IBM

REX Reference Summary

VM/370 • CMS

First Edition (November 1980) - for REX version 2.08

CONTENTS

REX is a command programming language which allows you to combine useful sequences of commands to create new commands. It is used in conjunction with, or as a replacement for, the CMS EXEC and EXEC 2 languages. REX is especially suitable for writing Execs or editor macros, but is also a useful tool for algorithm development.

REX EXEC

Invoke using: `REX [parameter]`

`REX ?`	describes REX EXEC and how to use the on-line help and tutorial.
`REX I`	installs REX in your CMS system under the name `EXEC` so that you may execute Execs written in any of the three languages. You probably will want to put '`EXEC REX I`' in your PROFILE EXEC.
`REX`	enters the on-line help, viewing an index of topics.
`REX 200nn`	(where `nn` is a REX error message number) describes the meaning of the error message and the likely cause of the error.
`REX kkk`	(where `kkk` is a REX keyword) gives immediate information on the specified topic.

Key to notation used on this card:

`GOTHIC` – indicates language keywords
italics – indicate defined syntactic units
`[ ]` – brackets indicate an optional item
`...` – ellipses mean multiple items are allowed
`{}` – braces specify list of alternatives (choose one)
`|` – separates alternatives in a list

DATA ITEMS

The REX language is designed for the easy manipulation of character strings. Its expressions and instructions manipulate the following *items*:

string — A *string* is a quoted string of characters. Use two quotes to obtain one quote inside the string. The string may be specified in hexadecimal if the final quote is followed by an `X`. Some valid *strings* are: `"Next"` `'Don''t touch'` `'1de8'x`

number — A *number* is a string of up to 9 decimal digits before and/or after an [optional] decimal point. It may have a leading sign. Some valid *numbers* are: `17` `98.07` `.101`

name — A *name* refers to a variable, which can be assigned any value. It may consist of up to 150 characters from the following selection: `A-Z, a-z, 0-9, @ # $ % ¢ _ . ? ! £` The first character may not be a digit or period, except if the *name* consists of the period alone. The *name* is translated to upper case before use, and forms the initial value of the variable. Some valid *names* are: `Fred` `COST?` `next_index` `A.j`

function-call — A *function-call* invokes an external routine with 0 to 7 arguments. The called routine will return a character string. A *function-call* has the format:
function-name`([`*expr*`] [,`*expr*`]...)`
Function-name must be adjacent to the left parenthesis, and may be a *name* or a *string*.

EXPRESSIONS

Most REX statements permit the use of expressions, following the style of PL/I. Expressions are evaluated from left to right, modified by the priority of the operators (as ordered below). Parentheses may be used to change the order of evaluation.
All operations (except prefix operations) act on two *items*, and result in a character string.

`Prefix + - ¬`	Prefix operations: Plus; Minus; and Not. (For + and -, *item* must evaluate to a *number*, for ¬, it must be '1' or '0'.)
`* / //`	Multiply; Divide; Divide and return the remainder. (Both *items* must evaluate to *numbers*.)
`+ -`	Add; Subtract. (Both *items* must evaluate to *numbers*.)
`(blank) ||`	Concatenate: with or without blank. Abuttal of *items* causes direct concatenation.
`= == ¬= >= <= ¬> ¬< > <`	Comparisons (arithmetic compare if both *items* evaluate to a *number*). The == operator checks for an **exact** match.
`&`	Logical And. (Both *items* must be '0' or '1'.)
`| &&`	Logical Or; logical Exclusive Or. (Both *items* must be '0' or '1'.)

The results of **arithmetic** operations are expressed to the same decimal precision as the more precise of the two *items*. For example, `123.57 + 12` will result in `135.57`. Results of division are rounded rather than truncated.

STATEMENTS

REX statements are built out of clauses consisting of a series of *items*, operators, etc. The semicolon at the end of each clause is often not required, being implied by line-ends and after the keywords `THEN`, `ELSE`, or `OTHERWISE`. A clause may be continued from one line to the next by using a comma at the end of the line. This then acts like a blank. Open *strings* or *comments* are not affected by line ends, and do not require a continuation character.
Keywords are shown in capitals in this list, however they may appear in either (or mixed) case. Keywords are only reserved when they are found in the correct context.

In the descriptions below: *expr* is an expression as described above; *stmt* is **any** one of the listed statements; *template* is a parsing template, as described in a later section; *name* is usually the name of a variable (see above).

name `= [`*expr*`];` — assignment: the variable *name* is set to the value of *expr*.

expr; — the value of *expr* is issued as a command.

`ADDRESS [{`*name*`|`*string*`} [`*expr*`]];` — redirect commands or a single command to new environment.

`ARGS [`*template*`];` — parse the argument string into variables. The contents of all variables except the last are translated to upper case. (Note: the first argument is always the name of the Exec or subroutine.)

`CALL` *name* `[`*expr*`];` — call an internal subroutine. On return, the variable *name* will have the value from the `RETURN` statement. Subroutines may be called recursively.

`DO [`*name* `=` *expri* `[TO` *exprt*`] [BY` *exprb*`]] [{UNTIL|WHILE}` *expr*`];`
`[`*stmt*`]... END;` — statement grouping with optional repetition and condition. The variable *name* is stepped from *expri* to *exprt* in steps of *exprb*. These *exprs* are evaluated only once at the top of the loop and must result in a whole number. This iterative phrase may be replaced by a single *expr* which is a loop count (no variable used). If a `WHILE` or `UNTIL` is given, its *expr* must evaluate to '0' or '1'. The condition is tested at the top of the loop if `WHILE` or at the bottom if `UNTIL`.

`DROP [`*name*`]...;` — drop (reset) the named, or all, variables.

`EXIT [`*expr*`];` — leave the Exec [with return code].

`IF` *expr* `{;|THEN}` *stmt*
`[ELSE[;]` *stmt*`]` — if *expr* evaluates to '1', execute the statement following the ';' or `THEN`. Otherwise (evaluates to '0') skip that statement and execute the one following the `ELSE` clause, if present.

`INTERPRET` *expr;* — evaluate *expr* and then execute the resultant string as if part of the original program.

`ITERATE [`*name*`];` — start next iteration of innermost repetitive loop [or loop with control variable *name*].

`LEAVE [`*name*`];` — terminate innermost loop [or the loop with control variable *name*].

`NOP;` — dummy statement, has no side-effects.

`PARSE ARGS [`*template*`];` `ARGS` without upper case translation.

`PARSE PULL [`*template*`];` `PULL` without upper case translation.

`PARSE SOURCE [`*template*`];` parse program source description `'CMS {COMMAND|FUNCTION} fn ft fm'`.

`PARSE VAR` *name* `[`*template*`];` parse the value of *name*.

`PARSE VERSION [`*template*`];` parse data describing interpreter.

`PROCEDURE;` — start a new generation of variables within a subroutine.

`PULL [`*template*`];` — read the next string from the system queue ("stack") and parse it into variables. The contents of all variables except the last are translated to upper case.

`PUSH [`*expr*`];` — push *expr* onto head of the system queue ("stack LIFO").

`QUEUE [`*expr*`];` — add *expr* to the tail of the system queue ("stack FIFO").

`RETURN [`*expr*`];` — evaluate *expr* and return the value to the caller. (Pushes the value onto the system queue if not a function or internal call.)

`SAY [`*expr*`];` — evaluate *expr* and then display the result on the user's console, using current line size.

`SELECT;`
`[WHEN` *expr*`{;|THEN}` *stmt*`]...`
`[OTHERWISE[;]` *stmt*`]...]`
`END;` — the `WHEN` *exprs* are evaluated in sequence until one results in '1'. the *stmt* immediately following it is executed and control then leaves the construct. If no *expr* evaluates to '1', control passes to those *stmts* following the `OTHERWISE` which must then be present.

`SIGNAL {ON|OFF}` *{name|string}*; — enable or disable exception traps. (The condition must be `ERROR`, `EXIT`, `NOVALUE`, or `SYNTAX`, and control will pass to the label of that name should the event occur while `ON`.)

`SIGNAL` *expr;* — go to the label specified. Any pending statements, `DO ... END`, `INTERPRET`, etc. are terminated.

`TRACE` *expr;* — if numeric then (if negative) inhibit tracing for a number of clauses, or (if positive) inhibit debug mode for a number of clauses. Otherwise trace according to first character of the value of *expr*:
'E' = trace after non-zero return codes.
'C' = trace all commands.
'A' = trace all clauses.
'R' = trace all clauses and expressions.
'I' = as 'R', but trace intermediate evaluation results and name substitutions also.
'L' = trace only labels.
'S' = display rest of program without any execution (shows control nesting).
'0' or null = no trace.
'!' = trace according to the next character, and inhibit command execution.
'?' = turn debug mode (pause after trace) on or off.

name: — form of labels for `CALL` or `SIGNAL`. The colon always acts as a clause separator.

`/*` form of comment `*/` may be used anywhere except in the middle of a *name* or *string*. (Required on first line to identify REX Execs.)

TEMPLATES for ARGS, PULL, and PARSE

The `PULL`, `ARGS`, and `PARSE` instructions use a *template* to parse a string. The template specifies the *names* of variables that are to be given new values, together with optional triggers to control the parsing. Each *name* in the template is assigned one word (without any leading or trailing blanks) from the input string in sequence, except that the last *name* is assigned the remainder of the string (if any) unedited. If there are fewer words in the string than names in the template, all excess variables are set to null. In all cases, **all** the variables in the template are given a new value.

If `PULL` or `ARGS` are used, then the separately assigned words **only** will first be translated to upper case. When this translation is not desired, use the `PARSE` instruction.

The parsing algorithm also allows some pattern matching, in which you may "trigger" on either a *string* or a *special-character* (the '(' is useful in the CMS environment, for example). A *special-character* is one of:

`+ - * / | & = ¬ < > , : ) (`

If the template contains such a trigger, then alignment will occur at the next point where the trigger exactly matches the data. A trigger match splits the string up into separate parts, each of which is parsed in the same way as a complete string is when no triggers are used.

COMPOUND VARIABLE NAMES

Any *name* may be "compound" in that it may be composed of several parts (separated by periods) some of which may have variable values. The parts are then substituted independently, to generate a fully resolved *name*. In general

$s_0.s_1.s_2.$ --- $.s_n$ will be substituted to form:
$d_0.v_1.v_2.$ --- $.v_n$ where d_0 is upper case of s_0
v_1-v_n are values of s_1-s_n.

This facility may be used for content-addressable arrays and other indirect addressing modes. As an example, the sequence:

```
J = 5;  A.J = 'fred';
```

would assign 'fred' to the variable '`A.5`'.

BUILT-IN VARIABLES

There are two built-in variables:

`RC`	is set to the return code after each executed command.
`SIGL`	is set to the line number of last line that caused `SIGNAL`, `CALL` or `RETURN` jump.

ISSUING COMMANDS to CMS

The default environment for commands in Execs is CMS. A command is an expression, which may include *function-calls*, arithmetic operations, and so on. Operators or other special characters (for example '(' or '*') must therefore be specified in a *string* if they are to appear in the issued command.

To issue a CP command or call another Exec, the first word of the expression value should be '`CP`' or '`EXEC`' respectively. Use the `OBEY` command instead if full CMS command resolution is to be applied.

In editor macros, the default environment for commands is the same as the filetype of the macro.

IBM

Mike Cowlishaw, Mail Point 182,
IBM UK Laboratories, Hursley Park, Winchester, UK.

CJN address: REXMAIL at WINPA
Telephone: (UK) 0962-4433

INTERACTIVE DEBUGGING

Execution of a TRACE instruction with a prefix '?' turns on debug mode. REX will then pause after most instructions which are traced at the console. You may then do one of three things:

(1) Enter a null line to continue execution.

(2) Enter an '=' to re-execute the clause last traced.

(3) Enter a list of REX instructions, which are interpreted immediately (DO-END statements must be complete, etc.). During execution of the string, no tracing takes place, except that non-zero return codes from host commands are displayed. Execution of a TRACE instruction with the '?' prefix will turn off debug mode. Other TRACE instructions affect the tracing that will occur when normal execution continues.

The numeric form of the TRACE instruction may be used to allow sections of program to be executed without pause for debug input. 'TRACE *n*', (i.e. positive result) will allow execution to continue without pause for *n* traced clauses. 'TRACE -*n*', (i.e. a negative result) will allow execution to continue without pause and with tracing inhibited for *n* clauses that would otherwise be traced.

REXFNS PACKAGE of BASIC FUNCTIONS

REX has no "built-in" functions although several packages of external functions are available. The REXFNS package will be loaded automatically if needed, and includes the following:

DATATYPE(string)	returns 'NUM' if the string is a valid number otherwise returns 'CHAR'.
DATE()	returns the date e.g. '6 Nov 80'.
DELSTR(string,n[,k])	deletes specified sub-string.
INDEX(haystack,needle)	returns the position of the needle in the haystack (same format as PL/I).
LASTPOS(needle,haystack)	returns the position of the last occurrence of the needle in the haystack.
LENGTH(string)	returns the length of the string.
LINESIZE()	returns terminal line length.
NEST()	will return the current depth of Exec nesting (independent of Exec language).
POS(needle,haystack)	returns the position of the needle in the haystack.
READFLAG()	returns 'STACK' if the next PULL will read from the stack, otherwise returns 'CONSOLE'.
REPEAT(string,n)	returns n+1 concatenated copies of string.
STRIP(string[,{'L'|'T'|'B'}])	returns string less leading, trailing, or both sets of blanks. Default is 'T'.
SUBSTR(string,n[,k])	returns the sub-string of string which begins at the nth character, and is of length k.
SYMBOL('*name*')	returns 'VAR' if the *name* has been used as a variable, otherwise returns 'LIT'.
TIME()	returns the local time e.g. '03:23:35'.
TRANS(string[,to-table[,from-table]])	Same as PL/I TRANSLATE function, except that default translate tables convert string to upper case.
TRUNC(string[,n])	returns all of the string up to the first '.' plus up to n characters after it (default n=0).
USERID()	returns Virtual Machine userid.
WCOUNT(string)	returns the number of words in the string.

REXFNS2 PACKAGE of EXTENDED FUNCTIONS

The extended function package is loaded automatically when needed. Note that the Logical functions (AND, OR, etc.) act on the individual bits within the character string arguments.

AND(string,string[,pad])	logically AND the strings.
CLCL(string,string[,pad])	compare character strings.
CLXL(hexes,hexes[,pad])	compare hex strings.
COUNTBUF()	return depth of Stack.
D2X(number[,length])	convert decimal to hex characters.
E2X(hexes)	pack hex characters to bytes.
FETCH(addr[,length])	get bytes from storage. addr must be in binary form.
OR(string,string[,pad])	logically OR the strings.
REVERSE(string)	reverses the string.
SUBSET()	returns '1' if in CMS SUBSET, else '0'.
TM(string,mask[,pad])	test string under mask.
TYPEFLAG([ht|rt])	test and set typing control flag.
VERIFY(string,ref[,¬])	check the string for invalid characters.
XOR(string,string[,pad])	exclusive OR the strings.
XRANGE([startchar[,endchar]])	return range of characters.
X2D(hexes)	convert hex characters to decimal.
X2E(string)	unpack bytes to hex characters.

REXWORDS PACKAGE of WORD PROCESSING FUNCTIONS

These functions all treat data as a series of words delimited by blanks. The package is loaded automatically when needed.

CENT(string,k)	returns string centred (width k).
FIND(string,phrase)	returns the word number of the first word where phrase matches the string. Returns '0' if the phrase is not found.
JUSTIFY(string,k)	justifies string to both margins (width k), by adding blanks between words.
LEFT(string,k)	returns a string of length k with string left justified in it.
RIGHT(string,k)	returns string right justified (width k).
SPACE(string,n)	puts n spaces between each word. n may be 0, to remove all blanks.
WORD(string,n)	returns the nth word in the string.

UTILITY MODULES

All commands that may be called from EXEC or EXEC 2 may be used with REX, except those that always attempt to set "Old EXEC" variables. Some of the modules available which are especially useful with Execs are:

CLEAR	clears a 3270 screen.
CONGET	does an immediate read to the Console regardless of the state of the input queue.
EMSG	behaves like &EMSG in CMS EXEC.
FSX	allows full screen control of console or dialed 3270s, especially useful for application modelling.
IEMP	allows Execs and macros to be kept in virtual storage to avoid disk I/O.
IOS3270	general menu manipulation program, directly sets REX variables, can display screens from libraries, etc.
IOX	uses the REX variables interface for I/O to CMS files, PUNCH, etc.

MODULES	tests for the existence of modules called from an Exec. Strongly recommended for use in any Execs which are "for export".
OSRESET	resets storage for PL/I etc.
PROMPT	will prompt the 3270 user with data in the command input area.
QEXEC	detects which of REX and EXEC2 are currently active.
REXDUMP	is a debugging aid that displays up to 50 characters, and the length, of all currently active variables.
RXLOCATE	function: controlled search for one string in another. Call as: LOCATE(needle,haystack[,n[,'¬']])
RXRND	function: returns random integer. Call as: RND([lower-limit[,upper-limit]])
STACKIO	general I/O package, similar to IOX, but interfacing to Execs via the Stack.

RESTRICTIONS

There are no restrictions on the length or content of manipulated character data (other than your Virtual Machine size).

CMS Restrictions:

Exec files cannot be more that 65,535 bytes wide or more than 65,533 (*sic*) records long.

Lines read from the console cannot exceed 130 bytes. REX will format output lines of any length to fit the console.

Stacked lines are limited to 130 or 255 bytes (depending on CMS Release).

Commands entered from CMS command level are translated to upper case by CMS before being passed to REX.

Command names, function names, and the environment named by an ADDRESS statement will be truncated to 8 characters.

Implementation Restrictions:

The *name* of a variable or label may not exceed 150 bytes, and a literal *string* may not exceed 250 bytes.

The internal representation of a clause (after removal of comments, extra blanks etc.) may not exceed 500 bytes.

A *number* may not have more than 9 digits before and/or after the decimal point.

The control stack (for IF, CALL etc.) is limited to depth 100.

Functions cannot have more than 7 arguments.

COMMAND and EXEC PLIST

REX Execs may be invoked from programs via SVC 202 and the standard CMS Plist. The Plist can be extended to allow an untokenised string to be passed and also to permit execution directly from storage.

For in-store execution, Filename and Filetype are still required in the file block, since these determine the logical program name and the default command environment.

REX always provides an extended Plist (without a file block) when invoking commands.

The standard CMS Plist consists of a series of 8-byte tokens, pointed to by GPR1, and terminated by 8X'FF'. The top byte of GPR1 may be set to X'00'. If the top byte of GPR1 is set to X'01' then this signifies that GPR0 points to the extended Plist. For calling REX, this has the form:


```
* The extended Plist:
*     a) defines the argument string
*     b) points to an optional File Block
EPL  DS  OF            ** Extended Plist
     DC  A(COMVERB)    -> CL5'EXEC '
     DC  A(BEGARGS)    -> argument string
     DC  A(ENDARGS)    -> character after end of
*                         argument string
     DC  A(FBL)        -> file block, if present,
*                         otherwise is A(0)
* The file block (only required for non-
* EXEC or in-storage files)
FBL  DS  OF            ** File block
     DC  CL8'name'     logical name of program
     DC  CL8'type'     default destination for
*                       commands (blanks or 'EXEC'
*                       both default to CMS)
     DC  CL2'mode'     should normally be ' '
     DC  H'extlen'     length of extension block
*                       in fullwords: H'2' for in-
*                       store execution, else H'0'
* Extension block starts here (only required
* for in-store program):
     DC  AL4(PGM)      -> Descriptor list start
     DC  AL4(PND-PGM)Length of descriptor list
* Descriptor list for in-store program:
PGM  DS  OF
     DC  A(line.1)     Address of line.1
     DC  F'len.1'      Length of line.1
      ...
     DC  A(line.k)     Address of line.k
     DC  F'len.k'      Length of line.k
PND  EQU *
```

SAMPLE REX EXEC

```
/* MOVE:  Move file to another disk */
Args me Fn Ft Fm (nfm .
If fn='' | fn='?' | nfm='' then do
  say 'Syntax is:  MOVE Fn [Ft [Fm]]  (x'
  say '  where "x" is the target disk.'
  exit 100; end

IOX STATE Fn Ft Fm '(FN FT FM'
if rc=28 then say 'Nothing to move!'
if rc¬=0 then exit rc

STATE fn ft nfm
if rc=0 then do
  Say '"'fn ft nfm'" exists: "Y" to replace'
  pull ans .
  if ans ¬= 'Y' then do
    Say 'File not moved'
    Exit; end
  end
 else if rc¬=28 then exit rc

COPYFILE fn ft fm '= =' nfm '(OLDDATE REPLACE'
if rc=0  then ERASE fn ft fm
if rc¬=0 then exit rc
Say '"'fn ft'" moved to disk "'nfm'"'
```