EXEC 2 file.

The presumption has no effect on &COMMAND or &SUBCOMMAND statements since these do not have the syntax of a command.

```
&PRINT         [word1 [word2  . . . ]]
&TYPE
```

Print at the console a line containing "word1", "word2", . . . , separated from each other by a single blank, or print a blank line if there are no words given. The line is truncated if necessary. (In CMS the line is truncated if its length exceeds 130.)

Unlike &BEGPRINT, surplus blanks are removed and the words to be printed are searched in the normal way for the names of EXEC 2 variables, which are replaced by their values.

```
&READ          [ n                          ]
               [ 1                          ]
               [ *                          ]
               [ ARGS                       ]
               [ STRING  var                ]
               [ VAR[S][var1 [var2 . . . ]] ]
```

Read from the stack (if the stack is not empty), or read from the console (otherwise), and execute or assign what is read according to the following rules:

n, 1, *
: Read "n" lines, or read an indefinite number of lines (*), and execute them individually as if they had been part of the EXEC 2 file. Reading stops (and normal execution resumes) when "n" lines have been read, or when a &BEGPRINT, &BEGSTACK, &EXIT, &GOTO, &LOOP, or &SKIP statement is encountered. Reading is suspended if a user-defined function or subroutine is invoked, and continues when control returns from that invocation.

  If a "&READ n" statement is read in response to a previous "&READ n" statement, the new value of n is added to the number of lines that remain from the previous statement. Reading stops if the number remaining becomes zero or less. The value of "n" may be negative.

  If a "&READ *" statement is read in response to a previous "&READ n or "&READ *" statement, or if a "&READ n" statement is read in response to a previous "&READ *" statement, an indefinite number of lines remain to be read.

ARGS
: Read a single line, assign the words in it to the EXEC 2 arguments &1, &2, . . . , and discard any other EXEC 2 arguments that were previously set. The number of arguments now set is the number of words in the line, which may be less or greater than the number of arguments previously set. (See the description of &ARGS. and the predefined variables &N and &1, &2, . . .)

STRING
: Read a single line and assign it, as a literal string, to "var", without removing any surplus blanks or replacing any EXEC 2 variables.

VARS
: Read a single line and assign the words in it to the variables "var1", "var2", ... If the number of words in the line read exceeds the number of variables given in the statement, the surplus words are discarded; or if the number of variables exceeds the number of words, the remaining variables are set to the null string. Therefore "&READ VARS" (without any variables) can be used to read a line and discard it. Asterisks may be used in lieu of variable names to indicate that the corresponding words in the line read are to be discarded.

In the case of &READ ARGS and &READ VARS . . . , the line that is read is scanned for words (leading, trailing, and other surplus blanks are discarded), but the words are treated as literals (there is no replacement of EXEC 2 variables).

The names of the variables in &READ VARS and &READ STRING are treated in the same way as on the left-hand side of an assignment statement. A variable of the form &j, where "j" is an unsigned integer without leading zeros, cannot be set with &READ VARS or &READ STRING if "j" exceeds the number of EXEC 2 arguments that are currently set.

Lines that are read are not truncated by the EXEC 2 interpreter; they are unaffected by the setting of &TRUNC.

(In CMS the maximum length of a line read from the console is 130, and the maximum length of a line read from the console stack is 255.)

```
&RETURN        [word]   [comment]
```

Return control to the most recent subroutine invocation (&CALL statement) to which return has not yet been made; or return " word" (or the null string) to the most recent user-defined function invocation to which a value has not yet been returned.

The generation of EXEC 2 arguments that was created at invocation is discarded; and the previous values, and the number previously set, become accessible again. The number of lines (if any) that remain to be read from the stack or console in response to a previous "&READ n" statement is reset to the number outstanding at the time of the invocation. Any loops that have been opened in the subroutine or function, and not closed, are aborted; and any loops that were open at the time of invocation are reinstated.

If there is both a subroutine invocation and a function invocation to which return has not yet been made, return is to the more recent point of invocation. If there is neither, execution stops abnormally with an error message.

```
&SKIP          [ n  [comment] ]
               [ 1            ]
```

If n > 0, skip the next "n" lines of the EXEC 2 file. If n < 0, transfer control to the line that is "-n" lines above the current line. If n = 0, transfer control to the next line.

If an attempt is made to transfer control to a line number that is zero or negative, execution stops abnormally with an error message. If control is transferred to a line below the last in the EXEC 2 file, execution stops normally with a return code of zero.

```
&STACK         [ FIFO [word1 [word2  . . . ]] ]
               [ LIFO                         ]
```

Place a line in the console stack containing "word1", "word2", . . . , separated from each other by a single blank, or stack a null line if there are no words. (In CMS, stacked lines are truncated at 255.) The line is by default stacked FIFO (first in, first out), but this can be changed by giving "LIFO" (last in, first out) as the first argument.

Unlike &BEGSTACK, surplus blanks are removed and the words to be stacked are searched in the normal way for the names of EXEC 2 variables, which are replaced by their values.

```
&SUBCOMMAND    environment [word1 [word2  . . . ]]
```

Issue to the given subcommand environment the subcommand comprising "word1", "word2", . . . , separated from each other by a single blank. When it is finished, its return code is obtainable from the predefined EXEC 2 variable &RC.

If the given environment does not exist, the only effect is to set a special return code. (In CMS it's -3.)

Normally, it is convenient to "presume" the environment so that this control statement does not have to be issued for every subcommand (see the description of &PRESUME). Note that the statement "&SUBCOMMAND environment" (without any additional arguments) is the only way of issuing a null subcommand.

```
&TRACE         [ ON [output-action] ]
               [ ERR                ]
               [ ALL                ]
               [ OFF                ]
               [ *                  ]
```

where "output-action", if given, is:

```
&PRINT   [word1 [word2 . . . ]]
```

or:

```
&COMMAND   word1 [word2 . . . ]
```

or:

```
&SUBCOMMAND   environment [word1 [word2 . . . ]]
```

Trace commands (and subcommands) that are issued from the EXEC 2 file; or trace commands (and subcommands) that yield an error return code (a return code that is not zero); or trace all executable statements; or do not trace any statements; or (if "*" is given) do not change the setting. The setting remains in effect until reset. The initial setting is OFF.

Trace information can be printed at the console, or passed to a command (or subcommand) for processing. The trace destination is determined by the output action, as described below.

ON
: When tracing is ON, each command is traced before it is executed, and subsequently the return code is traced if it is not zero. The return code is traced on a line by itself in the form "+++ E(nnn) +++".

ERR
: When ERR is in effect, commands that yield a nonzero return code are traced after execution, followed by the return code. The return code is traced on a line by itself in the form "+++ E(nnn) +++".

ALL
: When ALL is in effect, every executable statement, preceded by its line number, is traced before it is executed; nonzero return codes are traced (as for ON and ERR); and loop conditions and lines that are read from the console are also traced. The statement following an &IF clause, the action given in an &ERROR statement, and the conditional phrase in a &LOOP statement are traced as literal words (that is, without replacement of any variables). These statements and phrases are traced again, with the normal replacement of variables, at the time of their execution. A statement that is executed as a consequence of a satisfied &IF clause is preceded in the trace by an ellipsis. Words that exceed 24 characters in length are truncated in the trace at 21 characters and followed by an ellipsis. Statements that exceed 80 characters in length (with the line number and preceding ellipsis, if present) are truncated in the trace at an integral number of words and followed by an ellipsis.

OFF
: Do not trace any statements. This is the initial setting.

*
: Do not change the setting. "&TRACE" without arguments is equivalent to "&TRACE *".

output-action
: The output action gives the destination of the tracing information. The words in it are searched in the normal way for the names of EXEC 2 variables, which are replaced by their values, and the resulting sequence of words is set aside. When a trace line is produced, it is prefixed with the sequence of words, and the resulting EXEC 2 statement is executed without tracing. (See the descriptions of &PRINT, &COMMAND, and

&SUBCOMMAND). If the return code from the command or subcom mand is nonzero, execution stops abnormally with an error message.

Initially the output action is set to "&PRINT", which causes the trace to be printed at the console. If the output action is not given, the previous action remains in effect.

```
&TRUNC        [ k [comment] ]
              [ *           ]
```

Set the truncation column for EXEC 2 statements to "k", or set it to the maximum value, or (if no argument is given) do not change it. Initially, it is set to the maximum value. (In CMS the maximum value is 255.)

This setting affects only the reading of EXEC 2 statements from a file and the search for labels; it does not affect lines read from the console (which are not truncated), or lines appearing within a &BEGPRINT or &BEGSTACK statement (which are separately controlled); and it does not affect the length to which a statement can grow during or after replacement of EXEC 2 variables.

Changing the truncation column has the side-effect of purging the lookaside buffer (if there is one), and may consequently degrade performance if done within a loop.

```
&UPPER        ARGS
              VAR [S] [var1 [var2  . . . ]]
```

Translate to uppercase any lowercase alphabetic characters in the values of the EXEC 2 arguments &1, &2, . . . ; or translate to uppercase any lowercase alphabetic characters in the values of "var1", "var2", ...

A variable of the form &j, where "j" is an unsigned integer without leading zeros, cannot be translated with &UPPER VARS if "j" exceeds the number of EXEC 2 arguments that are currently set.

Predefined Variables

The following EXEC 2 variables are initialized or maintained automatically.

```
&
```

Initialized to its own name (the value "&").

```
&0
```

Initialized to the first word of the command string that is passed to the EXEC 2 interpreter. Normally, this variable has the same value as &FILENAME, but may be different if the EXEC 2 file was invoked via a synonym.

```
&1,&2, . . .
```

These are the EXEC 2 arguments. They are initialized to the arguments "arg1", "arg2", . . . , which are passed to the EXEC 2 file. They are reset by &ARGS or &READ ARGS; and they are temporarily reset by invocation of user-defined subroutines and functions. EXEC 2 arguments beyond the last that is set have an apparent null value, and cannot be set explicitly (for example, with an assignment statement).

```
&ARGSTRING
```

Initialized to the command string that is passed to the EXEC 2 file; treated as a single literal string starting with the character following the blank that terminates &0, and including any leading, embedded, or trailing blanks. The initial value includes the EXEC 2 arguments &1, &2, . . . , but &ARGSTRING is not affected by changes to them.

```
&BLANK
```

A word that has the value of a single blank.

```
&COMLINE
```

Initialized to zero, and maintained as the number of the line from which the last command (or subcommand) was issued from the EXEC 2 file.

```
&DATE
```

The true date on the primary meridian (Greenwich Mean Time (GMT)) in the form YY/MM/DD; evaluated when the statement containing it is executed.

```
&DEPTH
```

Maintained as the number of user-defined function and subroutine invocations to which return has not yet been made.

```
&FILEMODE
```

Initialized to the filemode (third qualifier) of the EXEC 2 file.

```
&FILENAME
```

Initialized to the filename (first qualifier) of the EXEC 2 file.

```
&FILETYPE
```

Initialized to the filetype (second qualifier) of the EXEC 2 file (for example, "EXEC").

```
&FROM
```

Initialized to zero, and maintained as the number of the line in the EXEC 2 file from which the last &GOTO statement was executed.

```
&LINE,&LINENUM
```

Maintained as the number of the current line in the EXEC 2 file.

```
&LINK
```

Maintained as the number of the line from which the currently executing user-defined function or subroutine was invoked, or has the value 0 if there are no user-defined functions or subroutines in execution.

```
&N,&INDEX
```

Maintained as the number of EXEC 2 arguments that are set. Initially this is the number of arguments that are passed to the EXEC 2 file. It is reset as a side effect of &ARGS and &READ ARGS, and it is temporarily reset by invocation of user-defined subroutines and functions.

```
&RC,&RETCODE
```

Initialized to zero, and maintained as the return code from the last command (or subcommand) issued from the EXEC 2 file.

```
&TIME
```

The true time-of-day on the primary meridian (Greenwich Mean Time (GMT)) in the form HH:MM:SS, evaluated when the statement containing it is executed.

Predefined Functions

A predefined function can be invoked only in the last term on the right-hand side of an assignment statement. The invocation takes the form:

    function-name OF [arg1 [arg2 . . .]]

The names of the predefined functions, and the rules for their use, are as follows.

```
&CONCATENATION OF    [word1 [word2  . . . ]]
&CONCAT
```

Concatenates "word1", "word2", . . . , into a single word, without intervening blanks; or yields the null string if there are no words.

```
&DATATYPE OF         [word]
&TYPE
```

Yields the value NUM if "word" represents a valid (signed or unsigned) number; otherwise, yields the value CHAR.

```
&DIVISION OF         dividend divisor
&DIV
```

Yields a numeric value which results from dividing the dividend by the divisor. Both the dividend and the divisor must be numeric and the divisor must not be zero.

If the dividend and divisor are both positive, or if they are both negative, the result is positive; if the dividend is positive and the divisor is negative, or vice versa, the result is negative; if the dividend is zero, then the result is zero.

In precise terms, the value is the integral part of the division of the absolute value of the dividend by the absolute value of the divisor, or minus this value if the dividend is not zero, and the sign of the dividend differs from that of the divisor.

```
&LEFT OF             word j
```

Yields a string of length "j" in which "word" is left-justified and either padded with blanks, or truncated on the right.

```
&LENGTH OF           [word]
```

Yields a numeric value representing the length of the word (that is; the number of characters in it); or yields zero if the word is absent.

```
&LITERAL OF          [string]
```

Yields the literal string that begins with the character following the blank that terminates "OF", and ends with the last nonblank character before or at the truncation column. Any leading or embedded blanks are retained, and the search for and replacement of any EXEC 2 variables that may appear in the string is suppressed.

```
&LOCATION OF         needle [haystack]
```

Searches "haystack" for the first occurrence of "needle", and yields a number indicating its starting position, or yields zero if there is no occurrence (or if the length of "needle" exceeds that of "haystack").

```
&MULTIPLICATION OF   i j [k  . . . ]
&MULT
```

Yields a numeric value representing the result of multiplying the given words, all of which must be numeric (signed or unsigned), and of which there must be at least two.

```
&PIECE OF            word   i [ j ]
&SUBSTR                       [ * ]
```

Extracts that piece of "word" that starts at character "i", with length "j"; or that starts at character "i" and runs to the end of the word (*).

The value of "i" (and "j" if given) must be numeric; also "i" must be positive, and "j" must not be negative.

If the value of "i" exceeds the length of the word, the value of the function is the null string. If "j" is given, but exceeds the remaining length of the word, the remaining length is used instead.

```
&POSITION OF         word [word1 [word2  . . . ]]
```

Compares "word" with "word1", "word2", . . . , looking for a match, and yields a numeric value representing the position of the first matching word, or yields zero if "word" does not match any of the other words (or if there are no other words given).

```
&RANGE OF            stem i j
```

Yields a string consisting of the words that are composed by appending to the given stem the numbers i, i+1, . . . , j, the words being separated from each other by a single blank; or yields the null string if i > j.

The stem is treated as a literal until after the composition is performed. The numbers that are appended to it are stripped of any plus sign or redundant leading zeros.

The composed names are searched for any EXEC 2 variables, which are replaced by their values in the usual way. If, as a result of this, a word is reduced to the null string, it is discarded from the result, and the next word is deemed immediately to follow the previous one.

```
&RIGHT OF            word j
```

Yields a string in which "word" is right-justified and either extended with blanks, or shortened on the left, to a length of "j".

```
&STRING OF           [string]
```

Yields the string that begins with the character following the blank that terminates "OF" and ends with the last nonblank character before, or at, the truncation column, suppressing the removal of any leading or embedded blanks in the string.

Each word in the string is searched in the usual way for the names of EXEC 2 variables, which are replaced by their values. However, blanks are not removed from the string, even if they are adjacent to a word that is reduced to the null string.

```
&TRANSLATION OF      word1 [word2 [word3]]
&TRANS
```

Makes a copy of "word1", modifies the characters in it as directed by "word2" and "word3", and yields the resulting string.

The rules for modification are as follows. Each character of the copy is considered in turn, and:

1. If "word2" does not contain a matching character, the character in the copy is left unchanged; or
2. If "word2" contains a matching character, in position "i" (or if it contains several matching characters, the first of which occupies position "i"), the character in the copy is replaced by the ith character of "word3", or by a blank if "word3" is not given or contains fewer than "i" characters.

The result has the same length as "word1".

```
&TRIM OF             [word]
```

Yields a string consisting of "word" with any trailing blanks removed, or yields the null string if "word" is not given.

```
&WORD OF             [word1 [word2 . . . ]] i
```

Yields the ith word from the given list of words, or yields the null string if "i" is zero or exceeds the number of words that are given. The value of "i" must be numeric, and must not be negative.

User-Defined Functions

A user-defined function can be invoked only in the last term on the right-hand side of an assignment statement. The invocation takes the form:

```
line-number  OF [arg1 [arg2  . . . ]]
label
```

The effect is to create a new generation of the EXEC 2 arguments &1, &2, . . . , initialized to "arg1", "arg2", . . . , and to invoke the given function; that is, to transfer control to the given line, or to a line starting with the given label, in such a way as to allow a value to be returned with the &RETURN statement.

The new generation of arguments supersedes the arguments that were previously set, making the previous values, and the number previously set, temporarily inaccessible. On entry to the body of the function, the values of the arguments, and the number set, are as given in the function invocation. Their values, and the number set, can be changed in the body of the function in the same way as outside, such as by assignment or with the &ARGS or &READ statement. On return, the new generation of arguments is discarded, and the previous values, and the number of arguments previously set, become accessible again.

The first character of a label must be a hyphen. The search for a label starts on the line following the function invocation; then, if a match is not found before the end of the file, the search resumes at the top. If a matching label does not exist, execution stops abnormally with an error message.

Types of Executable EXEC 2 Statements

- Null statement.

  A null statement is an executable statement in which the number of words is zero.
- Commands.

  An executable statement is deemed to be a command if it contains at least one word, and its first word does not start with an ampersand. It is issued immediately to the host system (CMS), or to a subcommand environment (for example, XEDIT). When it is finished, control returns to the EXEC 2 file, and its return code can be obtained from the predefined EXEC 2 variable &RC.
- Assignments.

  An executable statement is an assignment if the first word starts with an ampersand and the second word is an equal sign. The first word is taken as the name of an EXEC 2 variable, and assigned the value of the expression that follows the equal sign. The expression may be any of the following:
  - null
  - a single word, for example: ABC
  - an arithmetic expression, consisting of a sequence of words that represent positive or negative integers, separated by plus or minus signs, for example: 3 - 4 + -11 - 00
  - a function invocation, for example:

    &PIECE OF &1 2 1
  - an arithmetic expression (as above) in which the last term is replaced by a function invocation that yields a numeric value, for example:

    -1 + &LENGTH OF &1

  A variable of the form &j, where "j" is an unsigned integer without leading zeros, cannot be set with an assignment statement if "j" exceeds the number of EXEC 2 arguments that are currently set.

  The value of the variable on the left-hand side of the assignment statement is not modified until the expression on the right-hand side has been evaluated. If an assignment statement is syntactically invalid, or if evaluation of the expression results in numeric overflow, execution stops abnormally with an error message, without further evaluation.

  Control statements.

  An executable statement is a control statement if the first word is an EXEC 2 control word and the second word either is absent or is not an equal sign. Examples of control words are &GOTO, &EXIT, &IF, and &PRINT.

Rules for Interpreting Executable Statements

Executable statements are interpreted, one at a time, according to the following general rules.

1. The statement is scanned. This discards leading, trailing, and other surplus blanks, leaving a sequence of words separated from each other by a single blank.
2. The words forming the statement are searched for the names of any EXEC 2 variables, which are replaced by their values; except that if the variable is the target of an assignment, its name is retained. (A precise description is given in the section "EXEC 2 Name Substitution.") During this process, the words may grow or shrink in length.
3. If, as a result of step 1, a word is reduced to the null string, it is discarded from the statement so that the next word is deemed immediately to follow the previous one. With this exception, the words retain their identity; for example, if the value of a variable contains an embedded blank, the word containing it is still treated as one word, although when printed it might appear as two.
4. The statement is analyzed syntactically, and executed. Note that, except for identifying the targets of assignment, the syntax analysis is done after steps 1, 2, and 3 above.

EXEC 2 Name Substitution

The words that form an executable statement are searched for the names of EXEC 2 variables, which are replaced by their values. This is done according to the following steps:

1. Each word is inspected for ampersands, starting with the rightmost character of the word, and proceeding to the left.
2. If an ampersand is found, then it, with the rest of the word to the right, is taken as the name of an EXEC 2 variable, and replaced (in the word) by its value. This may increase or decrease the length of the word. Initially, all variables have a null value, except:
   a. the variables that represent the EXEC 2 control words and predefined functions, which are initialized to their own names (for example, the value of "&IF" is "&IF"); and
   b. the EXEC 2 arguments, and the other predefined variables, which have the values specified in the section "Predefined Variables."
3. Inspection resumes at the next character to the left, and the procedure is repeated from step 2 above, until the word is exhausted.

There is an exception if the word is the target of an assignment; in this case, inspection for ampersands stops on the second character of the word.

Note that any characters that are substituted are not themselves inspected for ampersands. They are, however, included in the name of the next variable if another ampersand is found to the left.

EXEC 2 in CMS

Since both CMS EXEC and EXEC 2 files are called in the same way, CMS examines the first statement of the EXEC file to determine which EXEC interpreter must handle it. If the first statement of the EXEC file is &TRACE, CMS calls the EXEC 2 interpreter to handle it. If the first statement is not &TRACE, CMS calls the CMS EXEC interpreter to handle it.

Some CMS limits that apply to EXEC 2 files are:

- EXEC 2 files used as CMS command files must have the &TRACE statement in the first record of the file. In subcommand environments, such as XEDIT for XEDIT macros, the &TRACE statement is optional.
- The maximum length of an EXEC 2 line is 255.
- The maximum length of a statement, after replacement of variables, is 511. (This limit is enforced only as needed by the interpreter; some statements can grow to a greater length.)
- The maximum length of a word, after replacement of variables, is 255.
- The maximum length of a line read from the console is 130, and from the console stack is 255.
- The maximum length of a printed line is 130.
- An EXEC 2 filename can be from one to eight characters long. The valid characters are A-Z, 0-9, and $, #, @. The filetype must be EXEC for files that are invoked from CMS command mode and XEDIT for files used as XEDIT macros.
- All EXEC 2 files have an initial lookaside buffer of 32 lines (see the &BUFFER description in the "Control Statements" section). The &BUFFER 0 statement must be issued to delete the lookaside buffer if the file is to be modified while being executed.
- In a context that requires numeric values, numbers must be in the range -2,147,483,648 to +2,147,483,647.
- In CMS, return codes for the &EXIT control statement are limited to the range -2,147,483,648 to +2,147,483,647. Attempts to exceed these limits will cause the EXEC 2 file to stop abnormally with an error message (NUMERIC OVERFLOW).
- CMS commands issued from EXEC 2 files are invoked in such a way that most information and error messages issued by the following CMS commands will not be typed: ERASE, LISTFILE, RENAME, STATE, and FILEDEF. This is also true for any other system or user command that makes a distinction in its operation based on flags passed in register 1. However, note that a nonzero return code from any of these commands will be reflected in the predefined variables &RETCODE and &RC.

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9
North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

SX24-5124-0 Printed in U.S.A.