BREXX/370 V2R5M3 User's Guide

Document Version 5.3
Authors: Peter Jacob (pej), Mike Großmann (mig)

I. BREXX/370 User's Guide

This user's guide documents only changes and amendments to the official BREXX User's manual. 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 saving 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 two numeric types:

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

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, a 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 1999999999999999e-17, or for 5.0 you will be stored as 50000000000000003e-17; this is not an error, but the usual behaviour for floating-point numbers. It 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.

B. B Labels/Procedures

There is no check for duplicate labels or procedures in the running rexx script, or subsequently called external rexx scripts. So a loaded script may use the same labels as the ones defined in the so far compiled scripts. Duplicate labels are always ignored and the ones identified are used for a call to them. Therefore choose your labels and procedure names carefully to avoid unexpected results.

C. Loading Procedures Search Sequence

1. The initial rexx (main script) will be loaded from the following DD-name sequence:

   SYSUEXEC, SYSPROC, SYSEXEC, or a fully qualified DSN.

The first occurrence is taken; other sites are omitted.

2. Load of REXX scripts that are later called

   A REXX that is invoked and is not available will be loaded from the DD-name sequence:

   RXLIB, DD-name or the DSN from which the main script was taken.

All loaded scripts (1 and 2) are instantly translated and placed into the stack, from which they are executed. If a script's label, whether an internal label or a procedure, is already on the stack, no external version will be loaded.

D. Inline Comments

You can start a comment with **--** instead of the standard **/* comment… */.** In this case, anything following the sign until the end of the line is regarded as a comment.

```
A=1   /* Set variable */
B=2   -- Set variable
```

II. Calling external REXX Scripts or Functions

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

A. Primary 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 terminates 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 needs to be pre-allocated during the TSO logon. It is not mandatory to have all of them allocated. It 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 originated)[1]
- from the delivered BREXX.RXLIB library, which then needs to be allocated with the DD-name RXLIB

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

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 are available in the calling REXX after the control is returned. Vice versa the called Rexx knows all the procedures and labels used so far, this means you can define "call-back" procedures usable by the called REXX.

III. BREXX MVS Functions

A. Host Environment Commands

ADDRESS MVS

Interface to certain REXX environments such as VSAM and EXECIO

ADDRESS TSO

Interface to the TSO commands, e.g. LISTCAT, ALLOC, FREE, etc.

Using the ADDRESS TSO command requires a TSO command processor module of the specified name. It will be called using the normal MVS conventions. If the module can't be loaded an error message will be displayed:

```
Error: Command TIME not found
1 - ADDRESS TSO TIME
+++ RC(-3) +++
```

Any parameter for the module is supplied to the module in the CPPL format.

TSO does some internal routing e.g. TIME, which is not a command processor module but will output the current time if performed in plain TSO. The BREXX command ADDRESS TSO TIME will lead to an error.

ADDRESS COMMAND **'CP host-command'**

Interface to the host system in which your MVS3.8 is running. Typically it is Hercules or VM370. For communication, a Hypervisor call is performed. The request must be enabled on the host system, or else you don't receive a result.

The result of the command is displayed on the screen, but can be trapped in a stem by the OUTTRAP command:

```
call outtrap('myresult.')
ADDRESS COMMAND 'CP help'
call outtrap('off')
/* result is stored in stem myresult. */
do i=1 to myresult.0
   Say myresult.i
end
```

Some Hercules commands:

`ADDRESS COMMAND 'CP HELP'` to get a list of Hercules commands

```
HHC01603I
HHC01602I Command               Description
HHC01602I ----------------      ----------------------------------
HHC01602I !message             *SCP priority message
HHC01602I #                     Silent comment
```

```
HHC01602I *                    Loud comment
HHC01602I .reply               *SCP command
HHC01602I ?                    alias for help
HHC01602I abs                  *Display or alter absolute storage
HHC01602I aea                  Display AEA tables
HHC01602I aia                  Display AIA fields
...
```

ADDRESS COMMAND 'CP DEVLIST' shows a list of all active devices

```
HHC02279I 0:010C 3505 jcl/dummy ascii trunc eof IOÝ2¨ open
HHC02279I 0:010D 3525 pch/pch10d.txt ascii IOÝ2¨ open
HHC02279I 0:0131 2314 dasd/sort01.131 Ý203 cyls¨ Ý0 sfs¨ IOÝ1570¨ open
HHC02279I 0:0132 2314 dasd/sort02.132 Ý203 cyls¨ Ý0 sfs¨ IOÝ1278¨ open
HHC02279I 0:0133 2314 dasd/sort03.133 Ý203 cyls¨ Ý0 sfs¨ IOÝ1262¨ open
...
```

And many others:

ADDRESS COMMAND 'CP clocks'

```
HHC02274I tod = DA19F16BF4009020    2021.214 07:49:10.743049
HHC02274I h/w = DA19E402B9C09020    2021.214 06:49:10.743049
HHC02274I off = 00000D693A400000       0.000 01:00:00.000000
HHC02274I ckc = DA19F16BF9400000    2021.214 07:49:10.764544
HHC02274I cpt = 7FFFFFE45026B700
HHC02274I itm = EAB2A163                     14:14:30.510471
```

If you run under the control of VM370 you can run VM commands

ADDRESS COMMAND 'CP vm-command'

ADDRESS FSS

Interface to the Formatted Screen Services. Please refer to BREXX370_Formatted_Screens_V2R5M3.pdf contained in the installation zip file.

The following host environments enable you to call external programs. The difference is the linkage conventions, and how input parameters are treated.

ADDRESS LINK/LINKMVS/LINKPGM

Call external an external program. The linkage convention of the called program can be found here:

The LINK and ATTACH host command environments (ibm.com)

ADDRESS LINKMVS

Call external an external program. The linkage convention of the called program can be found here:

The LINKMVS and ATTCHMVS host command environments (ibm.com)

Example:

```
/* REXX - INVOKE IEBGENER WITH ALTERNATE DDNAMES.   */
PROG   = 'IEBGENER'
PARM   = ''                        /* STANDARD PARM, AS FROM JCL    */
DDLIST = COPIES('00'X,8) ||,       /* DDNAME  1 OVERRIDE: SYSLIN    */
         COPIES('00'X,8) ||,       /* DDNAME  2 OVERRIDE:  N/A      */
         COPIES('00'X,8) ||,       /* DDNAME  3 OVERRIDE: SYSLMOD   */
         COPIES('00'X,8) ||,       /* DDNAME  4 OVERRIDE: SYSLIB    */
         LEFT('CTL',  8) ||,       /* DDNAME  5 OVERRIDE: SYSIN     */
         LEFT('REP',  8) ||,       /* DDNAME  6 OVERRIDE: SYSPRINT  */
         COPIES('00'X,8) ||,       /* DDNAME  7 OVERRIDE: SYSPUNCH  */
         LEFT('INP',  8) ||,       /* DDNAME  8 OVERRIDE: SYSUT1    */
         LEFT('OUT',  8) ||,       /* DDNAME  9 OVERRIDE: SYSUT2    */
         COPIES('00'X,8) ||,       /* DDNAME 10 OVERRIDE: SYSUT3    */
         COPIES('00'X,8) ||,       /* DDNAME 11 OVERRIDE: SYSUT4    */
         COPIES('00'X,8) ||,       /* DDNAME 12 OVERRIDE: SYSTERM   */
         COPIES('00'X,8) ||,       /* DDNAME 13 OVERRIDE:  N/A      */
         COPIES('00'X,8)           /* DDNAME 14 OVERRIDE: SYSCIN    */
ADDRESS 'LINKMVS' PROG 'PARM DDLIST'
```

ADDRESS LINKPGM

Call external an external program. The linkage convention of the called program can be found here:

[The LINKPGM and ATTCHPGM host command environments (ibm.com)]

ADDRESS ISPEXEC

support calls functions to Wally Mclaughlin ISPF for MVS on Hercules (e.g. TK4-). The functions supported depend on the functionality implemented in his API.

Example:

```
ADDRESS ISPEXEC
"CONTROL ERRORS RETURN"
"DISPLAY PANEL(PANEL1)"
```

OUTTRAP

If the commands write output to the terminal you can trap the output using the OUTTRAP command. This will re-direct it to a stem variable of your choice. Output produced by TSO full-screen macros cannot be trapped. OUTTRAP is not able to catch all output written to the terminal, it depends on the style which is used to perform the write. It may also happen that functions using TSO services will stop the recording without an OUTTRAP('OFF').

```
call outtrap('lcat.')
ADDRESS TSO 'LISTCAT LEVEL(PEJ)'
call outtrap('off')
/* listcat result is stored in stem lcat. */
```

```
do i=1 to lcat.0
   Say lcat.i
End
```

Result

```
NONVSAM ------- PEJ.BLOX
     IN-CAT --- SYS1.UCAT.TSO
NONVSAM ------- PEJ.BREXX.INST
     IN-CAT --- SYS1.UCAT.TSO
NONVSAM ------- PEJ.BREXX.INST2
     IN-CAT --- SYS1.UCAT.TSO
NONVSAM ------- PEJ.BREXX.NJE.INST2
     IN-CAT --- SYS1.UCAT.TSO
NONVSAM ------- PEJ.CMDPROC
     IN-CAT --- SYS1.UCAT.TSO
NONVSAM ------- PEJ.CNTL
     IN-CAT --- SYS1.UCAT.TSO
NONVSAM ------- PEJ.DSSLOAD.JCL
     IN-CAT --- SYS1.UCAT.TSO
…
```

ARRAYGEN

Similar to OUTTRAP, ARRAYGEN records output and places it in a source array (SARRAY). The recording is stopped with an ARRAGEN('OFF'), returning, the source array number. Where array-number receives the created array number which can be processed with the SARRAY functions. ARRAYGEN the same limitations apply as for OUTTRAP.

```
call arraygen('ON')
ADDRESS TSO 'LISTCAT LEVEL(BREXX)'
s1=arraygen('OFF')
call slist(s1)
```

Result

```
     Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   NONVSAM ------- BREXX.$FIX.LINKAPF.NJE38.XMIT
00002        IN-CAT --- SYS1.VMASTCAT
00003   NONVSAM ------- BREXX.$FIX.LINKAPF.XMIT
00004        IN-CAT --- SYS1.VMASTCAT
00005   NONVSAM ------- BREXX.$FIX.LINKLIB.NJE38.XMIT
00006        IN-CAT --- SYS1.VMASTCAT
00007   NONVSAM ------- BREXX.$FIX.LINKLIB.XMIT
00008        IN-CAT --- SYS1.VMASTCAT
00009   NONVSAM ------- BREXX.$INSTALL.MASTER.CNTL
00010        IN-CAT --- SYS1.VMASTCAT
…
```

B. Added BREXX Kernel functions and Commands

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

1. General

ABEND(user-abend-code)

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

AFTER(search-string,string)

The remaining portion of the string follows the first occurrence of the search-string within the string. If search-string is not part of the string an empty string is returned.

A2E(ascii-string) / E2A(ebcdic-string)

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

BEFORE(search-string,string)

The portion of the string that precedes the first occurrence of search-string within the string. If search-string is not part of the string an empty string is returned.

Example:

```
string='The quick brown fox jumps over the lazy dog'
say 'String                 'string
say 'Before Fox             'before('fox',string)
say 'After  Fox             'after('fox',string)
```

result:

```
String                 The quick brown fox jumps over the lazy dog
Before Fox             The quick brown
After  Fox              jumps over the lazy dog
```

BLDL(program-name)

Reports 1 if the program is callable via the active program library assignments (STEPLIB, JOBLIB, etc. DD statements). If it is not found, 0 is returned.

BASE64ENC(string)

Encodes a string or a binary string into a Base 64 encoded string. It is not an encryption process; it is, therefore, not usable for storing passwords.

BASE64DEC(base64-string)

Decodes a base64 string into a string or binary string

Example:

```
str='The quick brown fox jumps over the lazy dog'
stre=base64Enc(str)
say 'Encoded  'stre
strd=base64Dec(stre)
say 'Original "'strd'"'
say 'Decoded  "'strd'"'
```

Result:

```
Encoded  44iFQJikiYOSQIKZlqaVQIaWp0CRpJSXokCWpYWZQKOIhUCTgamoQISWhw==
Original "The quick brown fox jumps over the lazy dog"
Decoded  "The quick brown fox jumps over the lazy dog"
```

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
```

C2U(character-string)

Converts a character string into an unsigned Integer string

Example:

```
say c2d(' B5918B39'x) -1248752839
say c2u(' B5918B39'x) 3046214457
```

D2P(number,length[,fraction-digit])

D2P converts a number (integer or float) into a decimal-packed field. The created field is in binary format. The fraction digit parameter is non-essential, as the created decimal does not contain any fraction information, for symmetry reasons to the P2D function it has been added.

P2D(number,length,fraction-digit)

P2D converts a decimal-packed field (binary format) into a number.

CEIL(decimal-number)

CEIL returns the smallest integer greater or equal to the decimal number.

CONSOLE(operator-command)

Performs an operator command, but does not return any output. If you need the output for checking the result, please use the RXCONSOL function.

ENCRYPT(string,password) and
DECRYPT(string,password)
Encrypts a string or decrypts an encrypted string via a password. The encryption/decryption method is merely XOR-ing the string with the password in several rounds. This means the process is not foolproof and has not the quality of RSA encryption.

```
a10='The quick brown fox jumps over the lazy dog'
a11=encrypt(a10,"myPassword")
a12=decrypt(a11,"myPassword")
say "original  "a10
say "encrypted "c2x(a11)
say "decrypted "a12
```

```
Result
original  The quick brown fox jumps over the lazy dog
encrypted E361A8D7F001D537D0D6CDCAF9EFD83CCA00F984897FBD538AAF964CA80E2806D4310205CEFAC709C9EACB43
decrypted The quick brown fox jumps over the lazy dog
```

DEFINED('variable-name')

Tests if a 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 …
```

DUMPIT(address,dump-length)
DUMPIT displays the content at a given address of a specified length in hex format. The address must be provided in hex format; therefore, a conversion with the D2X function is required.

Example:

```
call mvscbs   /* load MVS CB functions */
call dumpit d2x(tcb()),256
```

Result:

```
0099C228 (+00000000) | 0098FA80 00000000 0099099C 0099D020 | .q.......r...r}.
0099C238 (+00000010) | 00000000 00000000 009A65F8 80000000 | ............8....
0099C248 (+00000020) | 0000FFFF 0099C020 00140908 00000000 | .....r{.........
0099C258 (+00000030) | 40D792B8 009BA1E0 002E03C0 002E0434 |  Pk....\...{....
0099C268 (+00000040) | 002E0434 002E20A8 00000085 00990A3C | .......y...e.r..
0099C278 (+00000050) | 00000002 00158000 00285308 40280F50 | ............ ..&
0099C288 (+00000060) | 00BDFC10 0029F060 402853EE 00000000 | ......0- .......
0099C298 (+00000070) | 001A20F8 00000000 00000000 009A6A18 | ...8..........¦.
0099C2A8 (+00000080) | 00000000 0099B3C8 00000000 00000000 | .....r.H........
0099C2B8 (+00000090) | 00215044 00000000 009BF548 00000000 | ..&.......5.....
0099C2C8 (+000000A0) | 009919C8 809A6010 00000000 00000000 | .r.H..-.........
0099C2D8 (+000000B0) | 00000000 0098EF54 00000000 00000000 | .....q..........
0099C2E8 (+000000C0) | 00000000 00000000 00000000 00000000 | ................
0099C2F8 (+000000D0) | 0099C350 00000000 00000000 0099B3C8 | .rC&.........r.H
0099C308 (+000000E0) | 00000000 00000000 00000000 00000000 | ................
0099C318 (+000000F0) | 80000040 00000000 0099BD10 00000000 | ... .....r......
```

DUMPVAR('variable-name')

DUMPVAR displays the content of a variable or stem-variable in hex format; the displayed length is variable-length +16 bytes. The variable name must be enclosed in quotes.
If no variable is specified, all so far allocated variables are printed.

Example:

```
v21.1='Stem Variable, item 1'
v21.2='Stem Variable, item 2'
v21.3='Stem Variable, item 3'
```

`call DumpVAR('v21.1')`

Result:

```
002C2818 (+00000000) | E2A38594 40E58199 89818293 856B4089 | Stem Variable, i
002C2828 (+00000010) | A3859440 F1000000 00000000 00000000 | tem 1...........
```

DATE([date-target-format],[date],[date-input-format])

The integrated DATE function replaces the RXDATE version stored in RXLIB. RXDATE will be available to guarantee the consistency of existing REXX scripts. It may be removed in a future release

Date **defaults to today**

Supported input formats

Base	days since 01.01.0001
JDN	days since Monday 24. November 4714 BC
UNIX	days since 1. January 1970
DEC	01-JAN-20 DEC format (Digital Equipment Corporation)

XDEC	01-JAN-2020 extended DEC format (Digital Equipment Corporation)
Julian	yyyyddd e.g. 2018257
European	dd/mm/yyyy e.g. 11/11/18
xEuropean	dd/mm/yyyy e.g. 11/11/2018, extended European (4 digits year)
German	dd.mm.yyyy e.g. 20.09.2018
USA	mm/dd/yyyy e.g. 12.31.18
xUSA	mm/dd/yyyy e.g. 12.31.2018, extended USA (4 digits year)
STANDARD	yyyymmdd e.g. 20181219
ORDERED	yyyy/mm/dd e.g. 2018/12/19
LONG	dd month-name yyyy e.g. 12 March 2018, the month is translated into month number (first 3 letters)
NORMAL	dd 3-letter-month yyyy e.g. 12 Mar 2018, the month is translated into month number
QUALIFIED	Thursday, December 17, 2020
INTERNATIONAL	date format 2020-12-01
TIME	date since 1.1.1970 in seconds

Supported output formats

Base	days since 01.01.0001
JDN	days since 24. November 4714 BC
UNIX	days since 1. January 1970
Julian	yyyyddd e.g. 2018257
Days	ddd days in this year e.g. 257
Weekday	weekday of day e.g. Monday
Century	dddd days in this century
European	dd/mm/yy e.g. 11/11/18
XEuropean	dd/mm/yyyy e.g. 11/11/2018,extended European (4 digits year)
DEC	dd/mm/yy e.g. 11-NOV-18, DEC format (Digital Equipment Corporation)
XDEC	dd/mm/yyyy e.g. 11-NOV-2018, extended DEC format (Digital Equipment Corporation)
German	dd.mm.yyyy e.g. 20.09.2018
USA	mm/dd/yyyy e.g. 12/31/18
xUSA	mm/dd/yyyy e.g. 12/31/2018, extended USA (4 digits year)
STANDARD	yyyymmdd e.g. 20181219
ORDERED	yyyy/mm/dd e.g. 2018/12/19
LONG	dd. month-name yyyy e.g. 12 March 2018
LS …….	time of day in microseconds (string format): 5 chars (digits) seconds, 6 chars, microseconds without delimiters
NORMAL	dd. month-name-short yyyy e.g. 12 Mar 2018
QUALIFIED	Thursday, December 17, 2020
INTERNATIONAL	date format 2020-12-01
TIME	date since 1.1.1970 in seconds

DATETIME([target-format],[timestamp],[input-format])

Formats a timestamp into various representations

Formats are:

T	timestamp in seconds	1615310123 (seconds since 1. January 1970)
E	timestamp European format	09/12/2020-11:41:13
U	timestamp US format	12.09.2020-11:41:13

O	Ordered Time stamp	2020/12/09-11:41:13
B	Base Time stamp	Wed Dec 09 07:40:45 2020

target-format defaults to **O**rdered
input-format defaults to **T**imestamp
timestamp defaults to today's current time

Time(option)
The TIME function supports the usual options of REXX, but some ones:

MS	Time of today in seconds.milliseconds
HS	Time of today in seconds.hundreds
US	Time of today in seconds.microseconds
CPU	used CPU time in seconds. milliseconds
LS	**time of day in microseconds, in string format, 5 chars (digits) seconds, 6 chars microseconds without delimiters**

STDATE([date-target-format],[date],[date-input-format])
Calculating the Startrek Stardate. Please see the separate document Stardate.pdf.

FILTER(string,character-table <,drop/keep>)

The filter function removes all characters defined in the character table if 'drop' is used as the filter-type. If 'keep' is specified, just those characters which are in the character table are kept.

Filter-type defaults to drop.

For example, remove 'o' and 'blank':

```
say FILTER('The quick brown fox jumps over the lazy dog',' o')
result:
Thequickbrwnfxjumpsverthelazydg
```

FLOOR(decimal-number)

FLOOR returns the smallest integer less or equal to the decimal number.

INT(decimal-number)

INT returns the integer value of a decimal number. Fraction digits are stripped off. There is no rounding in place. It's faster than saying intValue=number%1

JOBINFO()

returns jobname and additional information about the currently running job or TSO session in REXX variables, like JOB.NAME, JOB.NUMBER, STEP.NAME, PROGRAM.NAME

Example:

```
say jobinfo()
say job.name
say job.number
say job.step
say job.program

Result
PEJ
PEJ
TSU02077
ISPFTSO.ISPLOGON
IKJEFT01
```

JOIN(string,target-string[,join-table])

Join merges a string into a target-string. The merge occurs byte by byte; if the byte in target-string is defined in the join-table. The join-table consists of one or more characters, which may be overwritten. If it is in the target-string, it is replaced by the equivalent byte of the string. If it is not part of the join-table, it remains as it is. If the length of the string is greater than the target-string size is appending the target-string.

The join-table is an optional parameter and defaults to blank.

```
say JOIN('     Peter        Munich','Name=        City=')
result:
Name=Peter   City=Munich
```

LEVEL()

Level returns the current procedure level. The level of information is increased by +1 for every CALL statement or function call.

Example:

```
say 'Entering MAIN        'Level()
call proc1
say 'Returning from proc1 'Level()
return
proc1:
  say 'Entering proc1       'Level()
  call proc2
  say 'Returning from proc2 'Level()
return 0
proc2: procedure
  if level()>5 then return 4
say 'Entering proc2       'Level()
```

```
  prc=proc1()
  say 'Returning from proc1 'Level()
return 0
```

Result:

```
Entering MAIN        0
Entering proc1       1
Entering proc2       2
Entering proc1       3
Entering proc2       4
Entering proc1       5
Returning from proc2 5
Returning from proc1 4
Returning from proc2 3
Returning from proc1 2
Returning from proc2 1
Returning from proc1 0
```

ARGV(argument-number,calling-level)

Returns the argument specified by **argument-number** and the **calling-level.** With this function, you can determine calling procedure arguments in several stages.

calling-level	0	is the current procedure
	-1	is the procedure calling the current procedure
	-2	the caller of the caller …
		…
	1	is the very first procedure in the calling sequence
	2	is the second procedure
	3	…

Example:

```
RX MAIN "EUROPE"

call Sub1 "Germany", "Italy","UK"
return

sub1:
call sub2 'Munich','Rome','London'
return

sub2:
say 'argument 1 of SUB2: 'argv(1,0)
say 'argument 2 of SUB2: 'argv(2,0)
say 'argument 3 of SUB2: 'argv(3,0)

say 'argument 1 of SUB1: 'argv(1,-1)
```

```
say 'argument 2 of SUB1: 'argv(2,-1)
say 'argument 3 of SUB1: 'argv(3,-1)

say 'Calling argument 1 of main: 'argv(1,-2)
return
```

Result

```
argument 1 of SUB2: Munich
argument 2 of SUB2: Rome
argument 3 of SUB2: London
argument 1 of SUB1: Germany
argument 2 of SUB1: Italy
argument 3 of SUB1: UK
Calling argument 1 of main: EUROPE
```

LINKMVS(load-module, parms)
LINKPGM(load-module, parms)

Starts a load module. Parameters work according to standard conventions.

LISTIT('variable-prefix')

Returns the content of all variables and stem-variables starting with a specific prefix. The prefix must be enclosed in quotes. If no prefix is defined all variables are printed

Example:

```
v2='simple Variable'
v21.0=3
v21.1='Stem Variable, item 1'
v21.2='Stem Variable, item 2'
v21.3='Stem Variable, item 3'
call ListIt 'V2'
```

Output:

```
List Variables with Prefix 'V2'
-------------------------------
[0001]  "V2" => "simple Variable"
[0002]  "V21." =>
>[0001] "|.0" => "3"
>[0002] "|.1" => "Stem Variable, item 1"
>[0003] "|.2" => "Stem Variable, item 2"
>[0004] "|.3" => "Stem Variable, item 3"          .
```

LOCK('lock-string',<TEST/SHARED/EXCLUSIVE><,timeout>)

Lock-string

Locks a resource (could be any string, e.g. dataset-name>) for usage by a concurrent program (which must request the same resource). Typically it is used to keep the integrity of several datasets.

Lock modes are:

- TEST tests whether the resource is available
- SHARED shared access is wanted, other programs/tasks are also shared access granted, but no exclusive lock can be granted, while a shared lock is active
- EXCLUSIVE no other program/task can use the resource at this point.

timeout defines a maximum wait time in milliseconds to acquire the resource. If no timeout is defined the LOCK ends immediately if it couldn't be acquired.

returns 0 if the resource was locked
4 resource could not be acquired in the requested time interval

UNLOCK('lock-string')
Unlocks a previously locked resource.
0 unlock was successful
(else) unlock was not successful

MEMORY()

Determines and print the available storage junks

```
MVS Free Storage Map
--------------------------
AT ADDR  7909376    1176 KB
AT ADDR  3108864    1166 KB
Total               2342 KB
--------------------------
```

MTT(<'REFRESH'>)

Returns the content of the Master Trace Table in the stem variable **_LINE., _LINE.0** contains the number of returned trace table entries. The return code contains the number of trace table entries fetched.If -1is returned the Master Trace Table has not been changed since the last call, _LINE. remains unchanged. If the REFRESH option is used, the Trace Table will be recreated even if it has not changed.

```
Call mtt()
Do i=1 to _line.0
   Say _line.i
End
```

Ergebnis:

```
4000 08.48.56 JOB  891  $HASP395 BRXLINK  ENDED"
4000 08.48.56 JOB  891  IEF404I BRXLINK - ENDED - TIME=08.48.56"
0004 08.48.56 JOB  891  BRXLINK    ALIASES             IKJEFT01  RC= 0000"
0004 08.48.55 JOB  891  BRXLINK    LINKAUTH            IEWL      RC= 0000"
0004 08.48.53 JOB  891  BRXLINK    BRXLNK              IEWL      RC= 0004"
0004 08.48.53 JOB  891  IEFACTRT - Stepname  Procstep  Program   Retcode"
4000 08.48.51 JOB  891  IEF403I BRXLINK - STARTED - TIME=08.48.51"
4000 08.48.51 JOB  891  $HASP373 BRXLINK  STARTED - INIT  1 - CLASS A - SYS TK4-"
0200 08.48.50 JOB  891  $HASP100 BRXLINK  ON READER2"
…
```

MTTSCAN

MTTSCAN is an application that constantly analyses the Master Trace Table and passes control to the user's procedures for a registered function to perform user actions.

Example in **BREXX. V2R5M3.SAMPLE(MTTSCANT)**

In this example, the trace entries $HASP373 (LOGON)and $HASP395 (LOGOFF) are registered, and the associated call-back procedures will be called to initiate further actions.

```
/* ----------------------------------------------------------------*/
/*            + ---  REGISTER requested action                     */
/*            |           + --- action keyword in trace table      */
/*            |           |          + --- associated call back proc */
/*            Y           Y          Y                             */
call mttscan 'REGISTER','$HASP373','hasp373'
call mttscan 'REGISTER','$HASP395','hasp395'
/*            + ---  Start scanning Trace Table                   */
/*            |     + --- scan frequency in milliseconds          */
/*            Y     Y          default is 5000                    */
call mttscan 'SCAN',2000
return
/* ----------------------------------------------------------------
 * Call Back to handle $HASP373 Entries of the Trace Table: user LOGON
 *    arg(1) contains the selected line of the Trace Table
 * ----------------------------------------------------------------
 */
hasp373:
  user=word(arg(1),6)
  call console 'c u='user     You can for example cancel the user   */
  say user ' has logged on'
  say 'Trace Table entry: 'arg(1)
  say copies('-',72)
return
/* ----------------------------------------------------------------
 * Call Back to handle $HASP395 Entries of the Trace Table: user LOGOFF
 *   arg(1) contains the selected line of the Trace Table
 * ----------------------------------------------------------------
 */
hasp395:
```

```
  user=word(arg(1),6)
  say user ' has logged off'
  say 'Trace Table entry: 'arg(1)
  say copies('-',72)
return
```

RXCONSOL

RXCONSOL is an application that returns the output of a requested Console command in the stem variable CONSOLE.n

A return-code>0 means the command output could not be identified in the Master Trace Table

Example in **BREXX. V2R5M3.SAMPLE(CONSOLE)**

```
/* ------------------------------------------------------------
 *  RXCONSOL Sample: Show output of a Console command
 * ------------------------------------------------------------
 */
call rxconsol('D A,L')
say copies('-',72)
say center('Console Output of D A,L',72)
say copies('-',72)
do i=1 to console.0
   say console.i
end
```

Result

```
------------------------------------------------------------------------
                        Console Output of D A,L
------------------------------------------------------------------------
0000 08.17.13 TSU 3983  D A,L
0000 08.17.13           IEE102I 08.17.13 21.181 ACTIVITY 788
 788     00010 JOBS    00006 INITIATORS
 788    CMD1     CMD1     CMD1      V=V
 788    BSPPILOT BSPPILOT C3PO      V=V  S
 788    JES2     JES2     IEFPROC   V=V
 788    NET      NET      IEFPROC   V=V
 788    TP       TP       TCAM      V=V
 788    MF1      MF1      IEFPROC   V=V  S
 788    TSO      TSO      STEP1     V=V  S
 788    SNASOL   SNASOL   SOLICIT   V=V
 788    JRP      JRP      JRP       V=V  S
 788    NJE38    NJE38    NJEINIT   V=V
 788     00001 TIME SHARING USERS
 788     00001 ACTIVE  00040 MAX VTAM TSO USERS
 788    PEJ
```

Please note: The result of an operator command is not synchronously returned, but asynchronously assigned via the activity number (788 in the example above). In certain situations, this may fail, and then an exact match of the operator command and its output is impossible. You will then see more output than expected.

RXLIST()

Prints the currently loaded BREXX modules including their originating DSN.

The first entry is the starting REXX.

```
Loaded Rexx Modules
    REXX      Member    DDNAME    DSN
-----------------------------------------------------
  1 #RXL      RXL       SYSUEXEC PEJ.EXEC
  2 RXSORT    RXSORT    RXLIB    BREXX.RXLIB
  3 FMTLIST   FMTLIST   RXLIB    BREXX.RXLIB
  4 FSSAPI    FSSAPI    RXLIB    BREXX.RXLIB
```

NJE38CMD

NJE38CMD is an application that returns the output of a requested NJE38 command in the stem variable NJE38.n

A return-code>0 means the NJE38 command output could not be identified in the Master Trace Table

Example in **BREXX. V2R5M3.SAMPLE(NJECMD)**

```
/* ------------------------------------------------------------
 *  NJE38CMD Sample: Show available files in NJE38 inbox
 *    pass command to NJE38CMD and retrieve output
 * ------------------------------------------------------------
 */
rc=nje38CMD('NJE38 D FILES')
if rc>0 then do
   say 'Unable to pickup NJE38 results'
   return 8
end
say copies('-',72)
say center('NJE38 Spool Queue',72)
say copies('-',72)
do i=1 to nje38.0
   say nje38.i
end
```

Result

```
------------------------------------------------------------------------
                          NJE38 Spool Queue
------------------------------------------------------------------------
NJE014I  File status for node DRNBRX3A
File  Origin   Origin    Dest     Dest
 ID   Node     Userid    Node     Userid    CL  Records
0006  DRNBRX3A PEJ1      DRNBRX3A PEJ       A        50
```

```
0010  CZHETH3C FIX0MIG   DRNBRX3A MIG       A       119
Spool 00% full
```

VLIST(pattern[,"VALUES"/"NOVALUES")

VLIST scans all defined REXX-variable names for a specific pattern. This is mainly for stem-variables useful, which can have various compound components.

The pattern must be coded in the form "p1.p2.p3.p4.p5", p1, p2, p3, p4, and p5 are subpatterns that must match the stem variable name. There are up to 5 subpatterns allowed. You may use "*" as a subpattern for any variable in this position.

Example

```
ADDRESS.PEJ.CITY='Munich'
ADDRESS.MIG.CITY='Berlin'
ADDRESS.pej.pub='Hofbrauhaus'
ADDRESS.mig.pub='Steakhaus'
ADDRESS='set'
call xlist('*.*.CITY')
call xlist('ADDRESS')
call xlist('ADDRESS.*.CITY')
call xlist('ADDRESS.PEJ')
call xlist('ADDRESS.MIG')
call xlist()
exit
xlist:
say '>>> 'arg(1)
say vlist(arg(1))
return
```

Result

```
>>> *.*.CITY
ADDRESS.MIG.CITY='Berlin'
ADDRESS.PEJ.CITY='Munich'

>>> ADDRESS
ADDRESS='set'
ADDRESS.MIG.CITY='Berlin'
ADDRESS.MIG.PUB='Steakhaus'
ADDRESS.PEJ.CITY='Munich'
ADDRESS.PEJ.PUB='Hofbrauhaus'

>>> ADDRESS.*.CITY
ADDRESS.MIG.CITY='Berlin'
ADDRESS.PEJ.CITY='Munich'
```

```
>>> ADDRESS.PEJ
ADDRESS.PEJ.CITY='Munich'
ADDRESS.PEJ.PUB='Hofbrauhaus'

>>> ADDRESS.MIG
ADDRESS.MIG.CITY='Berlin'
ADDRESS.MIG.PUB='Steakhaus'

>>>
ADDRESS='set'
ADDRESS.MIG.CITY='Berlin'
ADDRESS.MIG.PUB='Steakhaus'
ADDRESS.PEJ.CITY='Munich'
ADDRESS.PEJ.PUB='Hofbrauhaus'
```

LASTWORD(string)

Returns the last word of the provided string.

PEEKS(decimal-address,length)

PEEKS returns the content (typically a string) of a main-storage address in a given length. The address must be in decimal format.
PEEKS is a shortcut of STORAGE(d2x(decimal-address),length).

PEEKA(decimal-address)

PEEKA returns an address (4 bytes) stored at a given address. The address must be in decimal format.
PEEKA is a shortcut of STORAGE(d2x(decimal-address),4).

PEEKU(decimal-address)

PEEKU returns an unsigned integer stored at the given decimal address (4 bytes). The address must be in decimal format.

RACAUTH(userid,password)

The RACFAUTH function validates the userid and password against the RAKF definitions. If both pieces of information are valid, one is returned.

RHASH(string,<slots>)

The function returns a numeric hash value of the provided string. The optional slots parameter defines the highest hash number before it restarts with 0. Slots default to 2,147,483,647

Even before reaching the maximum slot, the returned number is not necessarily unique; it may repeat (collide) for various strings. The calculation is based on a polynomial rolling hash function

ROUND(decimal-number,fraction-digits)

The function rounds a decimal number to the precision defined by fraction-digits. If the decimal number does not contain the number of fraction digits requested, it is padded with 0s.

ROTATE(string,position<,length>]

The function is a rotating substring if the requested length for the substring is not available, it takes the remaining characters from the beginning of the string. If the optional length parameter is not coded, the length of the string is used.

```
Rotate("1234567890ABCDEF",10,10)       ->      '0ABCDEF123'
Rotate("1234567890ABCDEF",1)           ->      '1234567890ABCDEF'
Rotate("1234567890ABCDEF",5)           ->      '567890ABCDEF1234'
```

PUTSMF(smf-record-type,smf-message)

Writes an SMF message of type smf-record-type. If you use a defined type with a certain structure, it must be reflected in smf-message. If necessary you can use den BREXX conversion functions (D2C, D2P, etc.) to create binary data.

SUBMIT(options[,mode])

Submits a job via the internal reader to your MVS system

Options are:

- fully qualified dataset name containing the JCL
- stem variable containing the JCL
- "*" for stack containing the JCL
- **SARRAY** string array, the mode must be the String Array number
- **LLIST** Linked List, the mode must be the Linked List number

```
submit("'pds-name(member-name)'") submit a DSN or a member in a PDS
submit('stem-variable.')          submit JCL stored in stem-variable
submit('*')                       submit JCL stored in a stack (queue)
submit('SARRAY',s1)               submit JCL stored in SARRAY s1
submit('LLIST',l1)                submit JCL stored in Linked-List l1
```

Note: The internal reader does not know your userid, therefore the &SYSUID variable will not be resolved with your userid. It also does not return any "SUBMIT" message, this can easily be achieved by a small REXX script analysing the master trace table.

SPLIT(string,stem-variable[,delimiter])

SPLIT splits a string into its words and stores them in a stem variable. The optional delimiter table defines the split character(s), which shall be used to separate the words. SPLIT returns the number of found words. Also, stem-variable.0 contains the number of words. The words are stored in the stem-variable.1, stem-variable.2, etc. It is recommended to enclose the receiving stem-variable-name in quotes.

Example:

```
Say Split('The quick brown fox jumps over the lazy dog','myStem.')
Call LISTIT
Result:
9
List all Variables
------------------
[0001]  "MYSTEM." =>
>[0001] "|.0" => "9"
>[0002] "|.1" => "The"
>[0003] "|.2" => "quick"
>[0004] "|.3" => "brown"
>[0005] "|.4" => "fox"
>[0006] "|.5" => "jumps"
>[0007] "|.6" => "over"
>[0008] "|.7" => "the"
>[0009] "|.8" => "lazy"
>[0010] "|.9" => "dog"
```

Example with a list of word delimiters:

```
say split('City=London,Address=Picadelly Circus 24(7th floor)','mystem.','()=,')
call listit
Result:
5
 List all Variables
 ------------------
 [0001]  "MYSTEM." =>
 >[0001] "|.0" => "5"
 >[0002] "|.1" => "City"
 >[0003] "|.2" => "London"
 >[0004] "|.3" => "Address"
 >[0005] "|.4" => "Picadelly Circus 24"
 >[0006] "|.5" => "7th floor"
 9
```

SPLITBS(string,stem-variable[,split-string])

SPLIT splits a string into its words and stores them in a stem variable. The split-string defines the string which shall be used to separate the words. SPLIT returns the number of found words. Also, stem-variable.0 contains the number of words. The words are stored in the stem-variable.1, stem-variable.2, etc. It is recommended to enclose the receiving stem-variable-name in quotes.

Example:

```
say splitbs('today</N>tomorrow</N>yesterday','mystem.','</N>')
call listit 'mystem.'
```

```
Result:
3
List Variables with Prefix 'MYSTEM.'
-------------------------------------
[0001]  "MYSTEM." =>
>[0001] "|.0" => "3"
>[0002] "|.1" => "today"
>[0003] "|.2" => "tomorrow"
>[0004] "|.3" => "yesterday"
```

EPOCHTIME([day,month,year])

EPOCHTIME returns the Unix (epoch) time of a given date. It's the seconds since 1. January 1970. You can easily extend the date by adding the seconds of the day.

For example

```
time= EPOCHTIME(1,1,2000)+3600*hours+60*minutes+seconds
```

As calculation internally is done on integer fields, the maximum date which is supported is **19. Januar 2038 04:14:07.** If no parameters are specified, the current date/time will be returned.

EPOCH2DATE(unix-epochtime)

EPOCH2DATE translates a Unix (epoch) time-stamp into a readable date/time format. Internally the date conversion is done by the RXDATE module of RXLIB

```
tstamp=EPOCHTIME()
say tstamp
SAY EPOCH2DATE(tstamp)
```

```
Result:
1600630022
20/09/2020 19:27:02
```

STIME()

Time since midnight in hundreds of a second

USERID()

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

UPPER(string)

UPPER returns the provided string in upper cases.

LOWER(string)

LOWER returns the provided string in lower cases.

MOD(number,divisor)
MOD divides and returns the remainder, equivalent to the // operation.

LOADRX("STEM","stemname.","procname")
LOADRX("SARRAY",array-number,"procname")

Sometimes it is useful to create a rexx procedure on the fly. For example, if you read field names from an external dataset and have to build an extraction routine. There are 2 ways to do so:

1. Create a stem containing the code line by line

```
xset.1="c=0"
xset.2="c=c+1"
xset.3="d=c+5"
xset.4="e=c+15"
xset.5="say c d e"
xset.0=5
call loadRX("STEM","XSET.","myrexx")
```

2. Create a sarray adding the lines to it.

```
s1=screate(32)
call sset(s1,,"A=0")
call sset(s1,,"A=A+1")
call sset(s1,,"A=A+1")
call sset(s1,,"A=A+1")
call sset(s1,,"A=A+1")
call sset(s1,,"say a")
call slist(s1)
xset.1="c=0"
xset.2="c=c+1"
xset.3="d=c+5"
xset.4="e=c+15"
xset.5="say c d e"
xset.0=5
s2=stem2str("xset.")
say "STEMSTR "s2
call loadRX("ARRAY",s1,"rexx2")
```

Once LOADRX is executed, the REXX-name is usable and can be called. A REXX procedure can be used just once, a reloading has no effect, as it does not overwrite an existing version.

STCSTOP()

Check in a started task (STC) if a STOP command has been sent via the console.
Rc=0 no stop was requested
Rc=1 STOP has been requested

If a STOP has been received the REXX script should terminate.

Sample STC

```
DO forever
    if stcstop()=1 then do
       CALL WTO 'STC STOP COMMAND RECEIVED'
       leave
    end
    call wait 1000
    /* do STC stuff             */
…
…
    /* loop to check STC status */
 end
 exit 0
```

VERSION(['FULL'])

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

```
SAY VERSION()          ->    V2R5M3
SAY VERSION('FULL')    ->    Version V2R5M3 Build Date 15. Jan 2021
```

WAIT(wait-time)

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

WORDDEL(string,word-to-delete)

WORDDEL removes a specific word from the string. If the specified word does not exist, the full string is returned.

Example

```
say worddel('I really love Brexx',1)
say worddel('I really love Brexx',2)
say worddel('I really love Brexx',3)
say worddel('I really love Brexx',4)
say worddel('I really love Brexx',5)
```

Result

```
really love Brexx
I love Brexx
I really Brexx
I really love
I really love Brexx
```

WORDINS(new-word,string,after-word-number)

WORDINS inserts a new word after the specified word number. If 0 is used as wobaserd number it is inserted at the beginning of the string.

Example

```
say wordins('really','I love BREXX',1)
say wordins('really','I love BREXX',2)
say wordins('really','I love BREXX',3)
say wordins('really','I love BREXX',0)
```

Result

```
I really love BREXX
I love really BREXX
I love BREXX really
really I love BREXX
```

WORDREP(new-word,string,word-to-replace)

WORDREP replace a word value by a new value.

Example

```
say wordrep('!!!','I love Brexx',1)
say wordrep('!!!','I love Brexx',2)
say wordrep('!!!','I love Brexx',3)
```

Result

```
!!! love Brexx
I !!! Brexx
I love !!!
```

WTO(console-message)

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

XPULL()

PULL function which returns the stack content case sensitive.

GETDATA([rexx-module])

The GETDATA function fetches all Data-Sections of the currently running REXX and creates either a stem, a sarray, an integer array (IARRAY) a or float array (FARRAY). The format of Data-Sections is embedded in a comment block and has the following format:

The comment which contains the data have the format:

```
/* DATA STEM stemname …
Content 1
Content 2
…
*/
```

The first line defines the target which receives the content, it can be

/* DATA STEM stemname.
/* DATA SARRAY array-variable
/* DATA IARRAY array-variable
/* DATA FARRAY array-variable

Neither of the arrays needs to be created prior to the call, they are created during the execution of the GETDATA function. It works on the current running rexx. If you have a complex and/or nested structure it is recommended to define the rexx-module as the parameter.

/* DATA STEM stemname

```
/* DATA STEM BANDS.
  LED ZEPPELIN              STAIRWAY TO HEAVEN
  EAGLES                    HOTEL CALIFORNIA
  AC/DC                     BACK IN BLACK
  JOURNEY                   DON'T STOP BELIEVIN'
  PINK FLOYD                ANOTHER BRICK IN THE WALL
  QUEEN                     BOHEMIAN RHAPSODY
  TOTO                      HOLD THE LINE
  DEEP PURPLE               SMOKE ON THE WATER
*/
```

The first comment line starts with **/* DATA STEM BANDS.** DATA defines the beginning of a data section, STEM stem-name associates a stem that will receive the data.
If you prefer a SARRAY to receive them, you can use alternatively: **/* DATA SARRAY BANDS,** in this case, a SARRAY is created and will receive the data, and the array number is stored in the specified variable (BANDS in the example). The SARRAY can be processed with the array functions.
The end of the data section is defined by a closing comment string in a separate line.

To eventually receive the data you must **call GETDATA.** GETDATA pushes all data sections of the REXX script in the requested stem or sarray.

```
call GetData

do i=1 to bands.0
   say i bands.i
end
```

Result

```
1 LED ZEPPELIN                  STAIRWAY TO HEAVEN
2 EAGLES                        HOTEL CALIFORNIA
3 AC/DC                         BACK IN BLACK
4 JOURNEY                       DON'T STOP BELIEVIN'
5 PINK FLOYD                    ANOTHER BRICK IN THE WALL
6 QUEEN                         BOHEMIAN RHAPSODY
7 TOTO                          HOLD THE LINE
8 DEEP PURPLE                   SMOKE ON THE WATER
```

LCS('string1','string2") Longest Common Subsequence

Find the Longest Common Subsequence of two strings.

```
Say LCS("thisisatest", "testing123testing")
```

Result

```
tsitest
```

2. GLOBAL Variables

You can define global variables which can be accessed from within the rexx whatever the current procedure variable scope is. STEMS are not supported.

SETG('variable-name','content')

SETG sets or updates a variable with the given content.

GETG('variable-name')

GETG returns the current content of the global variable.

Example:

```
call setg('ctime',time('l'))
call setg('city','Munich')
call testproc
exit 0
testproc: procedure
/* normal variable scope can't access variables from the calling rexx */
  say 'Global Variables from the calling REXX'
  say   getg('ctime')
  say   getg('city')
return 0
```

Result

```
Global Variables from the calling REXX
19:19:24.15
Munich
```

3. Dataset Functions

CREATE(dataset-name,allocation-information)

The CREATE function creates and catalogues a new dataset (if the user has the required authorisation level). If the dataset-name is not fully qualified, it will be prefixed by the user name.

Fully qualified DSN is: **"'BREXX.TEST.SEQ'"**
Not fully qualified: **"TEST.SEQ"** will be prefixed by user name (e.g. HERC01) **"HERC01.TEST.SQ"**

allocation-information can be:

DSORG, RECFM, BLKSIZE, LRECL, PRI, SEC, DIRBLKS, UNIT (not all are mandatory):.

The space allocations for PRI (primary space) and SEC (secondary space) are the number of tracks.

Example:

```
CREATE('TEST','recfm=fb,lrecl=80,blksize=3120,unit=sysda,pri=5,DIRBLKS=5')
```

If the create is successful, the return code will be zero; else a negative value will be returned. The CREATE function does not open the dataset.

Return codes:

0	Create was successful
-1	Dataset cannot be created (various reasons such as, space limitations, authorisation, etc.)
-2	Dataset is already catalogued

DIR(partitioned-dataset-name,['DETAILS'/'SUMMARY/MEMBERS]')

The DIR command returns the directory of a partitioned dataset. If the partitioned dataset is not fully qualified, it will be prefixed by the user name.

The second parameter defaults to **DETAILS.**

The directory is provided in the stem variable **DIRENTRY.**

If '**MEMBERS**' is specified then only then member names are returned in DIRENTRY.n.NAME.

If '**SUMMARY**' is specified then only a summary of each member is returned

DIRENTRY.0	contains the number of directory members
DIRENTRY.n NAME	member name
DIRENTRY.n.LINE	line with all available information

If '**DETAILS**' is specified all available information is returned in the stem.

DIRENTRY.0	contains the number of directory members
DIRENTRY.n.CDATE	creation date of the member, e.g. => "19-04-18"
DIRENTRY.n .INIT"	initial size of member
DIRENTRY.n.MOD"	mod level
DIRENTRY.n NAME	member name
DIRENTRY.n.SIZE"	current size of member
DIRENTRY.n.TTR	TTR of member
DIRENTRY.n.UDATE	last update date, e.g. " 20-06-09"
DIRENTRY.n.UID	last updated by user- id
DIRENTRY.n.UTIME"	last updated time
DIRENTRY.n.CDATE	creation date

n is the number of the member entry

EXISTS(dataset-name)
EXISTS(partitioned-dataset(member))

The EXISTS function checks the existence of a dataset or the presence of a member in a partitioned dataset.

EXISTS returns 1 if the dataset or the member in a partitioned dataset is available. It returns 0 if it does not exist. If the dataset-name is not fully qualified, it will be prefixed by the user name.

REMOVE(dataset-name)

The REMOVE function un-catalogues and removes the specified dataset (if the user has the required authorisation level). If dataset-name is not fully qualified, it will be prefixed by the user name.

If the removal is successful, the return code will be zero; else a negative value will be returned.

Return codes:

0	dsn successfully removed
<>0	DSN was not removed

REMOVE(partitioned-dataset(member))

The REMOVE function on members of a partitioned dataset removes the specified member (if the user has the required authorisation level). If the dataset-name is not fully qualified, it will be prefixed by the user name.

If the removal is successful, the return code will be zero; else a negative value will be returned.

RENAME(old-dataset-name,new-dataset-name)

The RENAME function renames the specified dataset. The user requires the authorisation for the dataset to rename as well as the new dataset. If the dataset-name is not fully qualified, it will be prefixed by the user name.

If the rename is successful, the return code will be zero; else a negative value will be returned.

RENAME(partitioned-dataset(old-member),partitioned-name(new-member))

The RENAME function on members renames the specified member into a new one. The user requires authorisation for the dataset. The RENAME must be performed in the same partitioned dataset.

If the rename is successful, the return code will be zero; else a negative value will be returned.

ALLOCATE(ddname,dataset-name)
ALLOCATE(ddname,partitioned-dataset(member-name))

The ALLOCATE function links an existing dataset or a member of a partitioned dataset to a dd-name, which then can be used in services requiring a dd-name. If dataset-name is not fully qualified, it will be prefixed by the user name.

If the allocation is successful, the return code will be zero; else a negative value will be returned.

FREE(ddname)

The FREE function de-allocates an existing allocation of a dd-name.

If the de-allocation is successful, the return code will be zero; else a negative value will be returned.

OPEN(dataset-name,open-option,allocation-information)

The OPEN function has now a third parameter, which allows the creation of new datasets with appropriate DCB and system definitions. If the dataset already exists, the existing definition is used, and the DCB is not updated.

If the dataset-name is not fully qualified, it will be prefixed by the user name.

The dataset-name may contain a member name, which must be enclosed within parentheses.
OPEN(“”myPDS(mymember)””)

If the open is performed with the read-option, the member name must be present, else the open fails. If the write-option is used, you can refer to a member-name that does not yet exist and will be created by following write commands. If the member name exists, the current content will be overwritten.

The open-options have not changed, please refer to the official BREXX documentation.

allocation-information can be:

DSORG, RECFM, BLKSIZE, LRECL, PRI, SEC, DIRBLKS, UNIT (not all are mandatory):.

The space allocations for PRI (primary space) and SEC (secondary space) is the number of tracks.

If the open is successful, a file handle (greater zero) will be returned; it will be less or equal zero if the open is not successful.

Important notice: opening a member of a partitioned dataset in write mode requires full control of the entire dataset (not just the member), if you edit or browse the member concurrently the open will fail.

EXECIO Command

The EXECIO is a host command; therefore, it is enclosed in apostrophes.
EXECIO performs data set I/O operations either on the stack or stem variables, it supports only dataset containing text records. For records containing binary data you can use
There is just a subset of the known EXECIO functions 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!
EXECIO <lines-to-read/*> <DISKR/DISKW/LIFOR/LIFOW/FIFOR/FIFOW>

(<STEM stem-variable-name/LIFO/FIFO> [SKIP skip-lines] [START first-stem-entry]

[KEEP keep-string] [DROP dropstring] [SUBSTR(offset,length)]

Lines-to read	is the number of records which shall be read from the file, * means read all records
DISKR	read from dataset
DISKW	write into dataset
LIFOR/FIFOR	read from stack, stack structure can't be changed, it is fixed by the ways it was created
LIFOW/FIFOW	write to stack inLIFO or FIFO way
STEM	read into a stem/write from a stem variable
first-stem-entry	start adding entries at given stem.number, only available on DISKR with STEM parameter
LIFO	read from / write into a lifo stack
FIFO	read from / write into a fifo stack
skip-lines	skip number of lines before processing dataset/stack
keep-string	process just records containing the string
drop-string	process just records which do not contain the string
SUBSTR	process a substring of the given record

```
/* 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."

/* Append File by Stem-Variable */
"EXECIO * DISKA dd-name (STEM stem-name."

/* ---- Read into REXX FIFO Stack  ------- */
"EXECIO * DISKR dd-name (FIFO "
do i=1 to queued()
   parse pull line
   say line
end
```

```
/* ---- Read into REXX LIFO Stack  ------- */
"EXECIO * DISKR dd-name (LIFO "
do i=1 to queued()
   parse pull line
   say line
end
```

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

3. TCP Functions

TCP Functions are only usable in TK4-, or an equivalent MVS3.8j installation running on SDL Hyperion with activated TCP support.

For non TK4- installation it might be necessary to start the TCP functionality in the Hercules console before the IPL of MVS is performed:

```
facility enable HERC_TCPIP_EXTENSION
facility enable HERC_TCPIP_PROB_STATE
```

for details, you look up the following document:
https://github.com/SDL-Hercules-390/hyperion/blob/master/readme/README.TCPIP.md

Important Notice: If TCP support is not enabled, the TCP environment is in an undefined state, and all subsequent TCP functions will end up with indeterminate results or even cause an ABEND.

In case of errors or ABENDs an automatic cleanup of open TCP sockets takes place. If in rare cases the BREXX cleanup cannot resolve it, then a reconnect will be rejected. You can then reset all sockets by the TSO command **RESET**.

TCPINIT()

TCPINIT initialises the TCP functionality. It is a mandatory call before using any other TCP function.

TCPSERVE(port-number)

TCPSERVE opens a TCP Server on the defined port-number for all its assigned IP-addresses.

The function returns zero if it is performed successfully, or else an error occurred.

TCPOPEN(host-ip,port-number[,time-out-secs])

Rc=TCPOPEN(host-ip,port-number[,time-out-secs]) is a Client function to open a connection to a server.

Host-ip can be an ip-address or a host-name, which translates into an ip-address. Port-number is the port in which the server listens for incoming requests. The timeout parameter defines how long the function will wait for a confirmation of the open request; the default is 5 seconds.

If rc= 0 the open was successful if less than zero an error occurred during the open process.

The BREXX variable **_FD** contains the unique token for the connection. It must be used in various subsequent TCP function calls to address the appropriate socket.

TCPWAIT([time-out-secs])

TCPWAIT is a Server function; it waits for incoming requests from a client. The optional timeout parameter defines an interval in seconds after the control is returned to the server, to perform for example some cleanup

activities, before going again in a wait. TCPWAIT returns several return codes which allow checking which action has ended the wait.

#receive	an incoming message from a client has been received
#connect	a new client requests a connect
#timeout	a time-out occurred
#close	a close request from a client occurred
#stop	a socket returned stop; typically the socket connection has been lost.
#error	an unknown error occurred in the socket processing

Example of a server TCPWAIT and how it is processed:

```
do forever
    event = tcpwait(20)
    if event <= 0 then call eventerror event
    select
        when event = #receive then do
             rc=receive()
             if rc=0 then iterate   /* proceed  */
             if rc=4 then leave     /* close client socket */
             if rc=8 then leave     /* shut down server    */
        end
        when event = #connect then call connect
        when event = #timeout then call timeout
        when event = #close then   call close
        when event = #stop  then   call close  /* is /F console cmd */
        when event = #error then   call eventError
        otherwise  call eventError
    end
end
```

TCPSEND(clientToken,message[,timeout-secs])

SendLength=TCPSEND(clientToken, message[,time-out-secs]) sends a message to a client. ClientToken specifies the unique socket of the client. The optional timeout parameter allows the maximum wait time in seconds to wait for confirmation from the client, that it has received it. The default timeout is 5 seconds.

If sendLength is less than zero, an error occurs during the sending process:

>0 message has been sent and received by the client, number of bytes transferred
-1 socket error
-2 client is not ready to receive a message

TCPReceive(clientToken,[time-out-secs])

MessageLength=TCPReceive(clientToken,[time-out-secs]) the message length is returned by the TCPRECEIVE Function,

The message itself is provided in the variable **_Data**.

If messageLength is less than zero, an error occurred during the receiving process:

>0	message has been received from, number of bytes received
-1	client is not ready to receive a message
-2	socket error

TCPTERM()

Closes all client sockets and removes the TCP functionality

TCPSF(port,[timeout],[svrname])

TCPSF is a generic TCP Server Facility. It opens a TCP server and controls all events. Call-back labels in the calling rexx support the event handling. Therefore the calling REXX-script must contain the following labels:

TCPCONNECT: There was a client connect request. The connect will be performed by the TCPSF. If you want, you can do some logging of the incoming requests.

ARG(1))		client token
Return codes from user procedure control the continuation:		
return	0	proceed
	4	immediately close client
	8	shut down server

TCPTIMEOUT There was a time-out, no user requests occurred. Typically it is used to allow some maintenance. Doing nothing (plain return 0) is also possible. If the user procedure wants to set a new time-out value, it must be set in the rexx variable NEWTIMEOUT. It is set in seconds.

There are no arguments passed.

return	0	proceed
	8	shut down server

TCPDATA client has sent a message

ARG(1))		client token
ARG(2)		contains the original message
ARG(3)		contains the message translated from ASCII to EBCDIC
Return codes from user procedure control the continuation:		
return	0	proceed
	4	immediately close client
	8	shut down server

TCPCLOSE client has closed the connection. TCPCLOSE can be used for housekeeping.

ARG(1))	client token

Return codes from user procedure control the continuation:
return 0 proceed
8 shut down server

TCPSTOP client will be stopped.
ARG(1)) client token
There is no special return code treatment

The following commands sent from a client are processed from the TCP Server:
/CANCEL shut down the TCP server
/QUIT log off the client from the TCP Server

An example of a TCP Server is defined in **BREXX. V2R5M3.SAMPLE($TCPSERV)**

4. TSO REXX Functions

TSO REXX functions are only available in TSO environments (online or batch) not in plain batch.

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

Returns a message indicating whether a dataset exists or not.

A fully qualified dataset-name must be enclosed in apostrophes (single quotes) they must be delivered to the MVS function, it is, therefore, necessary to put double quotes around the dataset-name. If the dataset-name does not contain an apostrophe, it is completed by the user-name as the prefix.

Return message:

OK	dataset or member is available
DATASET NOT FOUND	dataset or member is not available
INVALID DATASET NAME,	the 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
```

SYSALC(['DSN',dataset-name) or SYSALC(['DDN',dd-name)

A function which determines allocation information.

DSN and a **dataset-name** return all allocations in which the DSN occurs
DDN and a **dd-name** return the dataset-names which are allocated for it. If there is more than one Dataset, then the dd-name has an allocation with concatenated datasets.

The result is returned in the stem variable _RESULT.i, _RESULT.0 contains the number of entries.

```
call sysalc('DSN','Brexx.r')
call stemlist '_result.'
```

```
     Entries of STEM: _RESULT.
Entry   Data
-------------------------------------------------
00001   SYSUEXEC
00002   SYS00033
2 Entries
```

```
call sysalc('DSN','Brexy.r')        /* DSN does not exist */
```

```
call stemlist '_result.'
```

```
    Entries of STEM: _RESULT.
Entry   Data
-------------------------------------------------
0 Entries
```

```
call sysalc('DDN','SYSuexec')
call stemlist '_result.'
```

```
     Entries of STEM: _RESULT.
Entry   Data
-------------------------------------------------
00001   PEJ.EXEC
00002   BREXX.R
2 Entries
```

SYSVAR(request-type)

a TSO-only function to retrieve certain TSO runtime information.

Available request-types

SYSUID	UserID
SYSPREF	system prefix of current TSO session (typically hlq of userid)
SYSENV	FORE/BACK/BATCH foreground/background TSO execution, or plain batch
SYSISPF	ISPF active 1, not active 0
SYSTSO	TSO active 1, not active 0
SYSAUTH	script runs in authorised mode (1), 0 not authorised
SYSCP	returns the „host"-system which runs MVS38j. It is i seither MVS or VM/370
SYSCPLVL	shows the release of the „host"-system
SYSHEAP	allocated heap storage
SYSSTACK	allocated stack storage
RXINSTRC	BREXX Instruction Counter

```
say sysvar('SYSISPF')       -> ACTIVE
say sysvar('SYSUID')        -> PEJ
say sysvar('SYSPREF')       -> PEJ
say sysvar('SYSENV')        -> FORE
say sysvar('SYSAUTH')       -> 1
say sysvar('SYSCP')         -> Hercules
say sysvar('SYSCPLVL')      -> Hercules version 4.3.9999.9976-SDL-gcb24398-
                               modified (4.3.9999.9976)
say sysvar('RXINSTRC')      -> 5
```

MVSVAR(request-type)

Return certain MVS information.

SYSNAME	system name
SYSOPSYS	MVS release
CPUS	number of CPUs
CPU	CPU type
NJE	1 = NJE38 is running, 0 = NJE38 is not running/installed
NJEDSN	Dataset name of the NJE38 spool queue
SYSNETID	Netid of MVS (if any)
SYSNJVER	Version of NJE38
JOBNUMBER	current job number
JOBNAME	job name of the current execution
STEPNAME	step name of the current execution
PROGRAM	current main program running
REXX	REXX main script name
REXXDSN	REXX main script loaded from DSN
MVSUP	Time MVS is up and running (since IPL) in seconds. You can use sec2TIME() to convert it

```
Say MVSVAR(‘SYSNAME‘)       -> (TK4-)
SAY MVSVAR('SYSOPSYS')      -> MVS 03.8
SAY MVSVAR('CPU')           -> 3033
SAY MVSVAR('CPUS')          -> 0001
SAY MVSVAR('NJE')           -> 1
SAY MVSVAR('NJEDSN')        -> NJE38.NETSPOOL
SAY MVSVAR(SYSNETID)        -> DRNBRX3A
SAY MVSVAR(SYSNJVER)        -> V2.2.0 01/14/21 07.11
SAY MVSVAR('MVSUP')         -> 212885
SAY sec2time(MVSVAR('MVSUP'),'DAYS')-> 2 day(s) 11:08:05
```

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

Returns information of non-VSAM datasets in REXX variables:

SYSDSNAME	Dataset name
SYSVOLUME	Volume location
SYSDIRBLK	directory blocks of a PDS
SYSMEMBERS	number of members in a PDS
SYSMEMBER	member name if dataset-name addresses a member in a PDS (SYSDIRBLK, SYSMEMBERS will not be set)
SYSDSORG	PS for sequential, PO for partitioned datasets
SYSRECFM	record format, F,FB,V,VB, …
SYSLRECL	record length

SYSBLKSIZE	block size
SYSRECORDS	number of records (sequential files only)
SYSSIZE	real sequential file size for record format F or FB, else its equal to SYSSIZE2
SYSSIZE2	file size, reflects the sum of all records with its "real" record length, without trailing spaces.

A fully qualified dataset-name must be enclosed in apostrophes (single quotes) they must be delivered to the MVS function, it is, therefore, necessary to put double-quotes around the dataset-name. If the dataset-name does not contain an apostrophe, it is prefixed by the user-name

LISTDSIX("'"dataset-name"") or LISTDSIX('dd-name FILE')

LISTDSIX is an extended version LISTDSI, which contains some additional dataset attributes. Due to performance reasons, it has not been integrated into the standard LISTDSI. All LISTDSI variables are contained plus these additional ones, some are redundant nd are suffixed with an X:

SYSBLKSIZEX	block size (returned from extra analysis)
SYSCREATE	creation date in Julian date format
SYSDSORGX	PS for sequential, PO for partitioned datasets
SYSEXTENTS	number of extents
SYSLRECLX	record length
SYSNTRACKS	"116"
SYSRECFMX	record format, F,FB,V,VB, …
SYSREFDATE	last referenced date in Julian date format
SYSSEQALC	secondary allocation in SYSUNITS
SYSTRACKS	allocated tracks
SYSUNITS	allocation unit: CYLINDERS, TRACKS or BLOCKS

A fully qualified dataset-name must be enclosed in apostrophes (single quotes) they must be delivered to the MVS function, it is, therefore, necessary to put double quotes around the dataset-name. If the dataset-name does not contain an apostrophe, it is prefixed by the user-name

LISTVOL(volume)

Returns detailed information about the volume:

VOLVOLUME	Volume name
VOLTYPE	Volume type 3350,3360, 3390, etc.
VOLCYLS	physical cylinders
VOLTRKSCYL	tracks per Cylinders
VOLTRACKS	volume total tracks
VOLTRKALC	total tracks allocated
VOLTRKLEN	track length
VOLDIRTRK	maximum directory blocks of track

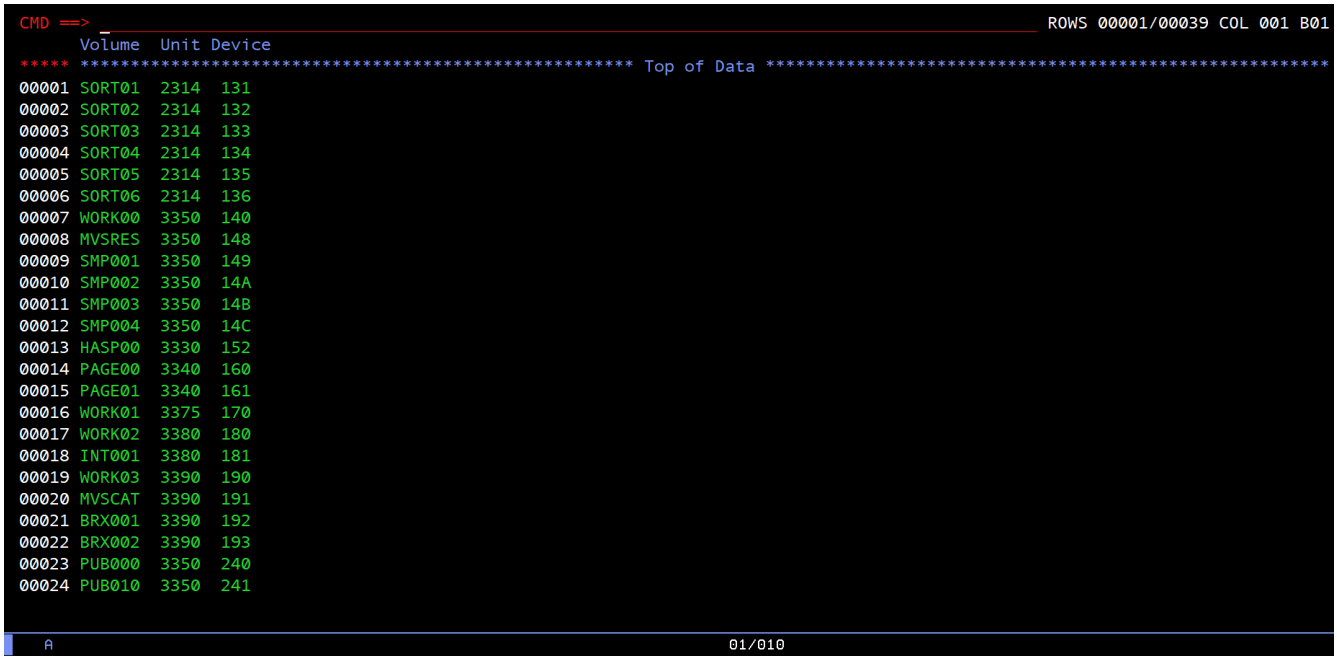
VOLDSNS	number of datasets residing on Volume
VOLTRKUSED	number of used tracks
VOLDEVICE	device number of volume, e.g. 241
VOLDSCBS	maximum number of DSCB
VOLDSCBTRK	maximum number of DSCBs in a track
VOLALTTRK	number of alternate tracks

LISTVOLS(option)

Returns a list of attached DISK Volumes. This function requests the information directly from the Hercules system and requires system administrator rights. It works only if the host system is MVS3.8.

Options can be FMTLIST, LIST, or STEM. If FMTLIST is specified the output is presented in an FMTLIST screen. LIST provides the result in the normal output device. STEM returns it in the stem VOLUMES.x.

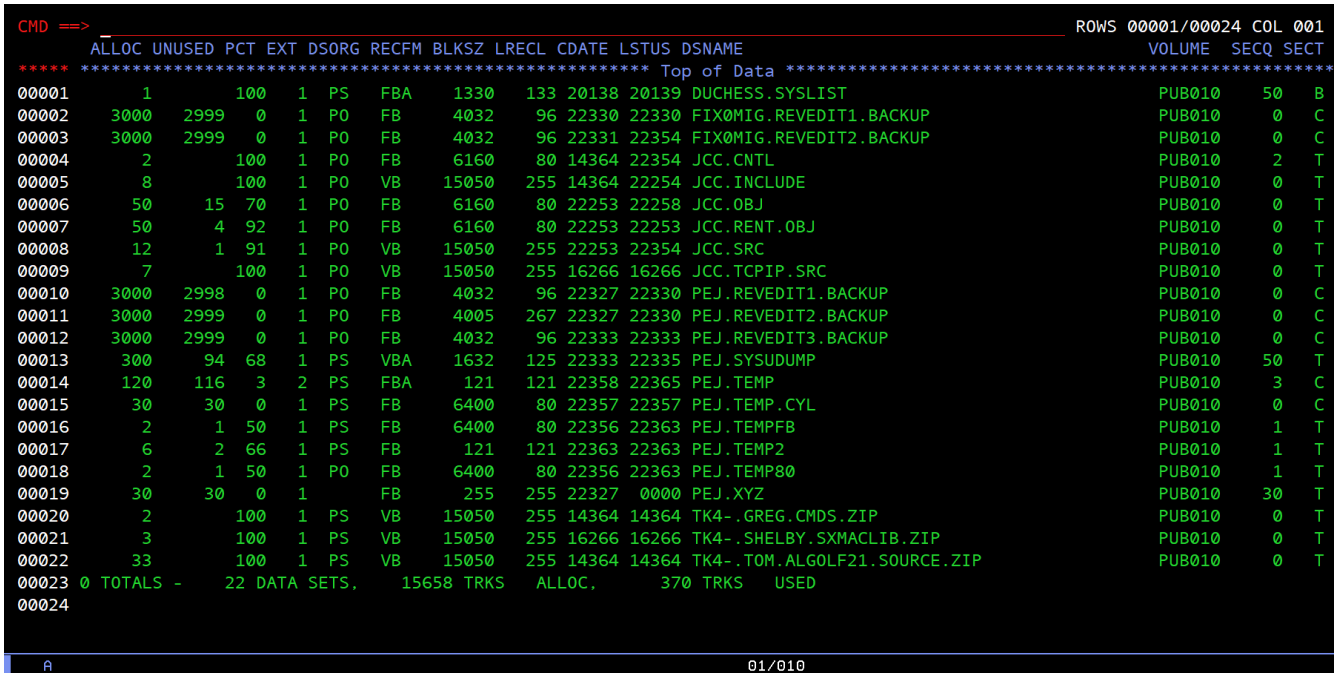
```
call listvols 'FMT'
```



VTOC(volume[,LIST/FMT/])

Produces a list of entries residing on the volume. If **LIST** is specified it is printed, **FMT** produces an FMTLIST screen displaying the content of the volume. If no option is defined the output is returned in the stem **VTOC.**

```
call vtoc 'PUB010','FMT'
```



```
CMD ==> _                                                                ROWS 00001/00024 COL 001
      ALLOC UNUSED PCT EXT DSORG RECFM BLKSZ LRECL CDATE LSTUS DSNAME                VOLUME  SECQ SECT
***** ************************************************** Top of Data ***************************************************
00001      1       100   1  PS   FBA    1330   133 20138 20139 DUCHESS.SYSLIST                PUB010    50   B
00002   3000    2999   0   1  PO   FB     4032    96 22330 22330 FIX0MIG.REVEDIT1.BACKUP        PUB010     0   C
00003   3000    2999   0   1  PO   FB     4032    96 22331 22354 FIX0MIG.REVEDIT2.BACKUP        PUB010     0   C
00004      2       100   1  PO   FB     6160    80 14364 22354 JCC.CNTL                        PUB010     2   T
00005      8       100   1  PO   VB    15050   255 14364 22254 JCC.INCLUDE                    PUB010     0   T
00006     50     15  70   1  PO   FB     6160    80 22253 22258 JCC.OBJ                        PUB010     0   T
00007     50      4  92   1  PO   FB     6160    80 22253 22253 JCC.RENT.OBJ                   PUB010     0   T
00008     12      1  91   1  PO   VB    15050   255 22253 22354 JCC.SRC                        PUB010     0   T
00009      7       100   1  PO   VB    15050   255 16266 16266 JCC.TCPIP.SRC                  PUB010     0   T
00010   3000    2998   0   1  PO   FB     4032    96 22327 22330 PEJ.REVEDIT1.BACKUP            PUB010     0   C
00011   3000    2999   0   1  PO   FB     4005   267 22327 22330 PEJ.REVEDIT2.BACKUP            PUB010     0   C
00012   3000    2999   0   1  PO   FB     4032    96 22333 22333 PEJ.REVEDIT3.BACKUP            PUB010     0   C
00013    300      94  68   1  PS   VBA    1632   125 22333 22335 PEJ.SYSUDUMP                  PUB010    50   T
00014    120     116   3   2  PS   FBA     121   121 22358 22365 PEJ.TEMP                      PUB010     3   C
00015     30      30   0   1  PS   FB     6400    80 22357 22357 PEJ.TEMP.CYL                  PUB010     0   C
00016      2       1  50   1  PS   FB     6400    80 22356 22363 PEJ.TEMPFB                    PUB010     1   T
00017      6       2  66   1  PS   FB      121   121 22363 22363 PEJ.TEMP2                     PUB010     1   T
00018      2       1  50   1  PO   FB     6400    80 22356 22363 PEJ.TEMP80                    PUB010     1   T
00019     30      30   0   1       FB      255   255 22327  0000 PEJ.XYZ                       PUB010    30   T
00020      2       100   1  PS   VB    15050   255 14364 14364 TK4-.GREG.CMDS.ZIP             PUB010     0   T
00021      3       100   1  PS   VB    15050   255 16266 16266 TK4-.SHELBY.SXMACLIB.ZIP       PUB010     0   T
00022     33       100   1  PS   VB    15050   255 14364 14364 TK4-.TOM.ALGOLF21.SOURCE.ZIP   PUB010     0   T
00023 0 TOTALS -    22 DATA SETS,    15658 TRKS   ALLOC,      370 TRKS   USED
00024
   A                                                                    01/010
```

PRINT(parameter)

Manages printing into a SYSOUT class. The page size is 60 lines, line size is 132. If a new line exceeds the page size a page break occurs and the title line is printed.

If a label $PRINT_header: is defined in your calling REXX script, it is called a call-back. Additional lines can be output there (using the PRINT command) as heading lines (appearing after each page break)

PRINT $ONTO,sysout-class	Define and open PRINT stream
PRINT <action,>line-to-print	print line (according to print action)
PRINT $TITLE,title-line	define title line, printed on a new page
PRINT $PAGE	skip to the next page, print page headers
PRINT $BANNER,text	PRINT a banner page
PRINT $CLOSE	close print stream

action:

$SKIP	add an empty line and print
$NOSKIP	print on the same line (no line feed)
$BOLD	print bold line (print it twice)

C. Set Theory Functions

SET Set theory is the branch of mathematical logic that studies sets, which can be informally described as collections of objects. Although objects of any kind can be collected into a set, set theory — as a branch of mathematics — is mostly concerned with those that are relevant to mathematics as a whole. Definition taken from Wikipedia

The following functions are supported, and described in further detail in the BREXX370 Array Functions document:

SUNIFY	sorts and unifies objects in the SARRAY. This ensures that each item is unique and eliminates duplicate entries.
SUNION	creates a new set by the union of 2 sets (arrays). Operation: **$set_1 \cup set_2$**
SINTERSECT	creates a new set by intersecting 2 sets (arrays). Operation: **$set_1 \cap set_2$**
SDIFFERENCE	creates a new set building the difference of set1 and set2, operation: **$set_1 - set_2$**
SDIFFSYM	creates a new set building the symmetrical difference of set1 and set2, operation: **$set_1 \Delta set_2$**

D. VSAM IO Functions

The VSAM IO Functionality is documented in BREXX370_VSAM_Users_Guide_V2R5M3.pdf delivered within the installation file BREXX370_V2R5M3-Final.zip

E. Formatted Screen Functions

The Formatted Screen Services is documented in BREXX370_Formatted_Screens_V2R5M3.pdf delivered within the installation file BREXX370_V2R5M3-Final.zip

F. Matrix and Integer Array functions

See new document **BREXX370 String Array functions_V2R5M3**

G. String Array functions and Linked List functions

See new document **BREXX370 String Array functions_V2R5M3**

H. RXLIB functions

BREXX can implement new functions or commands in REXX. They are transparent and are 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:

RXCOPY(source-dsn,target-dsn,[volume-name],['REPLACE'])

Copies a source dataset to a target dataset using the internal IEBCOPY or REPRO command. You can optionally define a target volume and the REPLACE option. As IEBCOPY requires an authorised mode, it can only run in ISPF environment, if it is also authorised. If not, you can run it in plain TSO command mode.

```
RXCOPY  pej.tempfb, pej.tempfb.copy,PEJ001

DSN PEJ.TEMPFB is sequential, invoke REPRO
Create 'PEJ.TEMPFB.COPY' with
DSORG=PS,RECFM=FB,UNIT=SYSDA,LRECL=80,BLKSIZE=6400,PRI=1,SEC=1,VOLSER=PEJ001
'PEJ.TEMPFB.COPY' successfully created
NUMBER OF RECORDS PROCESSED WAS 318
```

JES2QUEUE()

Returns the current content of the JES2 queue in an SARRAY

```
spool=JESQUEUE()
call slist spool
exit

     Entries of Source Array: 1
Entry   Data
-------------------------------------------------------
00001   BRXCLEAN   JOB04378  PRTPUN  ANY
00002   BRXKEYAC   JOB04326  PRTPUN  ANY
00003   BRXLINK    JOB04376  PRTPUN  ANY
00004   BRXLINK    JOB04379  PRTPUN  ANY
00005   BRXXBLD    JOB04380  PRTPUN  ANY
00006   BSPPILOT   STC01186  OUTPUT
00007   HERC01C    JOB03584  PRTPUN  ANY
00008   HERC01C    JOB03585  PRTPUN  ANY
00009   INIT       STC01187  OUTPUT
00010   INIT       STC01188  OUTPUT
00011   INIT       STC01189  OUTPUT
00012   INIT       STC01190  OUTPUT
```

```
00013   INIT        STC01191  OUTPUT
00014   INIT        STC01192  OUTPUT
00015   MFFBUILD    JOB03151  PRTPUN  HOLD
00016   MIGTEST     JOB04268  PRTPUN  ANY
00017   MVSMF       STC01140  PRTPUN  HOLD
00018   MVSMF       STC01142  PRTPUN  HOLD
00019   MVSMF       STC01147  PRTPUN  HOLD
00020   MVSMF       STC01153  PRTPUN  HOLD
00021   MVSMF       STC01173  PRTPUN  HOLD
00022   MVSMF       STC01174  PRTPUN  HOLD
00023   MVSMF       STC01199  OUTPUT
00024   MVSMF       STC01202  PRTPUN  ANY
00025   NET         STC01195  OUTPUT
00026   NJE38       STC01198  OUTPUT
00027   PEJ         TSU00860  PRTPUN  HOLD
00028   PEJ         TSU01002  OUTPUT
00029   PEJTEMP     JOB04201  PRTPUN  ANY
00030   PEJ1        TSU01303  PRTPUN  ANY
00031   PRINTPDS    JOB03250  PRTPUN  HOLD
00032   PRINTPDS    JOB04003  PRTPUN  ANY
00033   STCRX       STC01287  PRTPUN  ANY
00034   STCRX       STC01288  PRTPUN  ANY
00035   SUBTASM     JOB03150  PRTPUN  HOLD
00036   SYSLOG      STC01155  PRTPUN  HOLD
00037   SYSLOG      STC01185  OUTPUT
00038   TSO         STC01196  OUTPUT
38 Entries
```

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 includes 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.

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 initialises 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')
```

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, 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
```

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
…
```

LISTCAT(<list-cat-parameter>)

Returns listcat output in the stem LISTCAT.

LISTALL(<list-cat-parameter>)

List all datasets in the system by scanning all VTOCs.

LISTNCATL(<list-cat-parameter>)

List all not catalogued datasets in the system by scanning all VTOCs.

MVSCBS()

allows addressing of some MVS control blocks. There are several dependent control blocks combined. To use them, MVSCBS must be imported first. After that, they can be used.

Currently, integrated control blocks are:

CVT(), TCB(),ASCB(), TIOT(), JSCB(), RMCT(), ASXB(), ACEE(), ECT(), 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 the string in quotes, double quotes, or parenthesis,

Qtype can be :

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

```
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.

This function is deprecated and will be removed in a future release; please use the DIR function instead.

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

After successful completion, the stem variable.0 contains the number of lines read into the stem.

The file name can either represent an allocated dd name or a fully qualified DSN. 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

This function is deprecated and will be removed in a future release; please use the DATE function instead.

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

the 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	
UNIX	days since 1. January 1970	
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 is 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])

This function is deprecated and will be removed in a future release; please use the DUMPIT function instead.

Displays an 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([output_date_format]) or
TODAY([output_date_format[,date[,input_date_format]]) [date-format])

Returns today's date based on the requested format. You can also use a date which is in the past or the future. 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
```

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 DSN. The third parameter defines the file type and is either DSN or DDN. If it is missing DDN is the default.

I. Building TSO Commands

Any BREXX function may be turned into a TSO command by building a clist with a call to the BREXX script. The BREXX installation includes some sample CLISTs.V2R5M3.CMDLIB.

To run the specified CLISTs, place them in one of the pre-allocated clist libraries that are active in your TSO session; alternatively, use SYS2.CMDPROC. Once this is completed, you may call it straight from TSO.

1 LA List all allocated Libraries

The clist calls the BREXX LISTALC script with a BREXX CALL statement. A minus sign immediately following the REXX command tells BREXX to interpret a BREXX statement. The statement(s) must be coded in one line. To place more than one BREXX statement in a line, separate them by using a semicolon ';'.

```
REXX -                    +
CALL LISTALC('PRINT')
```

2 WHOAMI Display the current User Id

This one-liner outputs the userid() function by a say statement.

```
REXX -
SAY USERID()
```

3 TODAY Display today's Date

```
REXX -
SAY DATE(); SAY TIME()
```

4 USERS List Active Users

The clist calls the BREXX WHO script directly, therefore no minus sign is necessary:

```
REXX WHO
```

5 REPL Interactive REXX Processor

The clist calls the BREXX REPL which opens the interactive REXX processor. It allows you to enter and execute rexx statements.

```
RX REPL NOSTAE
```

6 EOT Interactive REXX Processor (Case Sensitive)

The clist calls the BREXX EOT which opens the interactive REXX processor. It allows you to enter and execute rexx statements.

```
RX EOT NOSTAE
```

Table of contents