BREXX/370 V2R2M0 User's Guide

Document Version 1.0
Authors: Peter Jacob (pej), Mike Großmann (mig), Gerard Wassink (gaw)

I. BREXX/370 User's Guide

In this user's guide, only changes and amendments to the official BREXX User's guide are documented. For the BREXX standard functions and commands refer to Vassilis N. Vlachoudis BREXX documentation at https://ftp.gwdg.de/pub/languages/rexx/brexx/html/rx.html

A. Some Notes on BREXX Arithmetic Operations

BREXX stores numeric values in the appropriate type format. The benefit compared to store it as strings is a significant performance improvement during calculations. As the expensive string to numeric conversion before and vice versa after arithmetic operations is omitted. This allows speedy calculations without the required conversion overhead.

BREXX supports 2 numeric types:

1. Integer

Integers are stored in 4-bytes a full word (LONG), this means their range is from -2,147,483,648 to +2,147,483,647

2. Decimal Numbers

Decimal Numbers (decimal numbers with a fractional part) are represented in the double-precision floating-point format (doubleword), the length is 8-bytes consisting of an exponent and the significand (fraction). It consists of 56 bits for the fraction part, 7-bit exponent and one-bit for the sign. This representation is IBM specific and differs slightly from the IIEE 754 floating-point standard.

The precision of floating-point numbers is not as good as decimal packed numbers which are not supported in BREXX (nor in REXX). This means, for example, 2.0 might be stored as 19999999999999999e-17, or for 5.0 you will is stored as 50000000000000003e-17. This is not an error, but the usual behaviour for floating-point numbers. This is caused by the conversion between the numbers of base 10 to base 2 a bit-exact reversibility is not always given. This effect may build up during arithmetic calculations.

II. Calling external REXX Scripts or Functions

Due to the extended calling functionality in the new version, an import of required REXX scripts is no longer required. You can now call any external REXX script directly.

A. Main REXX Script location via fully qualified DSN

If you call a REXX script using a fully qualified partitioned dataset (PDS) member name, it must be present in the specified PDS. You can also use a fully qualified sequential dataset name that holds your script. If it is not available an error message will terminate the call. In TSO you can invoke your script using the REXX or RX commands.

Example: **RX 'MY.EXEC(MYREX)'** if the script resides in a PDS, alternatively:

RX 'MY.SAMPLE.REXX' if it is a sequential dataset

B. Location of the Main REXX script via PDS search (TSO environments)

In TSO environments the main script can be called with the RX or REXX command. The search path for finding your script is SYSUEXEC, SYSUPROC, SYSEXEC, SYSPROC. At least one of these need to be pre-allocated during the TSO logon. It is not mandatory to have all of them allocated, it really depends on your planned REXX development environment. The allocations may consist of concatenated datasets.

C. Running scripts in batch

In batch, you can use the delivered RXTSO or RXBATCH JCL procedure and specify the REXX script and its location to execute it. There is no additional search path used to locate it.

D. Calling external REXX scripts

It is now possible to call external REXX scripts, either by:

CALL your-script parm1,parm2… or by function call:

value=your-script(parm1,parm2,…)

The call might take place from within your main REXX, or from a called subroutine. The search of the called script is performed in the following sequence:

- Internal sub-procedure or label (contained in the running REXX script)
- current PDS (where the calling REXX is originated)[1]
- from the delivered BREXX.RXLIB library, which then needs to be allocated with the DD-name RXLIB

E. Variable Scope of external REXX scripts

If the called external REXX does not contain a procedure definition, all variables of the calling REXX are accessible (read and update). If the called REXX creates new variables, they will be available in the calling REXX after control has been returned.

[1] only from the 1st library within a concatenation (this limitation may be lifted in a forthcoming release)

III. BREXX MVS Functions

A. Added BREXX Kernel functions and Commands

These are MVS-specific BREXX functions implemented and integrated into the BREXX kernel code. For the common BREXX functions take a look into the BREXX User's Guide.

3. General

ABEND(user-abend-code)

Terminates the program with specified User-Abend-Code. Valid values for the user evening abend-code are values between 0 and 4095.

EXECIO Command

The EXECIO is a host command, it needs to be therefore coded in apostrophes. There is just a subset of the known EXECIO implemented: Full read/write from a dd-name. The ddname must be allocated either by TSO ALLOC command, or DD statement in the JCL. Specifying a Dataset-Name (DSN) is not supported!

```
/* Read entire File into Stem-Variable*/
"EXECIO * DISKR dd-name (STEM stem-name."
/* Write Stem-Variable into File */
"EXECIO * DISKW dd-name (STEM stem-name."
```

After completing the Read stem-name.0 contains the number of records read
The number of lines to become written to the file is defined in stem-variable.0

The asterisk is a placeholder for the reading/writing the entire file. Replacing it by a number of lines does not have a different effect. It will always process the entire file.

USERID()

Returns the identifier of the currently logged-on user. (available in Batch and Online)

WAIT(wait-time)

Stops REXX script for some time, wait-time is in hundreds of a second

WTO(console-message)

Write a message to the operator's console. It will also appear in the JES Output of the Job.

4. TSO REXX functions

SYSDSN(dataset-name) or SYSDSN(dataset-name(member-name))

Returns a message indicating whether a dataset exists or not.

OK	dataset or member is available
DATASET NOT FOUND	dataset or member is not available
INVALID DATASET NAME, dsname	given dataset name is not valid
MISSING DATASET NAME	no dataset name given

Example:

```
x=SYSDSN("'HERC01.TEST.DATA'")
IF x = 'OK' THEN
  do something
ELSE
  do something other
```

SYSVAR(request-type)

a TSO-only function to retrieve certain TSO runtime information, as running in foreground/background, SYSUID, etc.

LISTDSI(dataset-name) or LISTDSI('dd-name FILE')

Returns information of the dataset in REXX variables, as SYSDSNAME, SYSVOLUME, SYSDSORG, SYSRECFM, SYSLRECL, SYSBLKSIZE

B. VSAM IO Functions

The VSAM IO Functionality is documented in BREXX370_VSAM_Users_Guide_V2R2M0.pdf delivered within the installation file BREXX370_V2R2M0-Final.zip

C. Formatted Screen Functions

The Formatted Screen Services is documented in BREXX370_Formatted_Screens_V2R2M0.pdf delivered within the installation file BREXX370_V2R2M0-Final.zip

D. RXLIB functions

BREXX has the capability to implement new functions or commands in REXX. They are transparent and will be called in the same way as basic BREXX functions. They are stored in the library BREXX.RXLIB and are automatically allocated (via DD RXLIB) in RXBATCH and RXTSO (Batch). In this release we deliver the following:

A2E(ascii-string)

Translates an ASCII string into EBCDIC. Caveat: not all character translations are biunique!

~~BSTORAGE(decimal-storage-address,storage-length)~~

~~Storage command in the original BREXX decimal implementation. The storage address is in decimal.~~
BSTORAGE has been removed as it was a temporary solution for users of the very first BREXX/370 version. To keep it, take it from a saved version of BREXX.RXLIB.

CEIL(decimal-number)

Smallest integer greater or equal then decimal number.

FLOOR(decimal-number)

Greatest integer less or equal then decimal number.

RXMSG(msg-number,'msg-level','message')

Standard message module to display a message in a formatted way
msg-number message number to be displayed
msg-level message level can be

- I for an information message
- W for a warning message
- E for an error message
- C for a critical message

Examples:

```
rc=rxmsg( 10,'I','Program started')
rc=rxmsg(200,'W','Value missing')
rc=rxmsg(100,'E','Value not Numeric')
rc=rxmsg(999,'C','Divisor is zero')
```

Displayed output:

```
RX0010I    PROGRAM STARTED
RX0200W    VALUE MISSING
RX0100E    VALUE NOT NUMERIC
RX0999C    DIVISOR IS ZERO
```

Additionally the following REXX variables are maintained, and can be used in the calling REXX script.

Return code from call RXMSG

0	an information message was written
4	a warning message was written
8	an error message was written
12	a critical message was written

MSLV contains the written message level

I	an information message was written
W	a warning message was written
E	an error message was written
C	a critical message was written

MSTX contains the written message text part

MSLN contains the complete message with the message number, message level and text

MAXRC contains the highest return code so far. This can be used to exit the top level REXX. If you used nested procedures it is required to expose MAXRC, to make it available in the calling procedures.

B2C(bit-string)

Converts bit string into a Character string

Examples:

```
say B2C('1111000111110000')  ->      10
say B2c('1100000111000010')  ->      AB
```

C2B(character-string)

Converts a character string into a bit string

Example:

```
say c2x('64'x)c2B('64'x)    ->    64 01100100
say c2x(10) c2B(10)         ->    F1F0 1111000111110000
say c2x('AB') c2B('AB')     ->    C1C2 1100000111000010
```

Caveat: Not all character translations are biunique!

created 23.3.19, updated 26.10.19 10:15

DCL('$DEFINE','structure-name')
DCL('field-name',[offset],length,[type])

Defines a structure of fields which maps typically to an I/O record. The function returns the next available offset in the structure.

$DEFINE	initializes the structure definition
structure-name	all following field definitions are associated with the structure-name.
field-name	name of the rexx variable containing/receiving the field content of the record
offset	offset of the field in the record. This definition is optional if left out the next offset from the previous DCL(field…) definition is used, or 1 if there was none.
length	length if the field in the record
type	field-type
CHAR	no translation takes place, CHAR is default
PACKED	Decimal Packed field. Translation into/from decimal packed into Numeric REXX value takes place

`call SPLITRECORD 'structure_name,record-to-split`

splits record-to-split in the defined field-names (aka REXX variables). The variable containing the record to split is typically read from a dataset.

`Record=SETRECORD('student')`

Combines the content of all defined fields (aka REXX variables) at the defined position and the defined length to a new record.

Example

```
n=DCL('$DEFINE','student')
n=DCL('Name',1,32,'CHAR')
n=DCL('FirstName',1,16,'CHAR')
n=DCL('LastName',,16,'CHAR')
n=DCL('Address',,32,'CHAR')
recin='Fred             Flintstone      Bedrock'
/*    '12345678901234567890123456789012345678901234567890   */
call splitRecord 'student',recin
say Name
say FirstName
say LastName
say Address
firstName='Barney'
LastName='Rubble'
address='Bedrock'
say setRecord('student')
```

DEFINED('variable-name')

Tests if variable or STEM exists, to avoid variable substitution the variable-name must be enclosed in quotes.

return values:

-1 not defined, but would be an invalid variable name
0 variable-name is not a defined variable
1 variable-name is defined it contains a string
2 variable-name is defined it contains a numeric value

To test whether a variable is defined you can use:

```
If defined('myvar')> 0 then …
```

DAYSBETW(date1,date-2[,[format-date1],[format-date2]])

Return days between 2 dates of a given format.

format-date1 date format of date1 defaults to European
format-date2 date format of date2 defaults to European
the format-dates reflect the Input-Format of RXDATE and can be found in details there.

DUMP(string, [hdr])

Displays string as a Hex value. This is useful to check if a received a string contains unprintable characters. One can specify hdr as an optional title.

Dump example:

```
CALL Dump 'This is the new version of BREXX/370 V2R1M0','Dump Line'
```

Output:

```
Dump Line
0000(0000)  This  is  the  new    vers ion  of B REXX
0000(0000)  E88A 48A4 A884 98A4   A89A 8994 984C DCEE
0000(0000)  3892 0920 3850 5560   5592 9650 6602 9577

0032(0020)  /370  V2R 1M0
0032(0020)  6FFF 4EFD FDF
0032(0020)  1370 0529 140
```

E2A(EBCDIC-string)

Translates an EBCDIC string into ASCII. Caveat: not all character translations are biunique!

JOBINFO()

Return information about currently running job or TSO session in REXX variables, like JOBNAME, JOBNUMBER, STEPNAME, PROGRAMNAME

LINKMVS(load-module, parms)

Starts a load module. Parameters (if any) will be sent in the same format as on the JCL PGM=…,PARMS='….' would pass them.

LISTALC()

Lists all allocated Datasets in this session or region.

```
SYS00003  SYS1.UCAT.TSO
SYSUEXEC  PEJ.EXEC
SYS00014  SYS1.UCAT.MVS
SYSEXEC   SYS2.EXEC
ISPCLIB   SYS2.ISP.CLIB
          ISP.V2R0M0.CLIB
ISPLLIB   SYS2.ISP.LLIB
          ISP.V2R0M0.LLIB
ISPMLIB   SYS2.ISP.MLIB
          ISP.V2R0M0.MLIB
ISPPLIB   SYS2.ISP.PLIB
          ISP.V2R0M0.PLIB
          SYS2.REVIEW.PLIB
ISPSLIB   SYS2.ISP.SLIB
          ISP.V2R0M0.SLIB
ISPTLIB   SYS2.ISP.TLIB
          ISP.V2R0M0.TLIB
ISPTABL   SYS2.ISP.TLIB
          ISP.V2R0M0.TLIB
…
```

LOWER(string)

Returns translated lower case string

MVSCBS()

Allows addressing of some MVS control blocks. There are severy dependent control blocks combined. To use them MVSCBS must be imported first. Thereafter they can be used.

Currently integrated control blocks are:

Cvt(), Tcb(), Ascb(), Tiot(), Jscb(), Rmct(), Asxb(), Acee(), Ecvt(), Smca()

The definition and the content of the MVS control blocks can be found in the appropriate IBM manuals: MVS Data Areas, Volume 1 to 5.

IMPORT command is described in Vassilis N. Vlachoudis BREXX documentation.

QUOTE(string,qtype)

Enclose string in quotes, double quotes, or parenthesis,

Qtype can be :

'	single quote (default)
"	double quote
(	bracket, the closing character will be)
[	square bracket, the closing character will be]

```
Mystring='string to be quoted'

Say QUOTE(mystring,'"')                -> "string to be quoted"
Say QUOTE(mystring,"'")                -> 'string to be quoted'
Say QUOTE(mystring,'(')                -> '(string to be quoted)'
Say QUOTE(mystring,'[')                -> '[string to be quoted]'
```

PDSDIR(pds-name)

Return all member names from the given PDS in a stem variable.

Example REXX

```
num=PDSDIR('BREXX.RXLIB')
do i=1 to num
   say PDSList.Membername.i
end
```

Result

```
A2E
BSTORAGE
B2C
C2B
DAYSBETW
DEFINED
DUMP
E2A
JOBINFO
LINKMVS
LISTALC
…
```

PDSRESET(pds-name)

Removes all members of a PDS and runs a compress. After execution the PDS is empty.

READALL(file,variable[,'DSN'/'DDN'])

Reads entire file into a stem variable. The file can be either a dd-name or a ds-name

After a successful completion, the stem variable.0 will contain the number of lines read into the stem.

The file name can either represent an allocated dd name or a fully qualified ds name. The third parameter defines the file type and is either DSN or DDN. If it is missing DDN is the default.

PERFORM(pds-name,process-member-rexx)

Reads member list of a PDS and runs the process-member-rexx against each member.
The REXX to be called receives the parameters:

Pds-name
Member-name

RXDATE(…)

RXDATE Transforms Dates from/to various formats

RXDATE(<output-format>,<date>,<input-format>)

date is formatted as defined in input-format, it defaults to today's date

Input Format represents the input date format, it defaults to 'EUROPEAN'

Base	days since 01.01.0001	
JDN	days since 24. November 4714 BC	
Julian	yyyyddd	e.g. 2018257
European	dd/mm/yyyy	e.g. 11/11/2018
German	dd.mm.yyyy	e.g. 20.09.2018
USA	mm/dd/yyyy	e.g. 12.31.2018
STANDARD	yyyymmdd	e.g. 20181219
ORDERED	is yyyy/mm/dd	e.g. 2018/12/19

Output Format represents the output date format, it defaults to 'EUROPEAN'

Apart from the formatting options that can be specified for the input, for the output we can additionally specify the following:

Days	ddd days this year e.g. 257	
Weekday	weekday	e.g. Monday
Century	dddd days this century	
SHORT	dd mon yyyy	e.g. 28. OCT 2018
LONG	dd month yyyy	e.g. 12. MARCH 2018

RXSORT(sort-type[,ASCENDING/DESCENDING])

Sorts the stem variable SORTIN. SORTIN.0 must contain the number of entries of SORTIN. The sort algorithms supported are:

QUICKSORT, SHELLSORT, HEAPSORT, BUBBLESORT

After Completion of RXSORT the stem variable SORTIN. is sorted. If you requested ASCENDING (also default) it is in ascending order, for DESCENDING in descending order.

Sorting with REXX is only recommended for a small number of stem entries. Up to 1000 entries, RXSORT works in a reasonable time.

If the stem you want to sort is not in SORTIN. you can use the SORTCOPY function to copy it over to SORTIN.

SEC2TIME(seconds[,'DAYS'])

Converts a number of seconds into the format hh:mm:ss, or days hh:mm:ss if the 'DAYS' parameter is specified.

```
say sec2Time(345000)              ->    95:50:00
say sec2Time(345000,'DAYS')       ->    3 day(s) 23:50:00
```

SORTCOPY(stem-variable)

Copies any stem variable into the stem SORTIN., which then can be used by RXSORT.
Stem-variable.0 must contain the number of entries of the stem.

STEMCOPY(source-stem-variable,target-stem-variable)

Copies any stem variable into another stem variable.
source-stem-variable.0 must contain the number of entries of the stem.
Stem-variables must end with a trailing '.', e.g. 'MYstem.'

STEMCLEN(stem-variable)

Cleansing of a stem variable, it removes empty and unset stem items and adjusts the stem numbering. Stem-variable.0 must contain the number of entries of the stem and will after the cleansing the modified number of entries.

Stem-variables must end with a trailing '.', e.g. 'MYstem.'

STEMGET(dataset-name)

Reads the saved content of one or more stem variables and re-apply the stem. Stem names are save in the dataset.

STEMINS(stem-to-insert,insert-into-stem,position)

Inserts **stem-to-insert** into **insert-into-stem** beginning at position. The content of the original stem at the position will be shifted down n positions, whereby n is the size of the stem to be inserted. Stem-variable(s).0 must contain the number of entries of the stem. Stem-variables must end with a trailing '.', e.g. 'MYstem.'

STEMPUT(dataset-name,stem1[,stem2{,stem3]…)

Saves the content of one or more stems in a fully qualified dataset-name

Stem-variable.0 must contain the number of entries of the stem. Stem-variables must end with a trailing '.', e.g. 'MYstem.'

STEMREOR(stem-variable)

Reorders stem variable from top to bottom. 1. element becomes last, 2. next to last, etc.
Stem-variable.0 must contain the number of entries of the stem. Stem-variables must end with a trailing '.', e.g. 'MYstem.'

STORDUMP(storage-address,storage-length, [hdr])

Displays a MVS storage area as a Hex value. One can specify hdr as an optional title.

Example:

```
CALL StorDump 16,64,'CVT 64 Bytes'
CVT 64 Bytes
00000010 +0000(0000)  ..:  :..: %::* :..S   î³:* :..S ëÓ.. ....
00000010 +0000(0000)  0000 2000 6105 300E   5F05 300E 5E00 0000
00000010 +0000(0000)  007C 0001 CA7C 0002   6A7C 0002 3E00 0000

00000030 +0032(0020)  .... .... ...: Çy .   ...: ÇÌ.:  :ò: Ác..
00000030 +0032(0020)  0000 0000 0000 6A00   0000 6700 40C3 6800
00000030 +0032(0020)  0000 0000 0008 88C0   0008 8801 08D4 5300
```

TODAY([date-format])

Returns today's date based on requested format. Details of date-formats can be found in the RXDATE output-format description

UNQUOTE(string)

Remove from string leading and trailing quotes, double quotes, parenthesis and '<' and '>' signs.

```
Say UNQUOTE(" 'quoted-string' ") ->    quoted-string
Say UNQUOTE("<entry 1>")               ->    entry 1
Say UNQUOTE("(entry 2)")               ->    entry 2
Say UNQUOTE("[entry 3]")               ->    entry 3
```

UPPER(string)

Returns translated upper case string.

VERSION(['FULL'])

Returns BREXX/370 version information, if FULL is specified additionally the Build Date of BREXX is returned.

```
SAY VERSION()          ->    V2R1M0
SAY VERSION('FULL')    ->    Version V2R1M0 Build Date 03. Apr 2019
```

WRITEALL(file,variable[,'DSN'/'DDN'])

Writes a stem variable into a file. The file can be either a dd-name or a ds-name

The stem variable.0 must contain the number of entries of the stem.

The file name can either represent an allocated dd name or a fully qualified ds name. The third parameter defines the file type and is either DSN or DDN. If it is missing DDN is the default.

For further details of these functions refer to the BREXX/370 User's Guide.

Credits: we would like to thank Gerard Wassink for proofreading and making improvement suggestions to this document.

Table of contents