BREXX/370 User's Guide

version V2R5M3

May 04, 2024

Contents

BREXX/370 User's Guide

Installation Guide

Introduction

This document covers the installation process of BREXX/370.

BREXX/370 is provided as-is, please test carefully in test systems only!

BREXX/370 is not the same as IBM's REXX; there are many similarities, but also differences, especially when using MVS-specific functions.

Rob Prins' TK5 Update 3 includes an installed version of BREXX/370.

Prerequisites

MVS TK4- / MVS/CE

This version of BREXX/370 has been developed and tested within Jürgen Winkelmann's TK4- (https://wotho.ethz.ch/tk4-/). It also comes pre-installed in MVS/CE (https://github.com/MVS-sysgen/sysgen). It may work in other versions of MVS, such as https://www.jaymoseley.com/hercules/installMVS/iSYSGENv7.htm but can't be guaranteed.

MVS TK5 (update 3)

This version of BREXX/370 was tested on Rob Prins' TK5. It will be included in TK5 (update 03). Therefore, you can skip the installation process unless you update your BREXX installation with a fix release.

Other MVS distributions

It should work with other MVS distributions, but this cannot be verified. Users should be aware of the following differences:

XMIT RECEIVE STEPLIB DD Statement

It might be necessary to add a STEPLIB DD statement to locate the library containing the RECV370:

```
1 //RECV370  EXEC PGM=RECV370
2 //STEPLIB  DD   DSN=library
```

Please add it to the Jobs where needed.

In TK4- and TK5 RECV370 is contained in a system library; therefore, a *STEPLIB DD* statement is not needed! On MVS/CE RECV370 is located in *SYSC.LINKLIB*.

REGION SIZE

It may be necessary to lower the REGION size parameter to 4 MB or 6 MB, as MVS may reject the REGION=8192K argument with the warning *REGION UNAVAILABLE, ERROR CODE=20*. TK4-, MVS/CE, and TK5 support 8MB region sizes.

```
1 //stepname EXEC PGM=xxxxx,REGION=6144K
```

TSO fullscreen support

BREXX/370 supports and operates effectively in the following TSO full-screen environments:

- Wally McLaughlin's version of ISPF
- Greg Price's REVIEW and RFE (Review Front End)

- Rob Prins' RPF environment

Recommendations

We recommend testing BREXX/370 in an isolated test system to avoid any impact on your current system. To achieve this, you can easily copy the entire Hercules/MVS directory to another location and install BREXX/370 there.

Preparation of your target MVS38J System

BREXX Catalogue

In TK5, the required Catalogue entry is already defined in the distribution and can therefore be omitted.

Make sure that your MVS system has a BREXX Alias pointing to a user catalogue defined in the master catalogue. To determine it, run the command:

```
listcat entries('brexx') all
```

The result must look like this:

```
ALIAS --------- BREXX
     IN-CAT --- SYS1.VSAM.MASTER.CATALOG
     HISTORY
       RELEASE----------------2
     ASSOCIATIONS
       USERCAT--UCPUB001
```

If the BREXX Alias is not defined, add it:

```
1 //ADDBREXX EXEC PGM=IDCAMS
2 //SYSPRINT DD SYSOUT=*
3 //SYSIN    DD *
4   DEFINE ALIAS (NAME(BREXX) RELATE(your-user-catalog))
```

If the submitted job is not running, it might be necessary to enter the password of the master-catalogue in the MVS console (in TK4- not needed).

If you omit this step, all BREXX data sets are catalogued in the Master Catalog. In this case, it may require the use of the Master Catalog password during the catalogue process. If you are running TK4- you do not see such requests as RAKF is providing the access authorisation of the Master Catalog, which therefore is not password protected. In the default TK4- configuration, only users HERC01 and HERC02 are authorised to update the master catalogue.

Important

All JCLs in the installation and sample library contain now a *NOTIFY=&SYSUID* parameter in the JOB card. If the patch, to resolve it during the Submit process by the current user-id, is not applied, you need to change &SYSUID to your userid, or remove it from the JOB card!

The patch can be found on: http://prycroft6.com.au/vs2mods/index.html#zp60034

This patch is already applied in TK5 and MVS/CE!

Make sure that dataset *BREXX.V2R5M3.INSTALL* is not already catalogued from a previous run. It is the recommended dataset name and will be created during the receiving process of RECV370.

Important

If a previous version of this dataset name is still catalogued, the new version ends up as not catalogued: with a *NOT CATLG 2* message! The Job output does not reveal by a ccod. Any later job which is accessing *BREXX.V2R5M3.INSTALL* will use the old version of the dataset.

Installation

Step 0 - Unzip BREXX/370 Installation File

The ZIP installation file consists of several files:

- BREXX370_Users_guide.pdf - This user guide
- BREXX370_RELEASE.XMIT - XMIT File containing BREXX modules and Installation JCL

Step 1 - Upload XMIT File

Use the appropriate upload facility of your terminal emulation. Such as IND$FILE or using *rdrprep* and inline JCL.

The file created during upload must have *RECFM FB* and *LRECL 80*. If the DCB does not match, the subsequent unpacking process fails.

Step 2 - Unpack XMIT File

Unpack the XMIT file with an appropriate JCL. If you don't have one you can use the following sample, just cut and paste it in one of your JCL libraries.

```
 1 //BRXXREC JOB 'XMIT RECEIVE',CLASS=A,MSGCLASS=H
 2 //* -------------------------------------------------------------
 3 //* RECEIVE XMIT FILE AND CREATE DSN OR PDS
 4 //* -------------------------------------------------------------
 5 //RECV370  EXEC PGM=RECV370,REGION=8192K
 6 //RECVLOG  DD SYSOUT=*
 7 //XMITIN   DD DSN=HERC01.BREXX.V2R5M3.XMIT,DISP=SHR
 8 //SYSPRINT DD SYSOUT=*
 9 //SYSUT1   DD DSN=&&XMIT2,
10 //          UNIT=3390,
11 //          SPACE=(TRK,(300,60)),
12 //          DISP=(NEW,DELETE,DELETE)
13 //SYSUT2   DD DSN=BREXX.V2R5M3.INSTALL,
14 //          UNIT=3390,
15 //          SPACE=(TRK,(300,60,20)),
16 //          DISP=(NEW,CATLG,CATLG)
17 //SYSIN DD DUMMY
```

- **HERC01.UPLOAD.XMIT** represents the uploaded XMIT File - please change it accordingly to the name you have chosen during the upload process.
- **BREXX.V2R5M3.INSTALL** is the name of the unpacked library (created during the UNPACK process). It is recommendable to remain with this DSN as it is used in later processes. **Make sure there is no previous version of this PDS catalogued.**

Important

If you use a different JCL to unpack the XMIT file, use *UNIT=3390* in the JCL. The unit type 3390 was the only reliably UNIT that ran in all tested TK4- environments. Other units may sometimes lead to various errors during the unpacking process.

Once the submitted job has successfully unpacked the XMIT file into the target PDS, you can proceed with STEP 3. The created library *BREXX.V2R5M3.INSTALL* contains all JCL to pursue with unpacking and installing.

The next steps make usage of the unpacked library (in this example *BREXX.V2R5M3.INSTALL*)

Please run the JCL in the given order (refer to the **Step x** reference in the table). Submit Step 3 as the first JCL of the installation sequence. Entries without a Step reference are used from the JCLs as input datasets.

Filename	Description	Used in Step

$CLEANUP	Cleanup: Remove unnecessary installation files	-> Step 7
$INSTALL	Install BREXX/370	-> Step 4
$CREKEYV	Create the Key/Value Database (optional)	-> Step 6
$README	Read me file	
$TESTRX	Test job to verify the BREXX/370 installation	-> Step 5
$UNPACK	Unpack subsequent libraries	-> Step 3
BUILD	Contains BREXX/370 Version and date and XMIT date	
CMDLIB	xmit packed command proc	
SAMPLES	xmit packed BREXX commands	
JCL	xmit packed example JCL	
LINKLIB	xmit packed BREXX Load library	
PROCLIB	xmit packed BREXX JCL procedures	
RXINSTDL	Internal CLIST used during Installation	
RXLIB	xmit packed include library	

Activating the new BREXX Release

The next steps describe how to enable your new BREXX Release. In summary, you must run the following jobs out of the above library in the listed sequence:

- **$UNPACK** - mandatory
- **$INSTALL** - mandatory
- **$TESTRX** - optional, recommended
- **$CREKEYV** - optional, recommended
- **$CLEANUP** - optional

See details in the step descriptions below.

Step 3 - Submit $UNPACK JCL of the unpacked Library

In the unpacking process, the contained installation files will be expanded into different partitioned datasets.

Important

Before submitting the *$UNPACK* JCL, the XMITLIB parameter must match the dataset name used in the expand JCL of Step2.

If you followed the dataset naming recommendations it is: **BREXX.V2R5M3.INSTALL** and no change is required.

```
 1 //BRXXUNP JOB 'XMIT UNPACK',CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID
 2 //*
 3 //* ---------------------------------------------------------------
 4 //* UNPACK XMIT FILES INTO INSTALL LIBRARIES
 5 //*   *** CHANGE XMITLIB= TO THE EXPANDED XMIT LIBRARY OF INSTALLATION
 6 //* ---------------------------------------------------------------
 7 //*           ---->   CHANGE XMITLIB TO YOUR UNPACKED XMIT FILE  <----
 8 //*                         XXXXXXXXXXX
 9 //*                        X     X     X
10 //*                       X      X      X
11 //*                      X       X       X
12 //*                     X        X        X
```

```
13 //XMITLOAD PROC XMITLIB='BREXX.V2R5M3.INSTALL',
14 //         HLQ='BREXX.V2R5M3',     <-- DO NOT CHANGE HLQ ----
15 //         MEMBER=
```

Important

If the job does not run and waits, check with option 3.8, the status. It is most likely "WAITING FOR DATASETS". The simplest method to resolve this is to LOGOFF and re-LOGON to your TSO session.

After completion of the *$UNPACK* JCL the following new Libraries are available:

The unpacking process removes any old version of the above libraries, before the creation of the new version. If no old version of these libraries is available, the delete steps end with *RC=4*, as well as the job ends with *RC=4*. **Ignore these errors**, if the individual unpack steps return with *RC=0*. Therefore please carefully check the output of this job.

Important

Before you install BREXX, you must decide either on the normal BREXX installation or the authorised BREXX installation.

With the authorised version you can call from BREXX utilities as IEBGENER, IEBCOPY, NJE38, etc. which run in authorised mode. This requires that the environment in which you start BREXX is authorised, meaning Wally Mclaughlin's ISPF, or RFE must be authorised. Plain TSO is already authorised.

Both installations are copied into the same partitioned datasets; they are, therefore, mutually exclusive!

If the standard installation is sufficient, continue with **Step 4** If you plan to use the authorised, continue with **Step 4A**. In this case, the MVS authorisation table needs to be updated as well.

Step 4 - Submit $INSTALL JCL for the Standard Installation

The **$INSTALL** JCL copies all member from the following two partitioned datasets into the appropriate SYS2 datasets.

- BREXX.V2R5M3.LINKLIB -> SYS2.LINKLIB
- BREXX.V2R5M3.PROCLIB -> SYS2.PROCLIB

All these members are BREXX/370 specific and do not conflict with existing members. Members of the system libraries remain untouched.

Important

Please log off and re-login to your TSO session before performing any online testing; this enforces the new loading of modules used during the testing, else you might see an 0C4. In rare situations, the installation of the BREXX Linklib members may create a new dataset extent in SYS2.LINKLIB. In this case, you must also restart your TK4- MVS session.

Continue with STEP 5

Step 4A- Submit $INSTAPF JCL for the Authorised Installation

The *$INSTPAPF* JCL copies all member from the following two partitioned datasets into the appropriate SYS2 datasets.

- BREXX.V2R5M3.LINKLIB -> SYS2.LINKLIB
- BREXX.V2R5M3.PROCLIB -> SYS2.PROCLIB

All these members are BREXX/370 specific and do not conflict with existing members. Members of the system libraries remain untouched.

To authorise the Modules to change the following Modules:

```
SYS1.UMODSRC(IKJEFTE2)
SYS1.UMODSRC(IKJEFTE8)
```

Add the BREXX modules to the sources:

```
1       DC     C'BREXX    '              BREXX/370
2       DC     C'REXX     '              BREXX/370
3       DC     C'RX       '              BREXX/370
```

To activate the changes submit the Jobs:

- SYS1.UMODCNTL(ZUM0001)
- SYS1.UMODCNTL(ZUM0014)

Aftewards you **must** restart your MVS:

- Shut down your MVS
- Re-IPL your job with the CLPA option
- Shut Down MVS again
- Perform normal IPL

Important

If you run Wally McLaughlin's ISPF the ISPF libraries must be authorised, otherwise calling a rexx from within ISPF will abend (usually S306).

Step 5 - Submit $TESTRX JCL of the unpacked Library

Submit *$TESTRX* start a test to verify the installation of BREXX/370. All steps should return with RC=0

Step 6 - Create the Key/Value Database (optional)

If you want to use Key/Value function in BREXX, modify and submit *$CREKEYV*. This JCL contains two CREATE CLUSTER definitions for the required VSAM datasets. As the K/V Database may contain data from many applications, it is generously sized, adjust it to a size which suits you. It is recommended to allocate it on a separate VOLUME, but not mandatory, insert the one chosen one.

Once it is created you can use it with the Key/Value functions.

Important

Do not run the JCL if you have already a K/V Database in place, else it will delete all your existing data.

Key/Value Database:

```
DEFINE CLUSTER                          -
         (NAME(BREXX.KEYVALUE)          -
          INDEXED                       -
          KEYS(44 0)                    -
          RECORDSIZE(64 8192)           -
          SHAREOPTIONS(2,3)             -
          CYLINDERS(600 50)             -
          VOLUMES(XXXXXX)               -
          UNIQUE                        -
```

```
          SPEED)                          -
       DATA                               -
          (NAME(BREXX.KEYVALUE.DATA))     -
       INDEX                              -
          (NAME(BREXX.KEYVALUE.INDEX))
```

Reference Database:

```
DEFINE CLUSTER                            -
         (NAME(BREXX.KEYREFS)             -
          INDEXED                         -
          KEYS(105 0)                     -
          RECORDSIZE(128 512)             -
          SHAREOPTIONS(2,3)               -
          CYLINDERS(250 50)               -
          VOLUMES(XXXXXX)                 -
          UNIQUE                          -
          SPEED)                          -
       DATA                               -
          (NAME(BREXX.KEYREFS.DATA))      -
       INDEX                              -
          (NAME(BREXX.KEYREFS.INDEX))
```

Step 7 - Submit $CLEANUP JCL of the unpacked Library

The $CLEANUP job removes all unnecessary installation files they are no longer needed, as they were merged into the appropriate SYS2.xxx library.

- BREXX.V2R5M3.LINKLIB
- BREXX.V2R5M3.PROCLIB

You may also wish to remove the uploaded XMIT File, which was used for the first unpack process.

Step 8 - ADD BREXX Libraries into TSO Logon

To run BREXX with its shortcut RX, REXX, BREXX you must allocate the BREXX libraries into your Logon procedure. There are several ways to achieve this. The easiest and recommended method for TK4 users is to add lines into *SYS1.CMDPROC(USRLOGON)*. Non TK4 installation may use different libraries. MVS/CE and Jay Moseley sysgen use *SYS1.CMDPROC(TSOLOGON)*.

```
 1 /* ALLOCATE RXLIB IF PRESENT */
 2 IF &SYSDSN('BREXX.V2R5M3.RXLIB') EQ &STR(OK) THEN DO
 3   FREE FILE(RXLIB)
 4   ALLOC FILE(RXLIB) +
 5     DSN('BREXX.V2R5M3.RXLIB') SHR
 6
 7 /* ALLOCATE SYSEXEC TO SYS2 EXEC */
 8 IF &SYSDSN('SYS2.EXEC') EQ &STR(OK) THEN DO
 9  FREE FILE(SYSEXEC)
10  ALLOC FILE(SYSEXEC) DSN('SYS2.EXEC') SHR
11 END
12
13 /* ALLOCATE SYSUEXEC TO USER EXECS */
14 IF &SYSDSN('&SYSUID..EXEC') EQ &STR(OK) THEN DO
15  FREE FILE(SYSUEXEC)
16  ALLOC FILE(SYSUEXEC) DSN('&SYSUID..EXEC') SHR
17 END
```

insert the clist above before the line *%STDLOGON* in *SYS1.CMDPROC(USRLOGON)*.

The update of the TSO Logon CLIST is an entirely manual process! Please take a backup of *USRLOGON* / *TSOLOGON* CLIST first to allow a recovery in case of errors!

Using the CLISTs as plain commands, you can either copy them into the user clist or allocate BREXX.V2R5M3.CMDLIB in the appropriate TSO start clist. This may be accomplished in TK4, MVS/CE and TK5 by including the following part in *SYS1.CMDPROC(USRLOGON)* / *SYS1.CMDPROC(TSOLOGON)*:

```
FREE FILE(SYSPROC)
ALLOC FILE(SYSPROC) +
  DSN('&SYSUID..CMDPROC','SYS1.CMDPROC','SYS2.CMDPROC', -
      'SYS2.REVIEW.CLIB','BREXX.V2R5M3.CMDLIB') SHR
```

Important

Users who upgrade from a previous release of BREXX need to update the logon clist and replace the RXLIB allocation with the current dataset name: BREXX.V2R5M3.RXLIB.

Step 9 - Your Tests

It is advised to LOGOFF and LOGON again to your system to make sure that the newly installed modules become active.

Now it's your turn to test BREXX/370! Please be advised BREXX/370 is not z/OS REXX, so you might miss some functions but find also functions not available in the "original".

Step 9 - Remove old BREXX Libraries (optional)

If you had a previous BREXX/370 version installed and your tests ran successfully, you can remove the libraries of the earlier BREXX version, for example, V2R2M0.

If you upgraded from the very first BREXX/370 version, you can remove the following libraries:

Dataset	Description
BREXX.CMDLIB	REXX commands
BREXX.SAMPLE	REXX Samples scripts
BREXX.JCL	REXX Job Control
BREXX.LINKLIB	BREXX Load Modules
BREXX.PROCLIB	BREXX JCL Procedures
BREXX.RXLIB	BREXX include Libraries

Additional Settings (optional)

If you want to communicate with the control program of the host system (either Hercules or VM) you can do so, by running:

```
ADDRESS COMMAND 'CP cp-parameter ...'
```

For VM you need to use a valid CP command. Example:

```
ADDRESS COMMAND 'CP QUERY TIME'
```

If your system is running within Hercules your CP commands are routed to Hercules and need to be Hercules commands. Example:

```
ADDRESS COMMAND 'CP DEVLIST'
```

To communicate with Hercules you need to enable the DIAG8 commands *DIAG8CMD ENABLE* in the Hercules console. In TK4-, TK5, and MVS/CE systems it is already enabled. If it is not enabled and you run an *ADDRESS COMMAND "CP command"* BREXX will abend typically with an 0C6.

Useful functions

There are JCL Procedures delivered, which facilitate the test and execution of REXX scripts. The installation process merges them into *SYS2.PROCLIB*.

The delivered RXLIB PDS contains several REXX functions, which are usable as if they were a BREXX internal function.

The delivered JCL procedures allocate the RXLIB library, and it is recommended to add it also into the TSO Logon procedures (Step 8).

TSO online

Executing rexx scripts in TSO uses either *RX* or *REXX*. You can either call scripts from dataset libraries or fully qualified dataset names.

To call a script from a library:

```
RX rexx-script-name
REXX rexx-script-name
```

BREXX performs all necessary allocations. It is advised to add a user-specific REXX library, naming convention: *&SYSUID.EXEC* (RECFM=VB, LRECL255). If available, the REXX-script searches path starts from there. The REXX library search sequence is:

1. SYSUEXEC - typically &SYSUID.EXEC
2. SYSUPROC - (optional)
3. SYSEXEC - (optional)
4. SYSPROC - (optional)

At least one of these libraries needs to be pre-allocated during the TSO logon process. 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. If you followed the instructions above then SYSEXEC is assigned to *SYS2.EXEC* and SYSUEXEC is assigned to *&SYSUID.EXEC*.

Alternatively, you can specify a fully qualified dataset-name and member name (if the dataset is a PDS):

```
RX 'dataset-name(rexx-script-name)'
REXX 'dataset(rexx-script-name)'
```

TSO Batch (start REXX JCL Procedure)

There is a JCL Procedure defined that allows you to run REXX Scripts in a TSO Batch environment. The Procedure performs all necessary BREXX and TSO allocations.

Some ADDRESS TSO commands as ALLOC/FREE are supported.

```
1 //DATETEST JOB CLASS=A,MSGCLASS=H,REGION=8192K,NOTIFY=&SYSUID
2 //*
3 //* ------------------------------------------------------------------*
4 //* TEST REXX DATE AS TSO BATCH
5 //* ------------------------------------------------------------------*
6 //REXX EXEC RXTSO,EXEC='DATE#T',SLIB='BREXX.V2R5M3.SAMPLES'
```

- **EXEC=** defines the rexx script to run
- **SLIB=** defines the library/partitioned dataset containing the rexx script defined in **EXEC**

Additionally, you can add a *P='input-parameters'* JCL Parameter field, if your rexx receives input parameters.

TSO CLISTs

To use the Clists of BREXX.V2R5M3.CMDLIB without an EXEC command, the library must be allocated to TSO, alternatively, you can copy the members to an allocated library (e.g. *SYS2.CMDPROC*)-

Plain Batch (start REXX JCL Procedure)

There is a JCL Procedure defined that allows you to run REXX Scripts in a plain Batch environment. The Procedure performs all necessary BREXX allocations

> **Warning**
>
> **ADDRESS TSO** commands are not supported here!

```
1 //DATETEST JOB CLASS=A,MSGCLASS=H,REGION=8192K,NOTIFY=&SYSUID
2 //*
3 //* ---------------------------------------------------------------*
4 //* TEST REXX DATE AS TSO BATCH
5 //* ---------------------------------------------------------------*
6 //REXX EXEC RXBATCH,EXEC='ETIME#T',SLIB='BREXX.SAMPLES'
```

- **EXEC=** defines the rexx script to run
- **SLIB=** defines the library/partitioned dataset containing the rexx script defined in **EXEC**

Additionally, you can add a *P='input-parameters'* JCL Parameter field, if your rexx receives input parameters.

BREXX/370 Sample Library

The Library *BREXX.version.SAMPLES* contains a variety of REXX scripts that cover the following areas:

- Basic functionality in Members starting with '$'
- FSS samples, starting with '#'
- VSAM samples beginning with '@'
- All other scripts are original samples delivered with Vasilis Vlachoudis BREXX installation.

BREXX/370 Hints

- Make sure your REXX files do not have line numbers! They are not wiped away by BREXX/370 and are thus considered script content. This causes mistakes during interpretation, and occasionally even system abends! To disable line numbering and delete existing numbers, use UNNUM as a primary command in the RFE or RPF Editor.
- If the BREXX/370 call leads to an S106 Abend, the most likely reason is the creation of a new extent in SYS2.LINKLIB during the installation process. Its size and number of extents are loaded during IPL and kept while MVS is up and running. The creation of new extents will therefore not be discovered. - You can either re-IPL your system or better - REORG SYS2.LINKLIB with IEBCOPY

Tokens and Terms

REXX expressions and instructions may contain the following items:

comment

A comment is a sequence of characters (on one or more lines) delimited by /* and */. Nested comments are also valid, as */* hello /* joe */ */*

string

A string is a sequence of characters delimited by a single quote or double quote. Use two quotes to obtain one quote inside a string. A string may be specified in binary or hexadecimal if the final quote is followed by a B or X. If it followed by an H then is treated as a hexadecimal number. Some valid strings are:

```
"Marmita"
'0100 0001'b
'He''s here'
'2ed3'x
'10'h (=16)
```

number

A number is a string of decimal digits with or without a decimal point. A number may be prefixed with a plus or minus sign, and/or written in exponential notation. Some valid numbers are:

```
23
12.07
141
12.2e6
+5
'-3.14'
```

symbol

A symbol refers to any group of characters from the following selection: *A-Z, a-z, 0-9, @ # $ _ . ? !*

Symbols are always translated to uppercase. Variables are symbols but the first character must not be a digit 0-9 or a dot '.'. Each symbol may consist of up to 250 characters.

function-call

A function-call invokes an internal, external, or built-in routine with 0 to 10 argument strings. The called routine returns a character string. A function-call has the format:

function-name([expr][,[expr]…)

function-name must be adjacent to the left parenthesis, and may be a symbol or a string.

All procedures can be called as functions or procedures. If a function is called as a procedure CALL left 'Hello',4 then the return string will be returned in the variable RESULT (where in this example will contain the string 'Hell')

copies('ab',3) / will return 'ababab' */*

Expressions

Most REXX instructions permit the use of expressions, following normal algebraic style. Expressions can consists of strings, symbols, or functions calls. Expressions are evaluated from left to right, modified by parentheses and the priority of the operators (as ordered below). Parentheses may be used to change the order of evaluation.

All operations (except prefix operations) act on two items and result in a character string.

Prefix + - ^ \

Prefix operations: Plus; Minus; and Not. (For + and -, the item must evaluate to a number; for ^ and \, the item must evaluate to "1" or "0".)

-4 / -4 */*

**

Exponentiate. (Both items must evaluate to numbers, and the right-hand item must be a whole number.)

```
2 ** 3  /* 8 */
2 ** -3         /* 0.125 */
```

* / % //

Multiply; Divide; Integer Divide; Divide and return the remainder. (Both items must evaluate to numbers.)

```
4 * 3   /* 12 */
4 / 3   /* 1.333.. */
5 % 3   /* 1 */
5 // 3  /* 2 */
```

+ -

Add; Subtract. (Both items must evaluate to numbers.)

```
2 + 3.02        /* 5.02 */
```

(blank) ||

Concatenate: with or without a blank. Abuttal of items causes direct concatenation.

```
a = 'One'
a 'two'         /* "One two" */
a || 'two'      /* "Onetwo" */
a'two'  /* "Onetwo" */
```

= > < >= <= ^= /= = ^> ^< >< <> == >> << >>= <<= ^== /== == ^>> ^<<

Comparisons (arithmetic compare if both items evaluate to a number.) The ==, >>, << etc. operators checks for an exact match.

```
'marmita' = ' marmita '         /* 1 (spaces are striped) */
'marmita' == ' marmita '        /* 0 */
'marmita' ^= ' marmita '        /* 0 (spaces are striped) */
'marmita' ^==' marmita '        /* 1 */
'2' = ' 2 '     /* 1 (arithmetic comparison) */
'2' == ' 2 '    /* 0 (string comparison) */
'2' >> ' 2 '    /* 1 (string comparison) */
```

&

Logical And. (Both items must evaluate to “0” or “1”.)

```
'a'='b' & 'c'='c'       /* 0 */
```

/ &&

Logical Or; Logical Exclusive Or. (Both items must evaluate to “0” or “1”.)

```
'a'='b' | 'c'='c'       /* 1 */
'A'='b' && 'c'='c'      /* 1 */
```

In Ansi REXX the results of arithmetic operations are rounded according the setting to *NUMERIC DIGITS* (default is 9). Here all arithmetic operations follow C arithmetics. For a more detail description look at the Implementation Restrictions document.

Instructions

Each REXX instruction is one ore more clauses, the first clause is the one that identifies the instruction. Instructions end with a semicolon or with a new line. One instruction may be continued from one line to the next by using a comma at the end of the line. Open strings or comments are not affected by line ends.

General Guidelines

`name;`
refers to a variable, which can be assigned any value. name is a symbol with the following exception: the first character may not be a digit or a period. The value of name is translated to uppercase before use, and forms the initial value of the value of the variable. Some valid names are:

- Fred
- COST?
- next
- index
- A.j

`name:`
is a form of labels for CALL instructions, SIGNAL instructions, and internal function calls. The colon acts as a clause separator.

`template;`
is a parsing template, described in a later section.

`instr;`
is any one of the listed instructions.

Instructions

`expression;`
the value of expression is issued as a command, normally to the command interpreter or to the specified environment specified by the ADDRESS instruction. Look also the section “Issuing Commands to Host System.”

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

```
1 fred = 'sunset'
2 a = 1 + 2 * 3
3 a = /* a contains '' */
```

`ADDRESS [<symbol | string> [expr]] | VALUE expr | (env);`
redirect commands or a single command to a new environment. ADDRESS VALUE expr may be used for an evaluated environment name.

```
1 address int2e 'dir' /* executes through int2e a dir cmd */
2 address system /* all the following command will be addressed to system */
3 env = 'dos'
4 address value env /* change address to dos */
5 address (env) /* change address to dos */
```

`ARG <template>;`
parse argument string(s) given to program or in an internal routine into variables according to template. Arguments are translated into uppercase before the parsing. Short for PARSE UPPER ARG.

```
1  /* program is called with args "autoexec.bat auto.old" */
2  arg src dest
3  /* src = "AUTOEXEC.BAT", dest="AUTO.OLD" */
4  /* a function is called MARMITA('Bill',3) */
```

```
5  marmita:
6  arg firstarg, secondarg
7  /* firstarg = "BILL", secondarg = "3" */
```

CALL [symbol | string] [<expr>] [,<expr>]... ;
[ON|OFF <condition> [NAME label]];

call an internal routine, an external routine or program, or a built-in function. Depending on the type of routine called, the variable RESULT contains the result of the routine. RESULT is uninitialized if no result is returned.

```
1  CALL SUBSTR 'makedonia',2,3
2  /* now. variable result = 'ake' */
3  /* the same can be obtained with */
4  result = SUBSTR('makedonia',2,3)
```

In the following sections there is a description of all the built-in rexx functions.
Internal functions are sequence of instructions inside the same program starting at the label that matches the name in the CALL instruction.
If the function is not found in the current program, then REXX will search for a file that matches the name in the CALL instruction and the same extension like the current program, and will load it as an external rexx function.
External routines are like internal but written in a separate module that can be used as a library. Rexx libraries are rexx files with many external routines which must be loaded with the built-in function LOAD before they are used (see below).
As external routines can be used any DOS command or program that uses standard input and output.

```
1  /* external programs can be called as routines */
2  /* and the output of the program (to stdout) will */
3  /* be returned as the result string of the function */
4  CALL "dir" "*.exe","/w" /* or */
5  files = "dir"('*.exe',"/w")
6  current_directory = 'cd'()
```

For CALL ON/OFF condition look below at the SIGNAL instruction.

DO [name=expri [TO exprt] [BY exprb] [FOR exprf]] | [FOREVER | exprr] ;
[UNTIL expru | WHILE exprw] ;
[instr]... ;
END[symbol] ;

DO is used to group many instructions together and optionally executes them repetively.
Simple DO loop are used to execute a block of instructions often used with IF-THEN statements.

> ## Note
>
> Simple DO loops are not affected with ITERATE or LEAVE instructions (see below)

```
1  IF name = 'Vivi' THEN DO
2          i = i + 1
3          SAY 'Hello Vivi'
4      END
```

Simple repetitive loops.

> ## Note
>
> in DO expr, expr must evaluate to an integer number.

```
1  DO 3    /* would display 3 'hello' */
2      SAY 'hello'
3  END
```

Inifinite loops

```
1  DO FOREVER    /* infinite loop, display always */
2      SAY 'lupe forever'    /* 'hello' */
3  END
```

Loops with control variable. name is stepped from expri to exprt in steps of exprb, for a maximum of exprf iterations.

```
1  DO i = 1 TO 10 BY 3     /* would display the numbers */
2      SAY i     /* 1, 4, 7, 10 */
3  END
```

> **Note**
>
> all the expressions are evaluated before the loop is executed and may result to any kind of number, integer or real.

Conditional loops

```
1  a = 2         /* would display */
2  DO WHILE a < 5     /* 2 */
3      SAY a     /* 4 */
4      a = a + 2
5  END
```

> **Note**
>
> exprw and expru are evaluated in each iteration and must result to 0 or 1. WHILE expression is evaluated before each iteration, where UNTIL expression is evaluated at the end of each iteration.

You can combine them like:

```
1  a = 1            /* would display */
2  DO FOR 3 WHILE a < 5     /* 1 */
3      SAY a        /* 2 */
4      a = a + 1        /* 3 */
5  END
```

DROP <name | (nameind)> [<name | (nameind)>]... ;

DROP (reset) the named variables or group of variables by freeing their memory. It returns them in their original uninitialized state. If a variable is enclosed in parenthesis then DROP resets all the variables that nameind contains as separate words. If an exposed variable is named, the variable itself in the older generation will be dropped! If a stem is specified all variables starting with that stem will be dropped.

```
1  j = 2
2  vars="j b stem."
3  DROP a x.1 y.j        /* resets variables A X.1 and Y.2 */
4  DROP z.        /* resets all variables with names
5      starting with Z. */
6  DROP (name)        /* resets variables j b and stem. */
```

EXIT [expr] ;

leave the program (with return data, expr). EXIT is the same as RETURN except that all internal routines are terminated.

```
1  EXIT 12*3    /* will exit the program with RC=36 */
```

IF expr [;] THEN [;] instr ;
[ELSE [;] instr];

if expr evaluates to “1”, executes the instruction following the THEN. Otherwise, when expr evaluates to “0”, the instruction after ELSE is executed, if ELSE is present.

```
1  IF name="Vivi"    THEN SAY "Hello Vivian"
2      ELSE SAY "Hello stranger"
```

INTERPRET expr ;
expr is evaluated and then is processed, as it was a part of the program.

```
1  cmd = "SAY 'Hello'"
2  INTERPRET cmd     /* displayes "Hello" */
```

ITERATE [name] ;
start next iteration of the innermost repetitive loop (or loop with control variable name).

```
1  DO    i = 1 TO 5        /* would display:     1 */
2      IF i=3 THEN ITERATE        /*     2 */
3      SAY i         /*     4 */
4  END          /*     5 */
```

LEAVE [name] ;
terminate innermost repetitive loop (or loop with control variable name).

```
1  DO    i = 1 TO 5        /* would display:     1 */
2      IF i=3 THEN LEAVE        /*     2 */
3      SAY i
4  END
```

LOWER name [name]...
translate the values of the specified individual variables to lowercase.

```
1  name = 'ViVi'
2  LOWER name        /* now, name = 'vivi' */
```

NOP ;
dummy instruction, has no effect.

```
1  IF name^='Vivi' THEN NOP; ELSE SAY 'Hello Vivi.'
```

NUMERIC DIGITS [expr] | FORM [SCIENTIFIC | ENGINEERING] | FUZZ [expr] ;
Set the number of significant digits used for all arithmetic operations.

> ## Note
>
> In BRexx all numerical operations are performed either with the 32bit integer type or 64 double precission, so the numeric digits is limited for floating point operations to maximum 22 digits.

PARSE [type] + ARG + [template] ;
Parse is used to assign data from various sources to one or more variables according to the template (see below for template patterns) where the optional type is one of:

- ARG, parses the argument string(s) passed to the program, subroutine, or function. UPPER first translates the strings to uppercase. See also the ARG instruction.
- AUTHOR parse the author string.
- EXTERNAL, prompts for input and parses the input string
- LINEIN, same as EXTERNAL
- NUMERIC, parse the current NUMERIC settings.
- PULL, read and parse the next string from REXX stack if not empty otherwise prompts for input. See the PULL instruction.

- SOURCE, parse the program source description e.g. “MSDOS COMMAND prog.r C:REXX.EXE C:DOSCOMMAND.COM”
- VALUE, parse the value of expr.
- VAR, parse the value of name.
- VERSION, parse the version string of the interpreter.

PROCEDURE [EXPOSE name|(varind) [name|(varind)]...] ;
start a new generation of variables within an internal routine. Optionally named variables or groups of variables from an earlier generation may be exposed. If a stem is specified (variable ending in ‘.’ dot, ie ‘A.’) then every variable starting with this stem will be exposed. Indirect exposure is also posible by enclosing inside parenthesis the variable varind which contains contains as separate words all variables to be exposed

```
 1  i = 1
 2  j = 2
 3  ind = "i j"
 4  CALL myproc
 5  CALL myproc2
 6  EXIT
 7  myproc: PROCEDURE EXPOSE i     /* would display */
 8  SAY i j         /* 1 J */
 9  RETURN
10  myproc2: PROCEDURE EXPOSE (ind)     /* would display */
11  say i j         /* 1 2 */
12  RETURN
```

PULL [template] ;
pops the next string from rexx internal stack. If stack is empty then it prompts for input. Translates it to uppercase and then parses it according to template. Short for PARSE UPPER PULL.

```
1  PUSH 'Vassilis Vlachoudis'
2  /* --- many instrs ---- */
3  PULL name surname     /* now: name='BILL', */
4      /* surname='VLACHOUDIS' */
```

PUSH [expr] ;
push expr onto head of the rexx queue (stack LIFO)

QUEUE [expr] ;
add expr to the tail of the rexx queue (stack FIFO)

RETURN [expr] ;
return control from a procedure to the point of its invocation. if expr exits, then it is returned as the result of the procedure.

```
1  num = 6
2  SAY num || '! = ' fact(num)
3  EXIT
4  fact: PROCEDURE         /* calculate factorial with */
5  IF arg(1) = 0 THEN RETURN 1         /* recursion */
6  RETURN fact(ARG(1)-1) * ARG(1)         /* displayes: 6! = 720 */
```

SAY [expr];
evaluate expr and then writes the result to standard output (normally user’s console) followed by a newline.

SELECT ;
WHEN expr [;] THEN [;] instr;
[WHEN expr [;] THEN [;] instr;]
[OTHERWISE [;] [instr]...];
END ;
SELECT is used to conditionally process one of several alternatives. Each WHEN expression is evaluated in sequence until one results in “1”. instr, immediately following it, is executed and control leaves the block. If no expr

evaluated to “1”, control passes to the instructions following the OTHERWISE expression that must then be present.

```
num = 10
SELECT
    WHEN num > 0 THEN SAY num 'is positive'
    WHEN num < 0 THEN SAY num 'is negative'
    OTHERWISE SAY num 'is zero'
END
```

SIGNAL [name] |
[VALUE] expr |
<ON | OFF> + condition + [NAME label];

Parameters: **condition** – Can be one of *ERROR HALT NOTREADY NOVALUE SYNTAX*

- name, jump to the label name specified. Any pending instructions, *DO … END*, *IF*, *SELECT*, and *INTERPRET* are terminated.
- VALUE, may be used for an evaluated label name.
- ON|OFF, enable or disable exception traps.
- Condition must be ERROR, HALT, NOTREADY, NOVALUE, or SYNTAX. Control passes to the label of the condition name if the event occurs while ON or to label if NAME label is specified.

```
SIGNAL vivi
...
vivi:
SAY 'Hi!'
```

A condition example:

```
SIGNAL ON SYNTAX NAME syntax_error
SAY 1/0     /* Control passes to label syntax_error */
...
syntax_error:
SAY 'Syntax error in line:' SIGL
```

TRACE option | VALUE expr;
Trace according to following option. Only first letter of option is significant.

- A (All) trace all clauses.
- C (Commands) trace all commands.
- E (Error) trace commands with non-zero return codes after execution.
- I (Intermediates) trace intermediate evaluation results and name substitutions also.
- L (Labels) trace only labels.
- N (Negative or Normal) trace commands with negative return codes after execution (default setting).
- O (Off) no trace.
- R (Results) trace all clauses and expressions.
- S (Scan) display rest of program without any execution (shows control nesting).
- ? turn interactive debug (pause after trace) on or off, and trace according to next character. null restores the default tracing actions.

TRACE VALUE expr may be used for an evaluated trace setting.

UPPER name [name]...
translate the values of the specified individual variables to uppercase.

```
name = 'Vivi'
UPPER name        /* now: name = 'VIVI' */
```

Templates for ARG, PULL, and PARSE

The *PULL*, *ARG* and *PARSE* instructions use a template to parse a string.

The simplest template is a list of variables where each of them is assigned one word from the string, except the last variable in the list which will contain the rest of the string.

```
1  PARSE VALUE "one two three four " WITH a b c
2  now a="one"; b="two"; c="three four"
3  PARSE VALUE "one two three four " WITH a b c d e
4  now a="one"; b="two"; c="three"; d="four" and e=""
```

A dot '.' can be in the place of one or more variables, it is used as a place-holder.

```
1  PARSE VALUE "one two three four " WITH a . . d
2  now a="one"
3  d="four"
```

A more complex parsing is to use patterns for triggering:

- **number** which specifies an absolute position in string 1 - is the first character in string
- **=(name)** as a position may be a variable enclosed in parenthesis after an equal symbol
- **[+|-]number** signed numbers are used as a relative positioning

```
1 PARSE VALUE "one two three four " WITH 2 a 6 b
2 now a="ne t"; b="wo three four " pos=6;
3 PARSE VALUE "one two three four " WITH 2 a =(pos) b
4 now a="ne t"; b="wo three four " PARSE VALUE "one two three four " WITH 2 a +2 b
5 now a="ne"; b=" two three four "
```

- **string** - may be used as a target position.

```
1  PARSE VALUE "marmita/bill/vivi' WITH a '/' b '/' c
```

now *a="marmita"*; *b="bill"*; *c="vivi"*

- **(name)** - also as a target may be used a variable encolsed in parenthesis

```
1  t = "%%"
2  PARSE VALUE "aabbcc%%ddeeff%%gg%%" WITH . (t) middle (t) . now middle="ddeeff"
```

A **comma** can be used as a "trigger" to move to the next string when there is more than one to be parsed (e.g. when there is more than one argument string to a routine).

```
1  CALL MyProc 'Hi',3,4
2  EXIT
3  MyProc:
4  PARSE ARG first, second, third      /* now first="Hi" */
5  ...          /* second=3 */
6  /* third=4 */
```

Compound Variable Names

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

```
1 s0.s1.s2.---.sn     /* is substituted to form */
2 d0.v1.v2.---.vn     /* where d0 is uppercase of s0, and
3     v1-vn are values of s1-sn */
```

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

```
1 J = 5
2 a.j = "fred"
```

would assign fred to the variable A.5.

The stem of name (i.e. that part up to and including the first “.”) may be specified on the *DROP* and *PROCEDURE EXPOSE* instructions and affect all variables starting with that stem. An assignment to a stem assigns the new value to all possible variables with that stem.

Special Variables

There are three special variables:

SIGL

holds the line number of the instruction that was last executed before control of program was transferred to another place. This can be caused by a SIGNAL instruction, a CALL instruction or a trapped error condition.

RC

is set to the errorlevel (return-code) after execution of every command (to host).

RESULT

is set by a RETURN instruction in a CALLed procedure.

Interactive Debugging

You can enter the interactive debuging either by executing a TRACE instruction with a prefix ‘?’ or when calling REXX from command line issuing as a first argument the trace option:

rexx ‘?A’ ‘HLQ.DATASET(MEMBER)’

In interactive debug, interpreter pauses before the execution of the instructions that are to be traced and prompts for input. You may do one of following things:

- Enter a null line to continue execution.
- Enter a list of REXX instructions, which are interpreted immediately (*DO-END* instructions must be complete, etc.).

During the execution of the string, no tracing takes place, except that non-zero return codes from host commands are displayed. Execution of a TRACE instruction with the “?” prefix turns off interactive debug mode. Other TRACE instructions affect the tracing that occurs when normal execution continues.

Built-in Functions

The following are the built-in REXX functions. Which are divided into the following categories:

- Rexx
- String
- Word
- Math
- Data Convert
- File Functions

Rexx Functions

ADDR (symbol[,[option][,pool]])
returns the physical address of symbol contents. Option can be 'Data' (default) variables data 'Lstring' lstring structure pointer 'Variable' variable structure.
If pool exist, the specific rexx pool is searched for the symbol. Valid pools are numbers from 0 up to current procedure nesting. (The result is normalized for MSDOS, ie *seg:ofs = seg*16+ofs*)

```
1  i = 5
2  SAY addr('i')           /* something like 432009 decimal */
3  SAY addr('i','L')       /* something like 433000 */
4  SAY addr('i','V')       /* something like 403004 */
5  SAY addr('i','V',0)     /* something like 403004 */
6  SAY addr('j')           /* -1, is J variable doesn't exist */
```

ADDRESS ()
return the current environment for commands.

```
1  SAY address() /* would display: SYSTEM */
```

ARG ([, n[, option]])

- ARG() returns the number of arguments
- ARG(n) return then nth argument
- ARG(n,option) option may be **Exist** or **Omitted** (only the first letter is significant) test whether the nth argument Exists or is Omitted.

Returns "0" or "1"

```
1  call myproc 'a',,2
2  ...
3  myproc:
4  SAY arg()               /* 3 */
5  SAY arg(1)              /* 'a' */
6  SAY arg(2,'O')          /* 1 */
7  SAY arg(2,'E')          /* 0 */
```

DATATYPE (string[, type])
DATATYPE(string) - returns "NUM" is string is a valid REXX number, otherwise returns "CHAR". DATATYPE(string,type) - returns "0" or "1" if string is of the specific type:

- Alphanumeric: characters A-Z, a-z and 0-9
- Binary: a valid BINARY number
- Lowercase: characters a-z
- Mixed: characters A-Z, a-z
- Number: is a valid REXX number
- Symbol: characters A-Z, a-z, 0-9, @%_.!#
- Uppercase: characters A-Z
- Whole-number: a valid REXX whole number
- X (heXadecimal): a valid HEX number

(only the first letter of type is required)
The special type 'Type' returns the either INT, REAL, or STRING the way the variable is hold into memory. Usefull when you combine that with INTR function.

```
1  SAY datatype('123')             /* NUM */
2  SAY datatype('21a')             /* CHAR */
3  SAY datatype(01001,'B')         /* 1 */
4  SAY datatype(i,'T')             /* maybe STRING */
```

DATE ([, option])
return current date in the format: dd Mmm yyyy

```
1  SAY date() /* 14 Feb 1993 */
```

or formats the output according to option

- Days returns number of days since 1-Jan as an integer
- European returns date in format dd/mm/yy
- Month returns the name of current month, ie. March
- Normal returns the date in the default format dd Mmm yyyy
- Ordered returns the date in the format yy/mm/dd
- (useful for sorting)
- Sorted returns the date in the format yyyymmdd
- (suitable for sorting)
- USA returns the date in the format mm/dd/yy
- Weekday returns the name of current day of week ie. Monday

DESBUF ()
destroys the all system stacks, and returns the number of lines in system stacks.

```
1  PUSH 'hello'            /* now stack has one item */
2  CALL desbuf             /* stack is empty, and RESULT=1 */
```

DROPBUF (**[**, num**]**)
destroys num top stacks, and returns the number of lines destroyed.

```
1  PUSH 'in stack1'                /* first stack has one item */
2  CALL makebuf            /* create a new buffer */
3  PUSH 'in stack2'                /* new stack has one item */
4  CALL dropbuf            /* one stack remains */
```

DIGITS ()
returns the current setting of NUMERIC DIGITS.

ERRORTEXT (n)
returns the error message for error number n.

```
1  SAY errortext(8)        /* "Unexpected THEN or ELSE" */
```

FORM ()
returns the current setting of NUMERIC FORM.

FUZZ ()
returns the current setting of NUMERIC FUZZ.

GETENV (varname)
returns the environment variable varname

```
1  SAY getenv("PATH")
```

HASHVALUE (string)
return an integer hashvalue of the string like Java *hash = s[0]*31^(n-1) + s[1]*31^(n-2) + … + s[n-1]*

```
1  SAY hashvalue("monday")         /* -1068502768 */
```

IMPORT (file)
import a shared library file using dynamic linking with rexx routines. If it fails, then try to load a rexx file so it can be used as a library. import first searches the current directory, if not found it searches the directories pointed by the environment variable RXLIB.
returns

- “-1” if already imported

- “0” on success
- “1” on error opening the file

```
1  CALL IMPORT FSSAPI
2  call import "veclib"
```

MAKEBUF ()
create a new system stack, and returns the number of system stacks created until now (plus the initial one).

```
1  PUSH 'hello'; SAY queued() queued(T)            /* display 1 1 */
2  CALL makebuf            /* create a new buffer */
3  PUSH 'aloha; SAY queued() queued(T)             /* display again 2 1 */
```

QUEUED (**[**, option**]**)
return the number of lines in the rexx stack (all stacks or the topmost) or the number of stacks. Option can be (only first letter is significant):

- All lines in All stacks (default)
- Buffers number of buffers created with MAKEBUF
- Topstack lines in top most stack

```
1  PUSH 'hi'
2  SAY queued(A) queued(B) queued(T)               /* 1 1 1 */
3  CALL makebuf
4  SAY queued(A) queued(B) queued(T)               /* 1 2 0 */
5  PUSH 'hello'
6  SAY queued(A) queued(B) queued(T)               /* 2 2 1 */
7  CALL desbuf
8  SAY queued(A) queued(B) queued(T)               /* 0 1 0 */
```

SOUNDEX (word)
return a 4 character soundex code of word in the format “ANNN” (used for phonetic comparison of words)

```
1  SAY soundex("monday")           /* M530 */
2  SAY soundex("Mandei")           /* M530 */
```

SOURCELINE (**[**, n**]**)
return the number of lines in the program, or the nth line.

```
1  SAY sourceline()        /* maybe 100 */
2  SAY sourceline(1)       /* maybe "/**/" */
```

STORAGE ([address[,[length][,data]]])
returns the current free memory size expressed as a decimal string if no arguments are specified. Otherwise, returns length bytes from the user’s memory starting at address. The length is in decimal; the default value is 1 byte. The address is a decimal number (Normalized address for MSDOS ie. seg:ofs = seg*16+ofs) If data is specified, after the “old” value has been retrieved, storage starting at address is overwritten with data (the length argument has no effect on this).

```
1  SAY storage()           /* maybe 31287 */
2  SAY storage(1000,3)             /* maybe "MZa" */
3  a = "Hello"
4  SAY storage(addr('a'),5,'aaa')  /* "Hello" */
5  SAY a           /* aaalo */
```

SYMBOL (name)
return “BAD” if name is not a valid REXX variable name, “VAR” if name has been used as a variable, or “LIT” if it has not.

```
1  i = 5
2  SAY symbol('i')         /* VAR */
```

```
3  SAY symbol(i)           /* LIT */
4  SAY symbol(':asd')              /* BAD */
```

TIME ([, option])
return the local time in the format: *hh:mm:ss* if option is specified time is formated as:

- Civil returns time in format hh:mmxx ie. 10:32am
- Elapsed returns elapsed time since rexx timer was reset or from begging of program in format ssssss.uuuuuu
- Hours returns number of hours since midnight
- Long returns time and milliseconds hh:mm:ss.uu
- Minutes returns number of minutes since midnight
- Normal returns time in format hh:mm:ss
- Reset returns elapsed time in format ssssss.uuuuuu (like Elapsed) and resets rexx internal timer.
- Seconds returns number of seconds since midnight

TRACE ([, option])
returns current tracing option. If option is specified then sets to new tracing option. Look up instruction TRACE.

```
1  SAY trace() /* normally 'N' */
```

VALUE (name[,[newvalue][,pool]])
returns the value of the variable name. If newvalue is specified then after the retrieval of the old value the newvalue will be set to the variable. If pool is specified then the operation takes place at the specific pool. Pool initially exist in this version of Rexx are:

- 0 up to the current PROCEDURE nesting specifing the pool of each PROCEDURE
- Negative values from -1 to minus current PROCEDURE nesting, shows relative values from current procedure.
- SYSTEM is the system pool (like GETENV,PUTENV)
- User can create his own POOLs, Look Programing Rexx

```
1  i = 5
2  j = "i"
3  SAY value(j)            /* 5 */
4  SAY value('j',10)               /* 'i' */
5  SAY j          /* 10 */
6  CALL Procedure
7  ...
8  Procedure: PROCEDURE
9  i = "I-var"
10 SAY value('i')          /* I-var */
11 SAY value('i',,0)               /* 5 */
12 SAY value('i',,1)               /* I-var */
13 SAY value('i',,-1)              /* 5 */
14 ...
```

VARDUMP ([symbol][,option])
returns the binary tree of the variables in the format

```
1  var = "value" \n
```

option can be “Depth” which prints out the binary tree in the format

```
1  depth var = "value" \n (used for balancing of variables )
```

symbol may be nothing for main bin-tree or a stem for an array bin-tree ie. “B.”
VARDUMP is an easy way to store the variables in a file or in stack and restores them later.

```
1  CALL write "vars.$$$", vardump() /* stores all variables */ /* in the file "vars.$$$" *
```

on a later run you can do

```
DO UNTIL eof("vars.$$$")                /* this will read all variables */
         INTERPRET read("vars.$$$")              /* from file, and restore them */
END
```

> **Warning**
>
> VARDUMP is not fully implemented and may not work when variables have non-printable characters.

String Functions

ABBREV (information, info[, length])
tests whether info is an abbreviation of information. returns “1” on true, else returns “0”. If length is specified then searching takes place only for the first length characters.

```
abbrev("billy","bill")          /* 1 */
abbrev("billy","bila")          /* 0 */
abbrev("billy","bila",3)        /* 1 */
```

CENTRE (string, length[, pad])
returns string centered in a padded string of length length.

```
center("rexx",2)                /* 'ex' */
center("rexx",8)                /* ' rexx ' */
center("rexx",8,'-')            /* '--rexx--' */
```

CHANGESTR (target, string, replace)
replaces all occurrences of the target in string, replacing them with the replace.

```
changestr("aa","aabbccaabbccaa","--")   /* --bbcc--bbcc-- */
```

COMPARE (string1, string2[, pad])
returns “0” if string1==string2, else it returns the index of the first nonmatching character. Shorter string is padded with pad if necessary

```
compare('bill','bill')          /* 0 */
compare('bill','big')           /* 3 */
compare('bi ','bi')             /* 0 */
compare('bi--*','bi','-')       /* 5 */
```

COUNTSTR (target, string)
counts all the appearances of target in string

```
countstr("aa","aabbccaabbccaa")         /* 3 */
```

COPIES (string, n)
returns n concatenated copies of string.

```
copies('Vivi',3)        /* 'ViviViviVivi' */
```

DELSTR (string, n[, length])
delete substring of string starting at the nth character and of length length.

```
delstr('bill',3)                /* 'bi' */
delstr('bill',2,2)              /* 'bl' */
```

INDEX (haystack, needle[, start])
return the position of needle in haystack, beginning at start.

```
index('bilil','il')             /* 2 */
index('bilil','il',3)           /* 4 */
```

INSERT (new,target[,[n][,[length][,pad]]])
insert the string new of length length into the string target, after the nth character (n can be 0)

```
insert('.','BNV',2)             /* 'BN.V' */
insert('.','BNV',2,2)           /* 'BN. V' */
insert('','BNV',2,2,'.')        /* 'BN..V' */
```

LASTPOS (needle, haystack**[**, start**]**)
return the position of the last occurrence of needle in haystack, beginning at start.

```
lastpos('il','bilil')           /* 4 */
lastpos('il','bilil',4)         /* 2 */
```

LEFT (string, length**[**, pad**]**)
return a string of length length with string left justified in it.

```
left('Hello',2)          /* 'He' */
left('Hello,10,'.')              /* 'Hello.....' */
```

LENGTH (string)
return the length of string

```
length('Hello')          /* 5 */
```

OVERLAY (new,target[,[n][,[length][,pad]]])
overlay the string new of length length onto string target, beginning at the nth character.

```
overlay('.','abcd',2)           /* 'a.cd' */
overlay('.','abcd')             /* '.bcd' */
overlay('.','abcd',6,3,'+')             /* 'abcd+.++' */
```

POS (needle, haystack**[**, start**]**)
return the position of needle in haystack, beginning at start.

```
pos('ll','Bill')                /* 3 */
```

REVERSE (string)
swap string, end-to-end.

```
reverse('Bill')          /* 'lliB' */
```

RIGHT (string, length**[**, pad**]**)
returns length righmost characters of string.

```
right('abcde',2)                /* 'de' */
```

SUBSTR (string,n[,[length][,pad]])
return the substring of string that begins at the nth character and is of length length. Default pad is space.

```
substr('abcde',2,2)             /* 'bc' */
substr('abcde',2)               /* 'bcde' */
substr('abcde',4,3,'-')         /* 'de-' */
```

STRIP (string[,[<"L"|"T"|"B">][,char]])

```
1
```

returns string stripped of Leading, Trailing, or Both sets of blanks or other chars. Default is “B”.

```
strip(' abc ')          /* 'abc' */
strip(' abc ','t')              /* ' abc' */
strip('-abc--',,'-')            /* 'abc' */
```

TRANSLATE (string[,[tableo][,[tablei][,pad]]])
translate characters in tablei to associated characters in tableo. If neither table is specified, convert to uppercase.

```
translate('abc')                /* 'ABC' */
translate('aabc','-','a')               /* '--bc' */
translate('aabc','-+','ab')             /* '--+c' */
```

VERIFY (string,reference[,[option][,start]])
return the index of the first character in string that is not also in reference. if “Match” is given, then return the result index of the first character in string that is in reference.

```
verify('abc','abcdef')          /* 0 */
verify('a0c','abcdef')          /* 2 */
verify('12a','abcdef','m')              /* 3 */
```

XRANGE ([start][,end])
return all characters in the range start through end.

```
xrange('a','e')         /* 'abcde' */
xrange('fe'x,'02'x)             /* 'feff000102'x */
```

Word Functions

DELWORD (string, n[, length])
delete substring of string starting at the nth word and of length length words.

```
delword('one day in the year',3)                /* 'one day' */
delword('one day in the year',3,2)              /* 'one day year' */
```

FIND (string, phrase[, start])
returns the word number of the first word of phrase in string. Returns “0” if phrase is not found. if start exists then search start from start word.

```
find('one day in the year','in the')    /* 3 */
```

JUSTIFY (string, length[, pad])
justify string to both margins (the width of margins equals length), by adding pads between words.

```
justify('one day in the year',22)       /*'one day in the year'
```

SUBWORD (string, n[, length])
return the substring of string that begins at the nth word and is of length length words.

```
subword('one day in the year',2,2)      /* 'day in' */
```

SPACE (string[,[n][,pad]])
formats the blank-delimited words in string with n pad characters between each word.

```
space('one day in the year',2)  /*'one day in the year' */
```

WORDS (string)
return the number of words in string

```
words('One day in the year')            /* 5 */
```

WORD (string, n)
return the nth word in string

```
word('one day in the year',2)          /* 'day' */
```

WORDINDEX (string, n)
return the position of the nth word in string

```
wordindex('one day in the year',2)     /* 5 */
```

WORDLENGTH (string, i)
return the length of the nth word in string

```
wordlength('one day in the year',2)    /* 3 */
```

WORDPOS (phrase, string[, start])
returns the word number of the first word of phrase in string. Returns “0” if phrase is not found

```
wordpos('day in','one day in the year')        /* 2 */
```

Math Functions

ABS (number)
return absolute value of number

```
abs(-2.3) /* 2.3 */
```

FORMAT (number[,[before][,[after][,[expp][,expt]]]])
rounds and formats number with before integer digits and after decimal places. expp accepts the values 1 or 2 (WARNING Totally differen’t from the Ansi-REXX spec) where 1 means to use the “G” (General) format of C, and 2 the “E” exponential format of C. Where the place of the totalwidth specifier in C is replaced by before+after+1. (expt is ignored!)

```
format(2.66)            /* 3 */
format(2.66,1,1)                /* 2.7 */
format(26.6,1,1,1)              /* 3.E+01 */
format(26.6,1,1,2)              /* 2.7E+01 */
```

IAND (n, m)
return bitwise AND of the integers n and m

```
iand(2,3) /* 2 */
```

INOT (n)
return bitwise complement of integers n

```
inot(2) /* -3 */
```

IOR (n, m)
return bitwise OR of the integers n and m

```
ior(2,3) /* 3 */
```

IXOR (n, m)
return bitwise XOR of the integers n and m

```
ixor(2,3) /* 1 */
```

MAX (number[, number]..)
returns the largest of given numbers.

```
max(2,3,1,5) /* 5 */
```

MIN (number[, number]..)
returns the smallest of given numbers.

```
1  min(2,3,1,5) /* 1 */
```

RANDOM ([min][,[max][,seed]])
returns a pseudorandom nonnegative whole number in the range min to max inclusive.

SIGN (number)
return the sign of number (“-1”,”0” or “1”).

```
1  sign(-5.2)              /* -1 */
2  sign( 0.0)              /* 0 */
3  sign( 5.2)              /* 1 */
```

TRUNC (number**[**, n**]**)
returns the integer part of number, and n decimal places. The default n is zero.

```
1  trunc(2.6) /* 2 */
```

ACOS (num)
Arc-cosine

ASIN (num)
Arc-sine

ATAN (num)
Arc-tangent

COS (num)
Cosine

COSH (num)
Hyperbolic cosine

EXP (num)
Exponiate

LOG (num)
Natural logarithm

LOG10 (num)
Logarithm of base 10

POW10 (num)
Power with base 10

SIN (num)
Sine function

SINH (num)
Hyperbolic sine

SQRT (num)
Square root

TAN (num)
Tangent

TANH (num)
Hyperbolic tangent

POW (a, b)
Raises a to power b

Data Convert Functions

B2X (string)
Binary to Hexadecimal

```
1  b2x('01100001')         /* 'a' */
```

BITAND (string1[,[string2][,pad]])
logically AND the strings, bit by bit

```
1  bitand('61'x,'52'x)            /* '40'x */
2  bitand('6162'x,'5253'x)        /* '4042'x */
3  bitand('6162'x,,'FE'x)         /* '6062'x */
```

BITOR (string1[,[string2][,pad]])
logically OR the strings, bit by bit

BITXOR (string1[,[string2][,pad]])
logically XOR the strings, bit by bit

C2D (string**[**, n**]**)
Character to Decimal. The binary representation of string is converted to a number (unsigned unless the length n is specified).

```
1  c2d('09'x)             /* 9 */
2  c2d('ff40')            /* 65344 */
3  c2d('81'x,1)           /* -127 */
4  c2d('81'x,2)           /* 129 */
```

C2X (string)
Character to Hexadecimal

```
1  c2x('abc')             /* '616263' */
2  c2x('0506'x)           /* '0506' */
```

D2C (wholenumber**[**, n**]**)
Decimal to Character. Return a string of length n, which is the binary representation of the number.

```
1  d2c(5)          /* '5'x */
2  d2c(97)         /* 'a' */
```

D2X (wholenumber**[**, n**]**)
Decimal to Hexadecimal. Return a string of length n, which is the hexadecimal representation of the number.

```
1  d2x(5)          /* '05' */
2  d2x(97)         /* '61' */
```

B2X (string)
Hexadecimal to Binary

```
1  x2b('a')                /* '01100001' */
```

X2C (string)
Hexadecimal to Character

```
1  x2c('616263')           /* 'abc' */
```

X2D (hex-string**[**, n**]**)
Hexadecimal to Decimal. hex-string is converted to a number (unsigned unless the length n is specified)

```
1  x2d('61')               /* 97 */
```

File Functions

General

There are two sets of I/O functions, the REXX-STEAM functions and the BREXX I/O routines.

Files can be referenced as a string containing the name of the file ie “TEST.DAT” or the file handle that is returned from OPEN function. (Normally the second way if prefered when you want to open 2 or more files with the same name).

There are always 3 special files:

Handle	FileName	Description
0	<STDIN>	Standard input
1	<STDOUT>	Standard output
2	<STDERR>	Standard error

All open files are closed at the end of the program from REXX interpreter except in the case of an error.

CHARIN ([stream[,[start][,[length]]]])
reads length bytes (default=1) from stream (default="<STDIN>") starting at position start

```
1  ch = charin("new.dat")          /* read one byte */
2  ch = charin("new.dat",3,2)      /* read two bytes from position in file 3 */
```

CHAROUT ([stream[,[string][,[start]]]])
write string to stream (default="<STDOUT>") starting at position start

```
1  CALL charout "new.dat","hello"  /* writes "hello" to file */
2  CALL charout "new.dat","hi",2)  /* writes "hi" at position 2 */
```

CHARS ([, stream])
returns the number of characters remaining in stream.

```
1  CHARS("new.dat")        /* maybe 100 */
```

CLOSE (file)
closes an opened file. file may be string or the handle number

```
1  CALL close 'new.dat'    /* these two cmds are exactly the same */
2  CALL close hnd  /* where hnd=open('new.dat','w') */
```

EOF (file)
returns 1 at eof, -1 when file is not opened, 0 otherwise

```
1  DO      UNTIL eof(hnd)=1
2          SAY read(hnd)   /* type file */
3  END
```

FLUSH (file)
flush file stream to disk

```
1  CALL flush 'new.dat'
```

LINEIN ([stream[,[start][,[lines]]]])
reads lines lines (default=1) from stream (default="<STDIN>") starting at line position start

```
1  line = linein("new.dat")              /* read one line */
2  line = linein("new.dat",3,2)    /* read two lines from new.dat starting at line 3 */
```

LINEOUT ([stream[,[string][,[start]]]])
write string with newline appended at the end to stream (default="<STDOUT>") starting at line position start

```
1  CALL lineout "new.dat","hello"  /* writes line "hello" to file */
2  CALL lineout "new.dat","hi",2)  /* writes line "hi" at line position 2 */
```

LINES ([, stream])
returns the number of lines remaining in stream. start

```
1  LINES("new.dat")        /* maybe 10 */
```

OPEN (file, mode)
opens a file. mode follows C prototypes:

“r”	for read	“w”	for write
“t”	for text,	“b”	for binary
“a”	for append	“+”	for read/write

and returns the handle number for that file. -1 if file is not found!

```
1  hnd     = open('new.dat','w')
2  IF      hnd = -1 THEN DO
3          SAY 'Error: opening file "new.dat".'
4          ...
5  END
6  irda = open('com3:115200,8,N,1,128','rw')
```

READ ([file][,<length | "Char" | "Line" | "File">])

reads one line from file. If the second argument exists and it is a number it reads length bytes from file otherwise reads a Char, Line or the entire File. If file is not opened, it will be opened automatically in “r” mode. If file is ommited, it is assumed to be <STDIN>

```
1  kbdin = READ()  /* reads one line from stdin */
2  keypressed = read(,1)   /* -//- char -//- */
3  linein = read('new.dat')        /* reads one line from file */
4  linein = read(hnd)      /* -//- */
5  ch = read('new.dat',"C")        /* if file 'data' is not opened
6          then it will be opened in "r" mode */
7  CALL write "new",read("old","F")        /* copy file */
```

SEEK (file**[**, offset**[**, <"TOF" | "CUR" | "EOF">**]]**)

move file pointer to offset relative from TOF Top Of File (default), CUR Current position, EOF End Of File and return new file pointer. This is an easy way to determine the filesize, by seeking at the end,

```
1  filesize = seek(file,0,"EOF")   /* return file size */
2  CALL seek 'data',0,"TOF"        /* sets the pointer to the start of the file */
3  filepos = seek('data',-5,"CUR")         /* moves pointer 5 bytes backwards */
```

STREAM (stream[,[option][,command]])

STREAM returns a description of the state, or the result of an operation upon the stream named by the first argument.
option can be “Command”, “Description”, “Status”
When option is “Command” the third argument must exist and can take on of the following values:

Command	Description
READ	open in read-only mode ASCII
READBINARY	open in read-only mode BINARY
WRITE	open in write-only mode ASCII
WRITEBINARY	open in write-only mode BINARY
APPEND	open in read/write-append mode ASCII
APPENDBINARY	open in read/write-append mode BINARY
UPDATE	open in read/write mode (file must exist) ASCII
UPDATEBINARY	open in read/write mode BINARY

When option is “Status”, STREAM returns the current status of the stream can be on of the followings: “READY”, “ERROR”, “UNKNOWN”
When option is “Description”, STREAM returns a description of the last error.

```
1  CALL stream "new.dat","C","WRITE"
2  CALL stream "new.dat","C","CLOSE"
3  CALL stream "new.dat","S"
```

WRITE([file][, string[,newline]])
writes the string to file. returns the number of bytes written. If string doesn't exist WRITE will write a newline to file. If a third argument exists a newline will be added at the end of the string. If file is not opened, it will be opened automatically with "w" mode. If file is ommited, it is assumed to be <STDOUT>

```
1  CALL write 'data','First line',nl
2  CALL write ,'a'          /* writes 'a' to stdout */
3  CALL write '','one line',nl     /* write 'one line' to stdout */
4  CALL write 'output.dat','blah blah'     /* writes 'blah blah' to 'output.dat' file*/
```

Calling external REXX Scripts or Functions

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

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:

1. *RX 'MY.EXEC(MYREX)'* if the script resides in a PDS, alternatively:
2. *RX 'MY.SAMPLE.REXX'* if it is a sequential dataset

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

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

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.

Calling external REXX scripts

It is now possible to call external REXX scripts, either by: *CALL your-script parm1,parm2…* or by function call: *value=your-script(parm1,parm2,…)* The call might take place from within your main REXX, or from a called subroutine. The search of the called script is performed in the following sequence:

- Internal sub-procedure or label (contained in the running REXX script)
- current PDS (where the calling REXX is originated) [1]
- from the delivered BREXX.V2R5M3.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)

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 control is returned.

BREXX MVS Functions

Host Environment Commands

ADDRESS MVS

Interface to certain REXX environments 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. The result of the command is displayed on screen, but can be trapped in a stem by the OUTTRAP command:

```
1 call outtrap('myresult.')
2 ADDRESS COMMAND 'CP help'
3 call outtrap('off')
4 /* result is stored in stem myresult. */
5 do i=1 to myresult.0
6 Say mayresult.i
7 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 0:0009 3215 *syscons cmdpref(/) IO[1541] open
HHC02279I 0:000C 3505 0.0.0.0:3505 sockdev ascii autopad trunc eof IO[3]
HHC02279I      (no one currently connected)
HHC02279I 0:000D 3525 /punchcards/pch00d.txt ebcdic IO[2] open
HHC02279I 0:000E 1403 /printers/prt00e.txt IO[6] open
HHC02279I 0:000F 3211 /printers/prt00f.txt IO[2] open
HHC02279I 0:0010 3270 GROUP=CONSOLE IO[3]
HHC02279I 0:0015 1403 /logs/mvslog.txt IO[2106] open
HHC02279I 0:001A 3505 0.0.0.0:3506 sockdev ebcdic autopad eof IO[3]
HHC02279I      (no one currently connected)
```

And many others: *ADDRESS COMMAND 'CP clocks'*:

```
HHC02274I tod = DC8F485DBB377093    2022.349 21:07:20.582007
HHC02274I h/w = DC8F485DBB377093    2022.349 21:07:20.582007
HHC02274I off = 0000000000000000       0.000 00:00:00.000000
HHC02274I ckc = DC8F485DC0400000    2022.349 21:07:20.602624
HHC02274I cpt = 7FFFFF7A01C85F00
HHC02274I itm = 7C0E1623                     07:31:40.233415
```

If you run under control of VM370 you can run VM commands: *ADDRESS COMMAND 'CP vm-command'*

ADDRESS FSS

Interface to the Formatted Screen Services. Please refer to formatted_screens.rst contained in the installation zip file.

Note

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

ADDRESS LINKMVS

Call an external program. The linkage convention of the called program can be found here: The LINKMVS and ATTCHMVS host command environments

Example:

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

ADDRESS LINKPGM

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

The LINKPGM and ATTCHPGM host command environments

ADDRESS ISPEXEC

Support calls functions to **Wally Mclaughlin ISPF** for MVS on Hercules. The functions supported depend on the functionality implemented in his API. Example:

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

OUTTRAP

If the commands writes output to terminal you can trap the output using the *OUTTRAP* command. This will redirect 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')*.

```
1 call outtrap('lcat.')
2 ADDRESS TSO 'LISTCAT LEVEL "BREXX"'
3 call outtrap('off')
4 /* listcat result is stored in stem lcat. */
5 do i=1 to lcat.0
6   Say lcat.i
7 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.

```
1 call arraygen('ON')
2 ADDRESS TSO 'LISTCAT LEVEL(BREXX)'
3 s1=arraygen('OFF')
4 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
```

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 into the BREXX User's Guide.

Note

When reading the function descriptions between parentheses the argument/parameter is required unless surrounded by *[]* brackets.

Functions

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.

Parameters: **user-abend-code** – specified User-Abend-Code

Return type: n/a

AFTER (`search-string`, `string`)

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

Parameters:
- **search-string** – search string
- **string** – string to search

Return type: string

A2E (`ascii-string`)

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

Parameters: **ascii-string** – string to translate

Return type: string

E2A (`ebcdic-string`)

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

Parameters: **ebcdic-string** – string to translate

Return type: string

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 string an empty string is returned.

Parameters:
- **search-string** – search string
- **string** – string to search

Return type: string

Example:

```
1 string='The quick brown fox jumps over the lazy dog'
2 say 'String                'string
3 say 'Before Fox            'before('fox',string)
4 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.

Parameters: **program-name** – program name
Return type: int

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.

Parameters: **string** – string to encode
Return type: string

BASE64DEC (base64-string)
Decodes a base64 string into a string or binary string.

Parameters: **base64-string** – string to decode
Return type: string

Example:

```
1 str='The quick brown fox jumps over the lazy dog'
2 stre=base64Enc(str)
3 say 'Encoded 'stre
4 strd=base64Dec(stre)
5 say 'Original "'strd'"'
6 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

Parameters: **bit-string** – string to decode
Return type: 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 than 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)
Encrypts a 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 an RSA encryption.

DECRYPT (string, password)
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 an RSA encryption.

Example:

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

Result:

```
original The quick brown fox jumps over the lazy dog
encrypted E361A8D7F001D537D0D6CDCAF9EFD83CCA00F984897FBD538AAF964CA80E2806D4310205CEFAC709C9
decrypted The quick brown fox jumps over the lazy dog
```

DEFINED ('variable-name')
Tests if variable or STEM exists, to avoid variable substitution, the variable-name must be enclosed in quotes. return values:

Return Value	Description
-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:

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

Result:

```
009B8148 (+00000000) | 009AE448 00000000 009AD99C 009BF020 | ..U.......R...0.
009B8158 (+00000010) | 00000000 00000000 009B2578 80000000 | ................
009B8168 (+00000020) | 0000FFFF 009C7C88 0013B908 00000000 | ......@h........
009B8178 (+00000030) | 40D871C0 009DA1E0 002C13C0 002C1434 |  Q.{...\...{....
009B8188 (+00000040) | 002C1434 002C30A8 00000085 009ADA3C | .......y...e....
009B8198 (+00000050) | 00000002 00158000 00262F88 4025EBD0 | ...........h ..}
009B81A8 (+00000060) | 00BF52C0 0027F268 4026306E 00000000 | ...{..2. ..>....
009B81B8 (+00000070) | 001D4FB8 00000000 00000000 009DC330 | ..|...........C.
009B81C8 (+00000080) | 00000000 009B8730 00000000 00000000 | ......g.........
009B81D8 (+00000090) | 001D3048 00000000 009DF548 00000000 | ..........5.....
009B81E8 (+000000A0) | 009B23C4 809D5F88 00000000 00000000 | ...D..¬h........
009B81F8 (+000000B0) | 00000000 009DC6EC 00000000 00000000 | ......F.........
009B8208 (+000000C0) | 00000000 00000000 00000000 00000000 | ................
009B8218 (+000000D0) | 009B8270 00000000 00000000 009B8730 | ..b...........g.
009B8228 (+000000E0) | 00000000 00000000 00000000 00000000 | ................
009B8238 (+000000F0) | 80000040 00000000 009B2E58 00000000 | ... ............
```

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:

```
1 v21.1='Stem Variable, item 1'
2 v21.2='Stem Variable, item 2'
3 v21.3='Stem Variable, item 3'
4 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 consistency of existing REXX scripts. It may be removed in a future release.

The three arguments are options. *date* defaults to today,

Supported input formats:

Format	Description
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, month is translated into month number (first 3 letters)
NORMAL	dd 3-letter-month yyyy e.g. 12 Mar 2018, 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:

Format	Description
Base	days since 01.01.0001
Days	ddd days in this year e.g. 257
Weekday	weekday of day e.g. Monday
Century	dddd days in this century
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, month is translated into month number (first 3 letters)
LS	time of day in microseconds 5 chars (digits) seconds, 6 chars, microseconds w/out delimiters
NORMAL	dd 3-letter-month yyyy e.g. 12 Mar 2018, 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

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

Formats a timestamp into various representations.

Parameters:

- **target-format** – *optional* target-format defaults to Ordered
- **timestamp** – *optional* timestamp defaults to today current time
- **input-format** – *optional* input-format defaults to Timestamp

Return type: string

Formats are:

Format	Description	Example
T	is 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

Time (`string`)
Time has gotten new input parameters. *String* can be one of:

- *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.

Parameters:
- **date-target-format** – *optional* date-target-format defaults to Ordered
- **timestamp** – *optional* timestamp defaults to today current time
- **input-format** – *optional* input-format defaults to Timestamp

Return type: string

FILTER (`string`, `character-table`**[**, `filter-type`**]**)

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

Parameters:
- **string** – string to filter
- **character-table** – filter table
- **filter-type** – *optional* either *drop* or *keep*

Return type: string For example, remove ‘o’ and ‘blank’: `1 say FILTER('The quick brown fox jumps over the lazy dog',' o') 2 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 currently running job or TSO session in REXX variables, like JOB.NAME, JOB.NUMBER, STEP.NAME, PROGRAM.NAME
Example:

```
1 say jobinfo()
2 say job.name
3 say job.number
4 say job.step
5 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.

Example:

```
SAY JOIN('     PETER        MUNICH','NAME=        CITY=          ')
NAME=PETER   CITY=MUNICH
```

LEVEL ()

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

Example:

```
 1 say 'Entering MAIN 'Level()
 2 call proc1
 3 say 'Returning from proc1 'Level()
 4 return
 5
 6 proc1:
 7    say 'Entering proc1 'Level()
 8    call proc2
 9    say 'Returning from proc2 'Level()
10 return 0
11
12 proc2: procedure
13    if level()>5 then return 4
14    say 'Entering proc2 'Level()
15    prc=proc1()
16    say 'Returning from proc1 'Level()
17 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:

```
 1 RX MAIN "EUROPE"
 2 call Sub1 "Germany", "Italy","UK"
 3 return
 4
 5 sub1:
 6 call sub2 'Munich','Rome','London'
 7 return
 8
 9 sub2:
10 say 'argument 1 of SUB2: 'argv(1,0)
11 say 'argument 2 of SUB2: 'argv(2,0)
12 say 'argument 3 of SUB2: 'argv(3,0)
13
14 say 'argument 1 of SUB1: 'argv(1,-1)
15 say 'argument 2 of SUB1: 'argv(2,-1)
16 say 'argument 3 of SUB1: 'argv(3,-1)
17
18 say 'Calling argument 1 of main: 'argv(1,-2)
19 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`)
Starts a load module. Parameters work according to standard conventions.

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:

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

Result:

```
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',['lock-modes'][,timeout]`)

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.

Parameters:

- **lock-string** – resource to lock
- **lock-modes** – *optional* One of TEST/SHARED/EXCLUSIVE. *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** – *optional* 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 resource was locked, **4** resource could not be acquired in the requested time interval

UNLOCK (`'lock-string'`)
Unlocks a previously locked resource.

Returns: 0 unlock was successful, otherwise unsuccesful

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 ofreturned trace table entries. The return code contains the number of trace table entries fetched. If **-1** is returned the Master Trace Table has not been changed since the last call, _LINE. remains unchanged.

If the optional *'REFRESH'* option is used, the Trace Table will be recreated even it it has not changed.
Example:

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

Result:

```
...
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.
Example:

```
/* ----------------------------------------------------------------
 * Scan Master Trace Table for LOGON/LOGOFF actions
 * ----------------------------------------------------------------
 */
/* ----------------------------------------------------------------*/
/*            + ---  REGISTER requestesd 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 millisedonds           */
/*            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 ()

An application that returns the output of a requested Console command in the stem variable *CONSOLE.n*

Returns: **>0** 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
```

Warning

The result of an operator command is not synchronously returned, but asynchronously assigned via the activity number. In certain situations, this may fail, then an exact match of 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.
Example:

```
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 ()

An application that returns the output of a requested NJE38 command in the stem variable *NJE38.n*

Returns: **>0** means the NJE38 command output could not be identified in the Master Trace Table

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

```
 1 /* ------------------------------------------------------------
 2  *  NJE38 Sample: Show available files in NJE38 inbox
 3  *    pass command to NJE38CMD and retrieve output
 4  * ------------------------------------------------------------
 5  */
 6 rc=nje38CMD('NJE38 D fILes')
 7 if rc>0 then do
 8    say 'Unable to pickup NJE38 results'
 9    return 8
10 end
11 say copies('-',72)
12 say center('NJE38 Spool Queue',72)
13 say copies('-',72)
14 do i=1 to nje38.0
15    say nje38.i
16 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, where they can have various compound components. The pattern must be coded in the form *p1.p2.p3.p4.p5*, p1, p2, p3,p4,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:

```
 1 ADDRESS.PEJ.CITY='Munich'
 2 ADDRESS.MIG.CITY='Berlin'
 3 ADDRESS.pej.pub='Hofbrauhaus'
 4 ADDRESS.mig.pub='Steakhaus'
 5 ADDRESS='set'
 6 call xlist('*.*.CITY')
 7 call xlist('ADDRESS')
 8 call xlist('ADDRESS.*.CITY')
 9 call xlist('ADDRESS.PEJ')
10 call xlist('ADDRESS.MIG')
11 call xlist()
12 exit
13 xlist:
14 say '>>> 'arg(1)
15 say vlist(arg(1))
16 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"
SIGL="11"
VLIST.0="2"
```

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, a 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.
Example:

```
Rotate("1234567890ABCDEF",10,10)
Rotate("1234567890ABCDEF",1)
Rotate("1234567890ABCDEF",5)
```

Result:

```
'0ABCDEF123'
'1234567890ABCDEF'
'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`)
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
- stack containing the JCL

Example:

```
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)
```

Warning

The internal reader has no knowledge of 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 store 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:

```
1 Say Split('The quick brown fox jumps over the lazy dog','myStem.')
2 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 list of word delimiters:

```
1 say split('City=London,Address=Picadelly Circus 24(7th floor)','mystem.','()=,')
2 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"
[0002]  "X" => "CITY=LONDON,ADDRESS=PICADELLY CIRCUS 24(7TH FLOOR)"
```

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

SPLIT splits a string into its words and store them in a stem variable. The split-string defines the string which shall be used to separate the words. SPLIT returns the number 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:

```
1 say splitbs('today</N>tomorrow</N>yesterday','mystem.','</N>')
2 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.
As calculation internally is done on integer fields, the maximum date which is supported is 19 January 2038 04:14:07. If no parameters are specified, the current date/time will be returned.
Example:

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

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
Example:

```
1 tstamp=EPOCHTIME()
2 say tstamp
3 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)
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

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

2. Create a sarray adding the lines to it:

```
 1 s1=screate(32)
 2 call sset(s1,,"A=0")
 3 call sset(s1,,"A=A+1")
 4 call sset(s1,,"A=A+1")
 5 call sset(s1,,"A=A+1")
 6 call sset(s1,,"A=A+1")
 7 call sset(s1,,"say a")
 8 call slist(s1)
 9 xset.1="c=0"
10 xset.2="c=c+1"
11 xset.3="d=c+5"
12 xset.4="e=c+15"
13 xset.5="say c d e"
```

```
14 xset.0=5
15 s2=stem2str("xset.")
16 say "STEMSTR "s2
17 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:

```
 1 DO forever
 2     if stcstop()=1 then do
 3        CALL WTO 'STC STOP COMMAND RECEIVED'
 4        leave
 5     end
 6     call wait 1000
 7     /* do STC stuff             */
 8 ...
 9     /* loop to check STC status */
10 end
11 exit 0
```

VERSION (**[**, 'FULL'**]**)
Returns BREXX/370 version information, if FULL is specified the Build Date of BREXX is added and returned.
Example:

```
SAY VERSION()       -> V2R5M3
SAY VERSION('FULL') -> Version V2R5M1 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:

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

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

```
3 say wordins('really','I love BREXX',3)
4 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:

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

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.

```
 1 call GetData
 2
 3 do i=1 to bands.0
 4    say i bands.i
 5 end
 6 Result
 7 1 LED ZEPPELIN                 STAIRWAY TO HEAVEN
 8 2 EAGLES                       HOTEL CALIFORNIA
 9 3 AC/DC                        BACK IN BLACK
10 4 JOURNEY                      DON'T STOP BELIEVIN'
11 5 PINK FLOYD                   ANOTHER BRICK IN THE WALL
12 6 QUEEN                        BOHEMIAN RHAPSODY
13 7 TOTO                         HOLD THE LINE
14 8 DEEP PURPLE                                  SMOKE ON THE WATER
```

LCS (‘string1’,’string2”)
Longest Common Subsequence. Find the Longest Common Subsequence of two strings.

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

Result:

```
tsitest
```

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:

```
 1 call setg('ctime',time('l'))
 2 call setg('city','Munich')
 3 call testproc
 4 exit 0
 5
 6 testproc: procedure
 7   /* normal variable scope can't access variables from the calling rexx */
 8   say 'Global Variables from the calling REXX'
 9   say getg('ctime')
10   say getg('city')
11 return 0
```

Result:

```
GLOBAL VARIABLES FROM THE CALLING REXX
19:45:12.538474
MUNICH
```

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 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”

Parameters: **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.

Returns: 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 as, space limitations, authorisation, etc.)
- -2 Dataset is already catalogued

Example:

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

DIR (`partitioned-dataset-name`)
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 directory is provided in the stem variable *DIRENTRY.*.
Table showing the structure of the returned stem. **n is the number of the member entry**.

STEM Name	Description
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

EXISTS (`dataset-name/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/partitioned-dataset(member)`)
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 Create was successful
- -1 Dataset cannot be created (various reasons as, space limitations, authorisation, etc.)
- -2 Dataset is already catalogued

The REMOVE function on members of a partitioned dataset removes the specified member (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.

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 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.
The RENAME function on members renames the specified member into a new one. The user requires the 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/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 creating new datasets with appropriate DCB and system definitions. If the dataset already exists, the existing definition is used, 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 parenthesis. e.g. *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.

Parameters: **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.

Warning

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'
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!
Syntax: *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)]*

EXECIO Param	Description
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

Example:

```
/* 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

TCP Functions

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

Warning

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.

Warning

In case of errors or ABENDs an automatic cleanup of open TCP sockets takes place. If in rare cases the cleanup cannot resolve it 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, else an error occurred.

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

Return	Description
#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:
Example:

```
 1 /* rexx */
 2 do forever
 3   event = tcpwait(20)
 4   if event <= 0 then call eventerror event
 5   select
 6     when event = #receive then do
 7       rc=receive()
 8       if rc=0 then iterate /* proceed */
 9       if rc=4 then leave /* close client socket */
10       if rc=8 then leave /* shut down server */
11     end
12     when event = #connect then call connect
13     when event = #timeout then call timeout
14     when event = #close then call close
15     when event = #stop then call close /* is /F console cmd */
16     when event = #error then call eventError
17     otherwise call eventError
18   end
19 end
```

TCPSEND (`clientToken`, `message`**[**, `timeout-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 occurred 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

Example: *SendLength=TCPSEND(clientToken, message[,time-out-secs])*

`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

Example: *MessageLength=TCPReceive(clientToken,[time-out-secs])*

`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:

Label	Description
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: - 0 proceed - 4 immediately close client
TCPCLOSE	client has closed the connection. TCPCLOSE can be used for housekeeping. ARG(1) client token Return codes from user procedure control the continuation: -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)*

TSO REXX Functions

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

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

Example:

```
1 x=SYSDSN("'HERC01.TEST.DATA'")
2 IF x = 'OK' THEN
3   do something
4 ELSE
5   do something other
```

SYSALC (`['DSN'/'DDN',dataset-name/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.
Examples:

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

Results:

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

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

Results:

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

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

Results:

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

SYSVAR (`request-type`)
TSO-only function to retrieve certain TSO runtime information. Available request-types:

Type	Description
SYSUID	UserID
SYSPREF	system prefix of current TSO session (typically hlq of userid)
SYSENV	FORE/BACK/BATCH forground/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 either MVS or VM/370
SYSCPLVL	shows the release of the host-system
SYSHEAP	allocated heap storage
SYSSTACK	allocated stack storage
RXINSTRC	BREXX Instruction Counter

Example:

```
1 say sysvar('SYSISPF')
2 say sysvar('SYSUID')
3 say sysvar('SYSPREF')
4 say sysvar('SYSENV')
5 say sysvar('SYSAUTH')
6 say sysvar('SYSCP')
7 say sysvar('SYSCPLVL')
8 say sysvar('RXINSTRC')
```

Result:

```
NOT ACTIVE
IBMUSER
IBMUSER
FORE
1
Hercules
Hercules version 4.4.1.10647-SDL-gd0ccfbc9
16
```

MVSVAR (request-type)

Return certain MVS information.

Type	Description
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

Example:

```
 1 Say MVSVAR('SYSNAME')
 2 SAY MVSVAR('SYSOPSYS')
 3 SAY MVSVAR('CPU')
 4 SAY MVSVAR('CPUS')
 5 SAY MVSVAR('NJE')
 6 SAY MVSVAR('NJEDSN')
 7 SAY MVSVAR(SYSNETID)
 8 SAY MVSVAR(SYSNJVER)
 9 SAY MVSVAR('MVSUP')
10 SAY sec2time(MVSVAR('MVSUP'),'DAYS')
```

Results:

```
MVSC
MVS 03.8
148
0002
1
NJE38.NETSPOOL
DRNBRX3A
V2.2.0 01/14/21 07.11
```

```
1339432
15 day(s) 12:03:52
```

LISTDSI (dataset)

Returns information of non-VSAM datasets in REXX variables.

Parameters: **dataset** – Either *“”dataset-name””* or *‘dd-name FILE’*

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

Variable	Description
SYSDSNAME	Dataset name
SYSVOLUME	Volume location
SYSDSORG	PS for sequential, PO for partitioned datasets
SYSRECFM	record format, F,FB,V,VB, …
SYSLRECL	record length
SYSBLKSIZE	block size
SYSSIZE	file size, For partitioned it is 0

LISTDSIX (dataset)

Parameters: **dataset** – Either *“”dataset-name””* or *‘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 and are suffixed with an X:

Variable	Description
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

Note

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:

Variable	Description
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.

Parameters: **option** – One of *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*.

`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.

`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

where *action* is one of:

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

Array functions

The BREXX Array functions are an implementation outside the REXX standard. They allow more direct access to array's items than compound variables (stems). The definition overhead is also negligible, allowing larger arrays as with stems. For performance reasons, the internal checking of boundaries, limits and content is kept at a basic level, if exceeded the REXX script will most likely end with a 0C4.

String Array Functions

Why String Arrays? There is a performance and storage overhead with stems, the stem name must be in a binary tree before the contents can be read. The allocation for content also contains a reserve in case a new version is a bit longer to avoid reallocation of the memory.

String Arrays have a pointer array addressing each content directly, adding a new item is therefore 2 times faster than in a stem, reading about 5 times faster. Another benefit is that you can easily add low-level functions (written in C) to work on the arrays directly. *SQSORT*, *SHSORT*, *SSEARCH*, *SSELECT* are some examples.

Managing Source Arrays

SCREATE (`size`)
Creates a Source Array, returned is the Source Array Number, which must be used in various Source Array functions. The size refers to the maximum number of entries of the array. Exceeding the maximum might lead to an 0C4 or other abends.
Depending on virtual storage availability, you can have up to 32 different arrays.
For example, see SGET.

Returns: the allocated array-number which can be used in subsequent array functions.

SSET (`array-number[, item-index], string-value`)
Sets a particular element of the array with a string value. The item index must not exceed the maximum size defined in the SCREATE function. If the item-index is not specified, the entry is added at the end of the array.
The item index must not exceed the maximum size defined in the SCREATE function. To minimise the overhead there is no checking of the limits in place. Exceeding it will cause an 0C4.
For example, see SGET.

SGET (`array-number, item-index[, offset]`)
Gets (returns) an element of the array as a string value. If an offset is defined the returned value starts at it.
The item index must not exceed the maximum size defined in the SCREATE function. To minimize the overhead there is no checking of the limits in place. Exceeding it will cause an 0C4.

```
 1 smax=15
 2 s1=screate(smax)
 3 do i=1 to smax
 4    call sset(s1,i,right(i,3,'0')'. Record')
 5 end
 6 do i=1 to smax
 7    say sget(s1,i)
 8 end
```

Result:

```
001. Record
002. Record
003. Record
004. Record
005. Record
006. Record
007. Record
008. Record
009. Record
010. Record
011. Record
012. Record
013. Record
014. Record
015. Record
```

SFREE (`array-number[, KEEP]`)
Removes the specified array and all its entries. All storage allocations are freed. If KEEP is specified all items are freed, but the array itself (array-number) remains allocated.

For example, see SWRITE.

Fast Compare and Swap Items

SSWAP (`array-number`, `item-number-1`, `item-number-2`)
Swaps the position of 2 elements in the array. As only pointers are moved a very fast function.

SCLC (`array-number`, `item-number-1`, `array-number-2`, `item-number-2`)
Compares 2 elements of the 2 arrays. SCLC is much faster than loading both items and comparing it to the REXX level. To compare items within one array just use for *array-number2* the value of *array-number-1*.

Returns: <0 if item-1 < item-2 0 if item-1 < item-2 >0 if item-1 > item-2

Example:

```
1 smax=15
2 s1=screate(smax)
3 do i=1 to smax
4    call sset(s1,i,right(i,3,'0')'. Record')
5 end
6 do i=1 to smax
7    say "Compare item "i" and 8, result: "sclc(s1,i,s1,8)
8 end
```

Results:

```
Compare item 1 and 8, result: -7
Compare item 2 and 8, result: -6
Compare item 3 and 8, result: -5
Compare item 4 and 8, result: -4
Compare item 5 and 8, result: -3
Compare item 6 and 8, result: -2
Compare item 7 and 8, result: -1
Compare item 8 and 8, result: 0
Compare item 9 and 8, result: 1
Compare item 10 and 8, result: 1
Compare item 11 and 8, result: 1
Compare item 12 and 8, result: 1
Compare item 13 and 8, result: 1
Compare item 14 and 8, result: 1
Compare item 15 and 8, result: 1
```

Sorting and Merging Arrays

SQSORT (`array-number`**[**, `ASCENDING/DESCENDING`**][**, `sort-offset`**]**)

Sorts an array using the quick sort algorithm in ascending or descending order, the default is ascending. The sort-offset defines the sorting scope up to the end of the item, any substrings before it are not treated. If you define for example 5, the array is sorted at offset 5 (up to the rest of the item). The sort-offset defaults to 1.

This sort is 100-150 times faster than the BREXX quick sort running on stems.

returns: the number of sorted items.

Example:

```
1 max=25
2 s1=SREAD("'pej.songs2'")
3 call slist s1
4 call sqsort(s1,'ASC',26)/* Sort Array S1 beginning column 26(song name) */
5 call slist s1           /* display sorted array */
6 call sfree s1
```

Result, song names are in sorted order:

```
     Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   LED ZEPPELIN                  STAIRWAY TO HEAVEN
00002   EAGLES                        HOTEL CALIFORNIA
00003   AC/DC                         BACK IN BLACK
00004   JOURNEY                       DON'T STOP BELIEVIN'
00005   PINK FLOYD                    ANOTHER BRICK IN THE WALL
00006   QUEEN                         BOHEMIAN RHAPSODY
00007   TOTO                          HOLD THE LINE
00008   KISS                          I WAS MADE FOR LOVIN' YOU
00009   BON JOVI                      LIVIN' ON A PRAYER
00010   NIRVANA                       SMELLS LIKE TEEN SPIRIT
00011   DEEP PURPLE                   SMOKE ON THE WATER
00012   METALLICA                     NOTHING ELSE MATTERS
00013   THE ROLLING STONES            (I CAN'T GET NO) SATISFACTION
00014   BRUCE SPRINGSTEEN             BORN IN THE U.S.A.
00015   QUEEN                         WE WILL ROCK YOU
00016   LYNYRD SKYNYRD                FREE BIRD
00017   SURVIVOR                      EYE OF THE TIGER
00018   THE CLASH                     SHOULD I STAY OR SHOULD I GO
00019   JIMI HENDRIX                  HEY JOE
00020   FLEETWOOD MAC                 LITTLE LIES
00021   AC/DC                         HIGHWAY TO HELL
00022   THE POLICE                    ROXANNE

     Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   THE ROLLING STONES            (I CAN'T GET NO) SATISFACTION
00002   PINK FLOYD                    ANOTHER BRICK IN THE WALL
00003   AC/DC                         BACK IN BLACK
00004   QUEEN                         BOHEMIAN RHAPSODY
00005   BRUCE SPRINGSTEEN             BORN IN THE U.S.A.
00006   JOURNEY                       DON'T STOP BELIEVIN'
00007   SURVIVOR                      EYE OF THE TIGER
00008   LYNYRD SKYNYRD                FREE BIRD
00009   JIMI HENDRIX                  HEY JOE
00010   AC/DC                         HIGHWAY TO HELL
00011   TOTO                          HOLD THE LINE
00012   EAGLES                        HOTEL CALIFORNIA
00013   KISS                          I WAS MADE FOR LOVIN' YOU
00014   FLEETWOOD MAC                 LITTLE LIES
00015   BON JOVI                      LIVIN' ON A PRAYER
00016   METALLICA                     NOTHING ELSE MATTERS
00017   THE POLICE                    ROXANNE
00018   THE CLASH                     SHOULD I STAY OR SHOULD I GO
00019   NIRVANA                       SMELLS LIKE TEEN SPIRIT
00020   DEEP PURPLE                   SMOKE ON THE WATER
00021   LED ZEPPELIN                  STAIRWAY TO HEAVEN
00022   QUEEN                         WE WILL ROCK YOU
```

SHSORT (`array-number`**[**, `ASCENDING/DESCENDING`**][**, `sort-offset`**]**)
Sorts an array using the shell sort algorithm in ascending or descending order, default is ascending.
The sort-offset defines the sorting scope up to the end of the item, any substrings before it are not treated. If you define for example 5, the array is sorted at offset 5 (up to the rest of the item). The sort-offset defaults to 1.
This sort is 100-150 times faster than the BREXX shell sort running on stems.

SMERGE (`array-number-1`, `array-number-2`)
Merges 2 arrays into a new array, based on their sort order.

Returns: the number of merged items.

Example:

```
max=10
s1=SCREATE(max)       /* Create a String Array called S1 */
s2=SCREATE(max)       /* Create a String Array called S2 */
do i=1 to max
   call sset(s1,i,right((max-i+1),4,'0')' A Record') /* Add new Record in Array S1 at p
   call sset(s2,i,right((max-i+1),4,'0')' B Record') /* Add new Record in Array S1 at p
end
say "Source Array S1"
say "---------------"
call slist s1
say "Source Array S2"
say "---------------"
call slist s2
s3=smerge(s1,s2)           /* Merge Array S1 and S2 into S3 */
say "Source Array S3"
say "---------------"
call slist s3
return
```

Result:

```
Source Array S1
---------------
00001   0010 A Record
00002   0009 A Record
00003   0008 A Record
00004   0007 A Record
00005   0006 A Record
00006   0005 A Record
00007   0004 A Record
00008   0003 A Record
00009   0002 A Record
00010   0001 A Record
Source Array S2
---------------
00001   0010 B Record
00002   0009 B Record
00003   0008 B Record
00004   0007 B Record
00005   0006 B Record
00006   0005 B Record
00007   0004 B Record
00008   0003 B Record
00009   0002 B Record
00010   0001 B Record
Source Array S3
---------------
00001   0001 A Record
00002   0001 B Record
00003   0002 A Record
00004   0002 B Record
00005   0003 A Record
00006   0003 B Record
00007   0004 A Record
00008   0004 B Record
00009   0005 A Record
00010   0005 B Record
00011   0006 A Record
```

```
00012   0006 B Record
00013   0007 A Record
00014   0007 B Record
00015   0008 A Record
00016   0008 B Record
00017   0009 A Record
00018   0009 B Record
00019   0010 A Record
00020   0010 B Record
```

Reporting and Manipulating entire Array

SREVERSE (`array-number`)
reverses the order of an array, the first item becomes the last item, the last item the first item, etc. The reverse takes place in the specified array. There is no new array created. The reverse process is very quick as just the string addresses are swapped, not the string content.

Returns: the number of elements of the array.

Example:

```
 1 smax=10
 2 s1=screate(smax)
 3 do i=1 to smax
 4    call sset(s1,i,right(i,6,'0')". Record")
 5 end
 6 say "Original"
 7 say "--------"
 8 call slist s1
 9 call sreverse(s1)
10 say "Reversed"
11 say "--------"
12 call slist s1
13 call sfree(s1)
14 EXIT 0
```

Result:

```
Original
--------
00001   000001. Record
00002   000002. Record
00003   000003. Record
00004   000004. Record
00005   000005. Record
00006   000006. Record
00007   000007. Record
00008   000008. Record
00009   000009. Record
00010   000010. Record
Reversed
--------
00001   000010. Record
00002   000009. Record
00003   000008. Record
00004   000007. Record
00005   000006. Record
00006   000005. Record
00007   000004. Record
00008   000003. Record
00009   000002. Record
00010   000001. Record
```

SWRITE (array-number, dsn/ddname)
Writes all entries of the specified array into an external dataset.
The dataset can be either a fully qualified Dataset Name or a pre-allocated DD Name.

Returns: the number of written entries.

For example, see SREAD.

SREAD (dsn/ddname<, size-of-array>)
Reads all entries of an external dataset into a new String Array. The dataset can be either a fully qualified Dataset Name or a pre-allocated DD Name. The optional parameter size-of-array is recommended for large datasets. If omitted the size of the array grows dynamically to accommodate the content.

Returns: the newly created Array number.

Example:

```
1 s1=sread("'pej.songs'")     /* import CSV formatted DSN */
2 s2=screate(sarray(s1))      /* create formatted version */
3 do i=1 to sarray(s1)
4    parse value sget(s1,i) with band':'song
5    call sset(s2,i,left(band,25)song)
6 end
7 call slist s2
8 say swrite(s2,"'pej.songs2'")' Entries exported'
```

The contents of pej.songs, list of 20 best rock songs (not rated by me):

```
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
KISS:     I WAS MADE FOR LOVIN' YOU
BON JOVI:     LIVIN' ON A PRAYER
NIRVANA:     SMELLS LIKE TEEN SPIRIT
DEEP PURPLE:     SMOKE ON THE WATER
METALLICA:     NOTHING ELSE MATTERS
THE ROLLING STONES:     (I CAN'T GET NO) SATISFACTION
BRUCE SPRINGSTEEN:     BORN IN THE U.S.A.
QUEEN:     WE WILL ROCK YOU
LYNYRD SKYNYRD:     FREE BIRD
SURVIVOR:     EYE OF THE TIGER
THE CLASH:     SHOULD I STAY OR SHOULD I GO
JIMI HENDRIX:     HEY JOE
FLEETWOOD MAC:     LITTLE LIES
AC/DC:     HIGHWAY TO HELL
THE POLICE:     ROXANNE
Result of fetched DSN:
    Entries of Source Array: 1
Entry   Data
-----------------------------------------------------
00001   LED ZEPPELIN                  STAIRWAY TO HEAVEN
00002   EAGLES                        HOTEL CALIFORNIA
00003   AC/DC                         BACK IN BLACK
00004   JOURNEY                       DON'T STOP BELIEVIN'
00005   PINK FLOYD                    ANOTHER BRICK IN THE WALL
00006   QUEEN                         BOHEMIAN RHAPSODY
00007   TOTO                          HOLD THE LINE
00008   KISS                          I WAS MADE FOR LOVIN' YOU
00009   BON JOVI                      LIVIN' ON A PRAYER
00010   NIRVANA                       SMELLS LIKE TEEN SPIRIT
```

```
00011   DEEP PURPLE                 SMOKE ON THE WATER
00012   METALLICA                   NOTHING ELSE MATTERS
00013   THE ROLLING STONES          (I CAN'T GET NO) SATISFACTION
00014   BRUCE SPRINGSTEEN           BORN IN THE U.S.A.
00015   QUEEN                       WE WILL ROCK YOU
00016   LYNYRD SKYNYRD              FREE BIRD
00017   SURVIVOR                    EYE OF THE TIGER
00018   THE CLASH                   SHOULD I STAY OR SHOULD I GO
00019   JIMI HENDRIX                HEY JOE
00020   FLEETWOOD MAC               LITTLE LIES
00021   AC/DC                       HIGHWAY TO HELL
00022   THE POLICE                  ROXANNE
22 Entries exported
```

SLIST (`array-number`**`[`**, `from`**`][`**, `to`**`][`**, `heading`**`]`**)
Prints the array content. With the optional from and to parameters, you can limit the range of entries to be printed. The optional heading parameter is printed in the heading line.
For example, see SREAD and others

SSEARCH (`array-number`, `search-string`, `from`**`[`**, `"option"`**`]`**)
Searches in a String Array for a certain string and returns the index number. For repeated searches, you can use the from parameter.

Parameters: **option** – *CASE* - means it is a case-sensitive search *NOCASE* - search is case-insensitive
Returns: index position if found, or zero.

Example:

```
1 s1=sread("'pej.songs2'")
2 ssc="ON"
3 ssi=ssearch(s1,ssc) /* Search string ON in array */
4 do while ssi>0
5    say "Found at "ssi": "sget(s1,ssi)
6    ssi=ssearch(s1,ssc,ssi+1)
7 end
```

Result:

```
Found at 4: JOURNEY                     DON'T STOP BELIEVIN'
Found at 9: BON JOVI                    LIVIN' ON A PRAYER
Found at 11: DEEP PURPLE                 SMOKE ON THE WATER
Found at 13: THE ROLLING STONES          (I CAN'T GET NO) SATISFACTION
```

SSEARCHI (`array-number`, `search-string`, `from`**`[`**, `"CASE"/"NOCASE"`**`]`**)
Searches in a String Array for a certain string and returns all occurrences in an internally created Integer Array. It is consecutively filled with the index referring to the String Array.

Parameters:
- **from** – line number to start the search
- **option** – *CASE* - means it is a case-sensitive search *NOCASE* - search is case-insensitive

Returns: the Integer Array number. Additionally, the variable *SCOUNT* is set with the number of entries found.

It is recommended to free the Integer Array if not needed anymore.
Example:

```
1 imax=50
2 s1=Screate(imax)
3 do i=1 to imax/2
4    call sset(s1,,'abcde'i)
5    call sset(s1,,'xyz'i)
6 end
7 i1=ssearchI(s1,'xyz')
```

```
 8 say 'found 'scount
 9 call ilist i1,20,25
10 call ifree i1
```

Result:

```
found 25
 Entries of IARRAY: 0
Entry   Data
-------------------------------------------------------
00020           40
00021           42
00022           44
00023           46
00024           48
00025           50
25 Entries
```

SSELECT (`array-number`, `search-1`**[**, `search-2`, …, `search-99`**]**)

Creates a subset of the array when an entry matches one of the specified search strings in a new array. There are up to 99 search strings allowed. The search is case-sensitive.

Returns: the newly created array.

Example:

```
1 s1=sread("'pej.songs2'")
2 call slist s1
3 s2=sselect(s1,'ON','OF','EE')   /* Search ON, OF, EE in array */
4 say copies('-',32)
5 say 'Selected'
6 say copies('-',32)
7 call slist s2
8 call sfree(s1)
9 call sfree(s2)
```

Result:

```
    Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   LED ZEPPELIN                    STAIRWAY TO HEAVEN
00002   EAGLES                          HOTEL CALIFORNIA
00003   AC/DC                           BACK IN BLACK
00004   JOURNEY                         DON'T STOP BELIEVIN'
00005   PINK FLOYD                      ANOTHER BRICK IN THE WALL
00006   QUEEN                           BOHEMIAN RHAPSODY
00007   TOTO                            HOLD THE LINE
00008   KISS                            I WAS MADE FOR LOVIN' YOU
00009   BON JOVI                        LIVIN' ON A PRAYER
00010   NIRVANA                         SMELLS LIKE TEEN SPIRIT
00011   DEEP PURPLE                     SMOKE ON THE WATER
00012   METALLICA                       NOTHING ELSE MATTERS
00013   THE ROLLING STONES              (I CAN'T GET NO) SATISFACTION
00014   BRUCE SPRINGSTEEN               BORN IN THE U.S.A.
00015   QUEEN                           WE WILL ROCK YOU
00016   LYNYRD SKYNYRD                  FREE BIRD
00017   SURVIVOR                        EYE OF THE TIGER
00018   THE CLASH                       SHOULD I STAY OR SHOULD I GO
00019   JIMI HENDRIX                    HEY JOE
00020   FLEETWOOD MAC                   LITTLE LIES
00021   AC/DC                           HIGHWAY TO HELL
00022   THE POLICE                      ROXANNE
```

```
------------------------------
Selected
------------------------------
    Entries of Source Array: 1
Entry   Data
-----------------------------------------------------
00001   JOURNEY                         DON'T STOP BELIEVIN'
00002   QUEEN                           BOHEMIAN RHAPSODY
00003   BON JOVI                        LIVIN' ON A PRAYER
00004   NIRVANA                         SMELLS LIKE TEEN SPIRIT
00005   DEEP PURPLE                     SMOKE ON THE WATER
00006   THE ROLLING STONES              (I CAN'T GET NO) SATISFACTION
00007   BRUCE SPRINGSTEEN               BORN IN THE U.S.A.
00008   QUEEN                           WE WILL ROCK YOU
00009   LYNYRD SKYNYRD                  FREE BIRD
00010   SURVIVOR                        EYE OF THE TIGER
00011   FLEETWOOD MAC                   LITTLE LIES
```

SCHANGE (`array-number`, `from-1`, `to-1`**`[`**, `from-2`, `to-2`**`[`**, `from-3`, `to-3`**`]]`**)

Changes the content of the array (line by line), from-1 is replaced by to1, from-2 by to-2, etc. If multiple change parameters are specified, a subsequent change may re-change a previous change.

Returns: the number of changes performed.

Example, input file is the same as in SSELECT:

```
 1 SAY COPIES('-',50)
 2 SAY "READ EXTERNAL INTO SARRAY, SELECT SUBSET"
 3 SAY COPIES('-',50)
 4 DSNIN=MVSVAR("REXXDSN")
 5 S1=SREAD("'"DSNIN"(LLDATA)'")      /* READ DATA */
 6 SAY SCHANGE(S1,'IN','**','EE','+++','EY','')
 7 SAY COPIES('-',50)
 8 SAY 'CHANGED ARRAY '
 9 SAY COPIES('-',50)
10 CALL SLIST S1
11 CALL SFREE S1
12 EXIT 0
```

Result:

```
--------------------------------------------------
CHANGED ARRAY
--------------------------------------------------
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   LED ZEPPEL**                    STAIRWAY TO HEAVEN
00002   EAGLES                          HOTEL CALIFORNIA
00003   AC/DC                           BACK ** BLACK
00004   JOURN                          DON'T STOP BELIEV**'
00005   P**K FLOYD                      ANOTHER BRICK ** THE WALL
00006   QU+++N                           BOHEMIAN RHAPSODY
00007   TOTO                            HOLD THE L**E
00008   KISS                            I WAS MADE FOR LOV**' YOU
00009   BON JOVI                        LIV**' ON A PRAYER
00010   NIRVANA                         SMELLS LIKE T+++N SPIRIT
00011   D+++P PURPLE                     SMOKE ON THE WATER
00012   METALLICA                       NOTH**G ELSE MATTERS
00013   THE ROLL**G STONES              (I CAN'T GET NO) SATISFACTIO
00014   BRUCE SPR**GST+++N               BORN ** THE U.S.A.
00015   QU+++N                           WE WILL ROCK YOU
```

```
00016   LYNYRD SKYNYRD                  FR+++ BIRD
00017   SURVIVOR                        E OF THE TIGER
00018   THE CLASH                       SHOULD I STAY OR SHOULD I GO
00019   JIMI HENDRIX                    H JOE
00020   FL+++TWOOD MAC                   LITTLE LIES
00021   AC/DC                           HIGHWAY TO HELL
00022   THE POLICE                      ROXANNE
22 Entries
```

SCOPY (`source-array[, from-entry][, to-entry][, old-array-to append][, start-position(from-array)][, length of substring]`)

Copies a source array into a new array, optionally you can append an existing array if you specify an existing array as 4. Parameter. Appending can also be done by the *SAPPEND* function.

Parameters:

- **from-entry** – defines an item number where the copy should begin. It defaults to 1.
- **to-entry** – is the item number to stop the copy. It defaults to the last array item.
- **start-position** – defines the substring offset, of the record to be copied. It defaults to 1.
- **substring** (*Length-of*) – is the length of the substring offset. It defaults to the length of the record.

Returns: the created array number

SAPPEND (`array-to-append, source-array[, from-entry][, to-entry][, start-position(from-array)][, length of substring]`)

Appends the array-to-append by the source-array.

Parameters:

- **from-entry** – defines the item number of the source-array which is taken as the first record to append. It defaults to 1.
- **to-entry** – is the item number to stop the append. It defaults to the last array item.
- **start-position** – defines the substring offset, of the record to be copied. It defaults to 1.
- **substring** (*Length-of*) – is the length of the substring offset. It defaults to the length of the record.

Returns: the appended array number.

Append the same array (items 10-20) to the array, input file is the same as in SSELECT:

```
 1 say copies('-',50)
 2 say "Read External into Sarray"
 3 say copies('-',50)
 4 dsnin=mvsvar("REXXDSN")
 5 s1=sread("'"dsnin"(lldata)'")
 6 call sAPPEND(s1,s1,10,20)
 7 say copies('-',50)
 8 say 'Appended Array by itself from entry to 20'
 9 say copies('-',50)
10 call slist s1
11 call sfree s1
12 EXIT 0
```

Result:

```
--------------------------------------------------
Appended Array by itself from entry to 20
--------------------------------------------------
    Entries of Source Array: 0
Entry   Data
------------------------------------------------------
00001   LED ZEPPELIN                    STAIRWAY TO HEAVEN
00002   EAGLES                          HOTEL CALIFORNIA
```

```
00003   AC/DC                       BACK IN BLACK
00004   JOURNEY                     DON'T STOP BELIEVIN'
00005   PINK FLOYD                  ANOTHER BRICK IN THE WALL
00006   QUEEN                       BOHEMIAN RHAPSODY
00007   TOTO                        HOLD THE LINE
00008   KISS                        I WAS MADE FOR LOVIN' YOU
00009   BON JOVI                    LIVIN' ON A PRAYER
00010   NIRVANA                     SMELLS LIKE TEEN SPIRIT
00011   DEEP PURPLE                 SMOKE ON THE WATER
00012   METALLICA                   NOTHING ELSE MATTERS
00013   THE ROLLING STONES          (I CAN'T GET NO) SATISFACTION
00014   BRUCE SPRINGSTEEN           BORN IN THE U.S.A.
00015   QUEEN                       WE WILL ROCK YOU
00016   LYNYRD SKYNYRD              FREE BIRD
00017   SURVIVOR                    EYE OF THE TIGER
00018   THE CLASH                   SHOULD I STAY OR SHOULD I GO
00019   JIMI HENDRIX                HEY JOE
00020   FLEETWOOD MAC               LITTLE LIES
00021   AC/DC                       HIGHWAY TO HELL
00022   THE POLICE                  ROXANNE
00023   NIRVANA                     SMELLS LIKE TEEN SPIRIT
00024   DEEP PURPLE                 SMOKE ON THE WATER
00025   METALLICA                   NOTHING ELSE MATTERS
00026   THE ROLLING STONES          (I CAN'T GET NO) SATISFACTION
00027   BRUCE SPRINGSTEEN           BORN IN THE U.S.A.
00028   QUEEN                       WE WILL ROCK YOU
00029   LYNYRD SKYNYRD              FREE BIRD
00030   SURVIVOR                    EYE OF THE TIGER
00031   THE CLASH                   SHOULD I STAY OR SHOULD I GO
00032   JIMI HENDRIX                HEY JOE
00033   FLEETWOOD MAC               LITTLE LIES
33 Entries
```

SSUBSTR (`array-number`, `from-column`**[**, `length`**][**, `INTERNAL/EXTERNAL`**]**)
Creates an array with the substring of each line (according to the SUBSTR REXX function). EXTERNAL (default) creates a new array with the substring results. INTERNAL works on the existing array.

Returns: the array number that has been created/used.

Example, the input file is the same as in SSELECT:

```
 1 say copies('-',50)
 2 say "Read External into Sarray, select Subset"
 3 say copies('-',50)
 4 dsnin=mvsvar("REXXDSN")
 5 s1=sread("'"dsnin"(lldata)'")      /* Create Linked List */
 6 /* call slist s1   */
 7 call sSUBSTR(s1,25,,'INTERNAL')
 8 say copies('-',50)
 9 say 'ARRAY from Column 25 '
10 say copies('-',50)
11 call slist s1
12 call sfree s1
13 EXIT 0
```

Result:

```
--------------------------------------------------
ARRAY from Column 25
--------------------------------------------------
    Entries of Source Array: 0
Entry   Data
```

```
-----------------------------------------------------
00001         STAIRWAY TO HEAVEN
00002         HOTEL CALIFORNIA
00003         BACK IN BLACK
00004         DON'T STOP BELIEVIN'
00005         ANOTHER BRICK IN THE WALL
00006         BOHEMIAN RHAPSODY
00007         HOLD THE LINE
00008         I WAS MADE FOR LOVIN' YOU
00009         LIVIN' ON A PRAYER
00010         SMELLS LIKE TEEN SPIRIT
00011         SMOKE ON THE WATER
00012         NOTHING ELSE MATTERS
00013         (I CAN'T GET NO) SATISFACTION
00014         BORN IN THE U.S.A.
00015         WE WILL ROCK YOU
00016         FREE BIRD
00017         EYE OF THE TIGER
00018         SHOULD I STAY OR SHOULD I GO
00019         HEY JOE
00020         LITTLE LIES
00021         HIGHWAY TO HELL
00022         ROXANNE
22 Entries
```

SUPPER (`array-number`**`[`**, `INTERNAL/EXTERNAL`**`]`**)

Creates/updates an array with the upper case version of each entry. EXTERNAL (default) creates a new array with the substring results. INTERNAL works on the existing array.

Returns: the array number that has been created/used.

```
1 sm1=15
2 s1=screate(sm1)
3 do i=1 to sm1
4    call sset(s1,,'abcdefghij 'i)
5 end
6 call slist s1
7 call supper(s1,'INTERNAL')
8 call slist s1
```

Result:

```
     Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   abcdefghij 1
00002   abcdefghij 2
00003   abcdefghij 3
00004   abcdefghij 4
00005   abcdefghij 5
00006   abcdefghij 6
00007   abcdefghij 7
00008   abcdefghij 8
00009   abcdefghij 9
00010   abcdefghij 10
00011   abcdefghij 11
00012   abcdefghij 12
00013   abcdefghij 13
00014   abcdefghij 14
00015   abcdefghij 15
15 Entries
```

```
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   ABCDEFGHIJ 1
00002   ABCDEFGHIJ 2
00003   ABCDEFGHIJ 3
00004   ABCDEFGHIJ 4
00005   ABCDEFGHIJ 5
00006   ABCDEFGHIJ 6
00007   ABCDEFGHIJ 7
00008   ABCDEFGHIJ 8
00009   ABCDEFGHIJ 9
00010   ABCDEFGHIJ 10
00011   ABCDEFGHIJ 11
00012   ABCDEFGHIJ 12
00013   ABCDEFGHIJ 13
00014   ABCDEFGHIJ 14
00015   ABCDEFGHIJ 15
15 Entries
```

SCOUNT (`array-number`, `search-string-1`**[**, `search-string-2`**[**, `search-string-3`…**]]**)
Counts the lines containing the search strings. Multiple occurrences of a search string in a line are not counted, but hits of additional search strings on a line will be counted.

Returns: the number of lines containing the search strings

SDROP (`array-number`, `drop-string-1`**[**, `drop-string-2`**[**, `drop-string-3`…**]]**)
Drops lines containing the drop strings at any position.
Alternatively, you can set the rexx -variable *sdrop.at.n=offset* to a certain offset to enforce an exact match. n refers to drop-string-n for which the search and drop should be performed. If for a certain position no sdrop.at variable is set, the search and drop is performed for any position in the line.
There can be up to 99 drop-strings.
An empty drop string is treated to drop empty lines.
The function works in an existing array (array-number) and may reduce the maximum number of items.

Returns: the number of items containing the drop strings

Example, the input file is the same as in SSELECT:

```
1 dsnin=mvsvar("REXXDSN")
2 s1=sread(""dsnin"(lldata)'")      /* Create Linked List */
3 call sDROP(s1,'AC','IN')
4 say copies('-',50)
5 say 'Items not containing AC or IN'
6 say copies('-',50)
7 call slist s1
8 call sfree s1
9 EXIT 0
```

Result:

```
--------------------------------------------------
Items not containing AC or IN
--------------------------------------------------
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   EAGLES                          HOTEL CALIFORNIA
00002   QUEEN                           BOHEMIAN RHAPSODY
00003   NIRVANA                         SMELLS LIKE TEEN SPIRIT
00004   DEEP PURPLE                     SMOKE ON THE WATER
00005   QUEEN                           WE WILL ROCK YOU
```

```
00006   LYNYRD SKYNYRD                  FREE BIRD
00007   SURVIVOR                        EYE OF THE TIGER
00008   THE CLASH                       SHOULD I STAY OR SHOULD I GO
00009   JIMI HENDRIX                    HEY JOE
00010   THE POLICE                      ROXANNE
10 Entries
```

SKEEP (`array-number`, `keep-string-1`**[**, `keep-string-2`**[**, `keep-string-3…`**]]**)
Keeps lines containing one or more of the specified keep strings.
The function works in the existing array (array-number) and may reduce the maximum number of items.

Returns: the number of lines containing the search strings.

Example, the input file is the same as in SSELECT:

```
 1 dsnin=mvsvar("REXXDSN")
 2 s1=sread("'"dsnin"(lldata)'")
 3 call slist s1
 4 call sKEEP(s1,'AC','IN')
 5 say copies('-',50)
 6 say 'Items containing AC or IN'
 7 say copies('-',50)
 8 call slist s1
 9 call sfree s1
10 EXIT 0
```

Result:

```
--------------------------------------------------
Items containing AC or IN
--------------------------------------------------
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   LED ZEPPELIN                    STAIRWAY TO HEAVEN
00002   AC/DC                           BACK IN BLACK
00003   JOURNEY                         DON'T STOP BELIEVIN'
00004   PINK FLOYD                      ANOTHER BRICK IN THE WALL
00005   TOTO                            HOLD THE LINE
00006   KISS                            I WAS MADE FOR LOVIN' YOU
00007   BON JOVI                        LIVIN' ON A PRAYER
00008   METALLICA                       NOTHING ELSE MATTERS
00009   THE ROLLING STONES              (I CAN'T GET NO) SATISFACTION
00010   BRUCE SPRINGSTEEN               BORN IN THE U.S.A.
00011   FLEETWOOD MAC                   LITTLE LIES
00012   AC/DC                           HIGHWAY TO HELL
12 Entries
```

SKEEPAND (`array-number`, `keep-string-1`**[**, `keep-string-2`**[**, `keep-string-3…`**]]**)
Keeps lines containing all of the specified keep strings.
The function works in the existing array (array-number) and may reduce the maximum number of items.

Returns: the number of lines containing the search strings.

Example, the input file is the same as in SSELECT:

```
 1 dsnin=mvsvar("REXXDSN")
 2 s1=sread("'"dsnin"(lldata)'")      /* Create Linked List */
 3 call slist s1
 4 call sKEEPAND(s1,'AC','IN')
 5 say copies('-',50)
 6 say 'Items containing AC AND IN'
 7 say copies('-',50)
 8 call slist s1
```

```
 9 call sfree s1
10 EXIT 0
```

Result:

```
-------------------------------------------------
Items containing AC AND IN
-------------------------------------------------
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   AC/DC                          BACK IN BLACK
00002   THE ROLLING STONES             (I CAN'T GET NO) SATISFACTION
2 Entries
```

SINSERT (`array-number`, `insert-after-line`, `number-of-lines`)
Inserts empty lines in an array. insert-after-line identifies the index after which the lines will be inserted. If you specify 0 they will be inserted starting with the first line. Existing lines will be shifted down by the requested number of lines.

Returns: the completion code. Zero means the insertion was successful. Return code 8 means the insertion failed, it is accompanied by an error message.

SDEL (`array-number`, `delete-from`, `number-lines`)
Deletes from the given line-number delete-from the specified number in number-lines.

SNUMBER (`source-array-number`**[**, `number-length`**]**)
Adds a line number in front of each item of an existing array. The optional number length can be specified as a seconds parameter, it defaults to 6.

```
1 s1=screate(10)
2 do i=1 to 10
3    call sset(s1,,'record 'i)
4 end
5 call slist s1
6 call snumber(s1,4)
7 call slist s1
```

Result:

```
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   record 1
00002   record 2
00003   record 3
00004   record 4
00005   record 5
00006   record 6
00007   record 7
00008   record 8
00009   record 9
00010   record 10
10 Entries
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   0001 record 1
00002   0002 record 2
00003   0003 record 3
00004   0004 record 4
00005   0005 record 5
00006   0006 record 6
```

```
00007   0007 record 7
00008   0008 record 8
00009   0009 record 9
00010   0010 record 10
```

S2HASH (source-array-number)

Creates an integer array. The items contain a hash value of each item in the source-array. Integer comparisons are much faster than string comparisons.

Returns: the integer array number.

Example, the input file is the same as in SSELECT:

```
 1 dsnin=mvsvar("REXXDSN")
 2 s1=sread("'"dsnin"(lldata)'")      /* Create Linked List */
 3 call slist s1
 4 i1=s2HASH(s1,25,,'INTERNAL')
 5 say copies('-',50)
 6 say 'Hashes from ARRAY'
 7 say copies('-',50)
 8 call ilist i1
 9 call sfree s1
10 call ifree i1
11 EXIT 0
```

Result:

```
--------------------------------------------------
Hashes from ARRAY
--------------------------------------------------
    Entries of IARRAY: 0
Entry   Data
--------------------------------------------------------------------------
00001   1762230186
00002    742146533
00003     54160282
00004    333486878
00005   1063451074
00006   1131679241
00007   1557910634
00008   -562308514
00009   1574372382
00010   1488490451
00011   -254564903
00012      8800208
00013   1324062534
00014   1821602997
00015   2118017364
00016   1965748661
00017   1329580209
00018   1871612546
00019   1881722666
00020   1061408928
00021   1598195414
00022   -238030808
22 Entries
```

SSPLIT (string-to-split, delimiter-chars)

SPLIT splits a string into lines and stores them in a SARRAY. The optional delimiter table defines the split character(s), which shall be used to separate the lines. The delimiter string may consist of more than one character. This function is useful if you have file content in one string containing the line-feed character.

Returns: the array number created.

SEXTRACT (`array-number`, `begin-lino`, `end-lino`)
SEXTRACT extracts lines of a SARRAY. The first parameter is the line to begin, second is the last line to be extracted, it is not the number of lines. End-lino defaults to the last line of the source array.

SCUT (`array-number`, `begin-string`, `end-string`**[**, `from-line`**][**, `NO-DELIMITER/DELIMITER`**]**)
SCUT extracts lines of a SARRAY. If NO-DELIMITER is specified, the extraction starts with the lines after the begin-string and ends with the line before the end string is found. If DELIMITER is specified, the delimiter lines are included. The default is NO-DELIMITER.
For example, we have the following SARRAY (s1):

```
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   Record 1
00002   Record 2
00003   Record 3
00004   Record 4
00005   Record 5
00006   From Here
00007   Data 1
00008   Data 2
00009   Data 3
00010   End
00011   Record 6
00012   Record 7
00013   Record 8
00014   Record 9
00015   Record 10
15 Entries
```

And the following REXX:

```
1 s2=sextract(s1,"From Here","End")
2 call slist s2
```

Result:

```
    Entries of Source Array: 1
Entry   Data
-----------------------------------------------------
00001   Data 1
00002   Data 2
00003   Data 3
3 Entries
```

SARRAY (`array-number`)
Returns information about the Source Array. The following BREXX variables are set:

- *sarrayhi* highest element number set in the array
- *sarraymax* maximum entries available
- *sarrayADDR* address of the Source Array

Returns: highest array entry.

Set Theory and Arrays

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

To utilize arrays with Set Theory operations, they must be unique and sorted. If an array does not follow these principles, its results may be unpredictable.

SUNIFY (array-number)
Creates and sorts an array and just keeps unique elements.

Returns: the number of removed elements.

```
1 s1=screate(10)
2 do i=1 to 10
3    call sset(s1,,right(random(1,5),4,'0'))
4 end
5 call slist s1
6 call sunify(s1)
7 call slist s1
```

Result:

```
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   0005
00002   0005
00003   0003
00004   0005
00005   0003
00006   0004
00007   0003
00008   0001
00009   0001
00010   0003
10 Entries
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00001   0001
00002   0003
00003   0004
00004   0005
4 Entries
```

SUNION (array-1, array-2)
Builds a new array consisting of elements of both arrays. Any duplicates are removed in the new array. The new array is sorted. The set operation is: array1 ∪ array2

Returns: the created array number.

```
 1 s1=screate(10)
 2 s2=screate(10)
 3 do i=1 to 10
 4    call sset(s1,,right(random(1,5),4,'0'))
 5    call sset(s2,,right(random(3,14),4,'0'))
 6 end
 7 call slist s1
 8 call slist s2
 9 call sunify(s1)
10 call sunify(s2)
11 call slist s1
12 call slist s2
13 s3=sunion(s1,s2)
14 call slist s3
```

Result:

```
    Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   0003
00002   0002
00003   0004
00004   0003
00005   0004
00006   0002
00007   0002
00008   0001
00009   0005
00010   0005
10 Entries
    Entries of Source Array: 1
Entry   Data
-------------------------------------------------------
00001   0008
00002   0012
00003   0009
00004   0006
00005   0013
00006   0009
00007   0007
00008   0003
00009   0005
00010   0006
10 Entries
    Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   0001
00002   0002
00003   0003
00004   0004
00005   0005
5 Entries
    Entries of Source Array: 1
Entry   Data
-------------------------------------------------------
00001   0003
00002   0005
00003   0006
00004   0007
00005   0008
00006   0009
00007   0012
00008   0013
8 Entries
    Entries of Source Array: 2
Entry   Data
00001   0001
00002   0002
00003   0003
00004   0004
00005   0005
00006   0006
00007   0007
00008   0008
00009   0009
```

```
00010   0012
00011   0013
11 Entries
```

SINTERSECT (array-1, array-2)

Creates a new array by intersecting two existing arrays. It contains which are in both arrays. The set operation is: array1 ∩ array2.

Returns: the created array number.

```
 1 s1=screate(8)
 2 s2=screate(8)
 3 do i=1 to 8
 4    call sset(s1,,right(random(1,5),4,'0'))
 5    call sset(s2,,right(random(3,14),4,'0'))
 6 end
 7 call slist s1
 8 call slist s2
 9 call sunify(s1)
10 call sunify(s2)
11 call slist s1
12 call slist s2
13 s3=sintersect(s1,s2)
14 call slist s3
```

Result:

```
    Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   0002
00002   0001
00003   0005
00004   0005
00005   0005
00006   0005
00007   0007
00008   0001
8 Entries
    Entries of Source Array: 1
Entry   Data
-------------------------------------------------------
00001   0007
00002   0011
00003   0006
00004   0013
00005   0008
00006   0003
00007   0006
00008   0014
8 Entries
    Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   0001
00002   0002
00003   0005
00004   0007

3 Entries
    Entries of Source Array: 1
```

```
Entry   Data
-------------------------------------------------------
00001   0003
00002   0006
00003   0007
00004   0008
00005   0011
00006   0013
00007   0014
7 Entries
    Entries of Source Array: 2
Entry   Data
-------------------------------------------------------
00001   0007
1 Entries
```

STDROP (array-1, array-2)

SDIFFERENCE (array-1, array-2)

Creates a new array which is the difference between array-1 and array-2. All elements contained in array-2 are dropped from array-1 (if contained). The set operation is: array1 - array2.

Returns: the created array number.

```
 1 s1=screate(8)
 2 s2=screate(8)
 3 do i=1 to 8
 4    call sset(s1,,right(random(1,5),4,'0'))
 5    call sset(s2,,right(random(3,14),4,'0'))
 6 end
 7 call slist s1
 8 call slist s2
 9 call sunify(s1)
10 call sunify(s2)
11 call slist s1
12 call slist s2
13 s3=sdifference(s1,s2)
14 call slist s3
15
16
17      Entries of Source Array: 0
18  Entry   Data
19  -------------------------------------------------------
20  00001   0001
21  00002   0002
22  00003   0004
23  00004   0004
24  00005   0001
25  00006   0004
26  00007   0001
27  00008   0004
28  8 Entries
29      Entries of Source Array: 1
30  Entry   Data
31  -------------------------------------------------------
32  00001   0009
33  00002   0003
34  00003   0008
35  00004   0004
36  00005   0014
37  00006   0011
```

```
00007   0005
00008   0006
8 Entries
    Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   0001
00002   0002
00003   0004
3 Entries
    Entries of Source Array: 1
Entry   Data
-------------------------------------------------------
00001   0003
00002   0004
00003   0005
00004   0006
00005   0008
00006   0009
00007   0011
00008   0014
8 Entries
    Entries of Source Array: 2
Entry   Data
-------------------------------------------------------
00001   0001
00002   0002
2 Entries
```

SDIFFSYM (array-1, array-2)

Creates a new array building the symmetrical difference of set1 and set2, operation: set1 Δ set2

```
smax=25
s1=screate(smax)     -- Create first Array
s2=screate(smax)     -- Create second Array
do i=1 to smax       -- Preset Arrays with randomised numbers
   call sset(s1,,right(random(1,smax%2),4,'0'))
   call sset(s2,,right(random(3,smax%3),4,'0'))
end
call sunify(s1)       -- Unify first array
call sunify(s2)       -- Unify second array
s3=sdiffsym(s1,s2)    -- build symmetrical difference
call slist s1,,,'First Set'
call slist s2,,,'Second Set'
call slist s3,,,'Symmetrical Difference'
```

Result:

```
    Entries of Source Array: 0
Entry   First Set
-------------------------------------------------------
00001   0002
00002   0003
00003   0004
00004   0005
00005   0006
00006   0007
00007   0008
00008   0009
00009   0010
00010   0012
```

```
10 Entries
    Entries of Source Array: 1
Entry   Second Set
-----------------------------------------------------
00001   0003
00002   0004
00003   0005
00004   0006
00005   0007
00006   0008
6 Entries
    Entries of Source Array: 4
Entry   Symmetrical Difference
-----------------------------------------------------
00001   0002
00002   0009
00003   0010
00004   0012
4 Entries
```

Integer Array Functions

ICREATE (`elements`, `mode`)
Creates an integer array with the size elements. Returned is the array number to be used to address the array with ISET and IGET. You can have up to 64 integer arrays. Depending on the virtual storage they may contain 1 million elements and more. Accessing integer arrays is very fast as there is no overhead compared to STEM variables.

Parameters:

- **elements** – Number entries available
- **mode** – The initialization type. If mode is not set the array remains uninitialized.

Mode	Description
Element-Number	index of element
NULL	elements are set to 0
DESCENT	index of the element in reverse order
SUNDARAM	prime numbers (Sundaram algorithm)
PRIME	prime numbers (sieve of Erasthones)

ISET (`array-number`, `element-number`, `integer-value`)
Sets a certain element of an array with an integer value.

IGET (`array-number`, `element-number`)
Gets (returns) a certain element of an array with an integer value.

Integer Matrix

The integer matrix is based on an integer array, the rows and columns are internally translated into the position in the array.

IMCREATE (`rows`, `columns`)
Creates an integer matrix containing the specified number of rows and columns. The matrix is initialized with zeros.

Returns: allocated array-number which can be used in subsequent array functions.

IMSET (`array-number`, `row`, `column`, `integer-value`)
Sets a certain element of the matrix to an integer value.

IMGET (`array-number`, `row`, `column`)
Gets (returns) a certain element of the matrix.

IMADD (`array-number`, `row`, `column`, `integer-value`)
Adds an integer value to a certain element of the matrix.

IMSUB (`array-number`, `row`, `column`, `integer-value`)
Subtracts an integer value from a certain element of the matrix.

IARRAY(array-number,'ROW'/'COLUMN)`

Returns: the number of rows or columns of the matrix.

IFREE (`array-number`)
Frees a defined integer array or matrix.

Float Array

FCREATE (`elements`, `mode`)
Creates an float array with the size elements. Returned is the array number to be used to address the array with FSET and FGET. You can have up to 64 integer arrays.

Returns: the allocated array-number which can be used in subsequent array functions.

FSET (`array-number`, `element-number`, `float-value`)
Sets a certain element of an array with a float value.

FGET (`array-number`, `element-number`)

Returns: (gets) a certain element of the float array.

FARRAY (`array-number`)

Returns: the highest array index set in the float array

FLIST (`array-number`**[**, `from`**][**, `to`**][**, `heading`**]**)
Prints the array content. With the optional from and to parameters, you can limit the range of entries to be printed. The optional heading parameter is printed in the heading line.

FFREE (`array-number`)
Frees a defined float array.

Linked List functions

LLCREATE ()
Creates a Linked List, returned is the Linked List Number(llist-number) which must be used in various Linked List operations.
The Linked List is bidirectional. You can have up to 32 different Linked Lists, depending on the virtual storage availability.

Returns: the allocated linked-list-number which can be used in subsequent linked list functions.

LLFREE (`llist-number`)
Removes the Linked List and all its entries. All storage allocations are freed.

LLCLEAR (`llist-number`)
Clears (removes) the Linked List entries, but the list header remains intact. From there you can add new entries to it.

LLADD (`llist-number`, `"entry-text"`)
Adds a new entry (llentry) at the end of the Linked List and links up the previous entry with a forward and the new entry backward reference. If the operation is successful a pointer (llpointer) to the new entry is returned. If the operation fails a return code < 0 is returned.
The internal current pointer (llcurrent) is set to the new entry and can be used in subsequent Linked List operations.
Example see *LLINSERT*

LLDEL (`llist-number`**[**, `llist-pointer`**]**)

Removes an entry, defined by the current entry or the specified llist-pointer (llpointer. If the operation was successful the internal current pointer (llcurrent) is set to the next entry, if there is no one, to the last element. Returned will be the internal current pointer (llcurrent). If the operation fails a return code < 0 is returned.
Example:

```
1 ll1=llread("'pej.songs2'")          /* Create Linked List */
2 call lllist ll1
3 call llset(ll1,"POSITION",3)        /* set to 3. Entry    */
4 call lldel(ll1)                     /* remove AC/DC       */
5 call lllist ll1
6 call llfree ll1
```

Result:

```
     Entries of Linked List: 0 (0)
Entry Entry Address     Next    Previous      Data
-------------------------------------------------------
    1     3061c8      306258           0   LED ZEPPELIN                STAIRWAY TO HEAVE
    2     306258      3062e8      3061c8   EAGLES                      HOTEL CALIFORNIA
    3     3062e8      306378      306258   AC/DC                       BACK IN BLACK
    4     306378      306408      3062e8   JOURNEY                     DON'T STOP BELIEV
    5     306408      306498      306378   PINK FLOYD                  ANOTHER BRICK IN
    6     306498      306528      306408   QUEEN                       BOHEMIAN RHAPSODY
    7     306528      3065b8      306498   TOTO                        HOLD THE LINE
    8     3065b8      306648      306528   KISS                        I WAS MADE FOR LO
    9     306648      3066d8      3065b8   BON JOVI                    LIVIN' ON A PRAYE
   10     3066d8      306768      306648   NIRVANA                     SMELLS LIKE TEEN
   11     306768      3067f8      3066d8   DEEP PURPLE                 SMOKE ON THE WATE
   12     3067f8      306888      306768   METALLICA                   NOTHING ELSE MATT
   13     306888      306918      3067f8   THE ROLLING STONES          (I CAN'T GET NO)
   14     306918      3069a8      306888   BRUCE SPRINGSTEEN           BORN IN THE U.S.A
   15     3069a8      305498      306918   QUEEN                       WE WILL ROCK YOU
   16     305498      306a38      3069a8   LYNYRD SKYNYRD              FREE BIRD
   17     306a38      306ac8      305498   SURVIVOR                    EYE OF THE TIGER
   18     306ac8      305458      306a38   THE CLASH                   SHOULD I STAY OR
   19     305458      305658      306ac8   JIMI HENDRIX                HEY JOE
   20     305658      306b58      305458   FLEETWOOD MAC               LITTLE LIES
   21     306b58      305618      305658   AC/DC                       HIGHWAY TO HELL
   22     305618           0      306b58   THE POLICE                  ROXANNE
Linked List contains 22 Entries
       List counter  22 Entries
Current active Entry 305618
     Entries of Linked List: 0 (0)
Entry Entry Address     Next    Previous      Data
-------------------------------------------------------
    1     3061c8      306258           0   LED ZEPPELIN                STAIRWAY TO HEAVE
    2     306258      306378      3061c8   EAGLES                      HOTEL CALIFORNIA
    3     306378      306408      306258   JOURNEY                     DON'T STOP BELIEV
    4     306408      306498      306378   PINK FLOYD                  ANOTHER BRICK IN
    5     306498      306528      306408   QUEEN                       BOHEMIAN RHAPSODY
    6     306528      3065b8      306498   TOTO                        HOLD THE LINE
    7     3065b8      306648      306528   KISS                        I WAS MADE FOR LO
    8     306648      3066d8      3065b8   BON JOVI                    LIVIN' ON A PRAYE
    9     3066d8      306768      306648   NIRVANA                     SMELLS LIKE TEEN
   10     306768      3067f8      3066d8   DEEP PURPLE                 SMOKE ON THE WATE
   11     3067f8      306888      306768   METALLICA                   NOTHING ELSE MATT
   12     306888      306918      3067f8   THE ROLLING STONES          (I CAN'T GET NO)
   13     306918      3069a8      306888   BRUCE SPRINGSTEEN           BORN IN THE U.S.A
   14     3069a8      305498      306918   QUEEN                       WE WILL ROCK YOU
   15     305498      306a38      3069a8   LYNYRD SKYNYRD              FREE BIRD
   16     306a38      306ac8      305498   SURVIVOR                    EYE OF THE TIGER
   17     306ac8      305458      306a38   THE CLASH                   SHOULD I STAY OR
```

```
    18     305458      305658      306ac8   JIMI HENDRIX                  HEY JOE
    19     305658      306b58      305458   FLEETWOOD MAC                 LITTLE LIES
    20     306b58      305618      305658   AC/DC                         HIGHWAY TO HELL
    21     305618           0      306b58   THE POLICE                    ROXANNE
 Linked List contains 21 Entries
        List counter  21 Entries
 Current active Entry 306378
```

LLINSERT (llist-number, "entry-text"**[**, llist-pointer**]**)

Inserts a new entry (llentry) before the current entry or the specified llist-pointer. All link information from the predecessor and successor entries is updated.

If the operation is successful a pointer (llpointer) to the inserted entry is returned. If the operation fails a return code < 0 is returned.

The internal current pointer (llcurrent) is set to the new entry and can be used in subsequent Linked List operations.

Example:

```
 1 ll1=llread("'pej.songs2'")           /* Create Linked List */
 2 say copies('-',32)
 3 say "Run Through Linked List"
 4 say copies('-',32)
 5 say llget(ll1,"FIRST")
 6 do while llset(ll1,"NEXT")>0
 7    say llget(ll1)
 8 end
 9 call llset(ll1,"POSITION",2)       /* set to 1. Entry    */
10 call llinsert(ll1,"CREAM                          I AM SO GLAD")
11 call lllist ll1
12 call llfree ll1
```

Result:

```
--------------------------------
Run Through Linked List
--------------------------------
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
KISS                          I WAS MADE FOR LOVIN' YOU
BON JOVI                      LIVIN' ON A PRAYER
NIRVANA                       SMELLS LIKE TEEN SPIRIT
DEEP PURPLE                   SMOKE ON THE WATER
METALLICA                     NOTHING ELSE MATTERS
THE ROLLING STONES            (I CAN'T GET NO) SATISFACTION
BRUCE SPRINGSTEEN             BORN IN THE U.S.A.
QUEEN                         WE WILL ROCK YOU
LYNYRD SKYNYRD                FREE BIRD
SURVIVOR                      EYE OF THE TIGER
THE CLASH                     SHOULD I STAY OR SHOULD I GO
JIMI HENDRIX                  HEY JOE
FLEETWOOD MAC                 LITTLE LIES
AC/DC                         HIGHWAY TO HELL
THE POLICE                    ROXANNE
     Entries of Linked List: 0 (0)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     305258      305138           0   LED ZEPPELIN                  STAIRWAY TO HEAVE
```

```
      2     305138      3052e8      305258   CREAM                          I AM SO GLAD
      3     3052e8      305378      305258   EAGLES                         HOTEL CALIFORNIA
      4     305378      305408      3052e8   AC/DC                          BACK IN BLACK
      5     305408      305498      305378   JOURNEY                        DON'T STOP BELIEV
      6     305498      305528      305408   PINK FLOYD                     ANOTHER BRICK IN
      7     305528      3055b8      305498   QUEEN                          BOHEMIAN RHAPSODY
      8     3055b8      305648      305528   TOTO                           HOLD THE LINE
      9     305648      3056d8      3055b8   KISS                           I WAS MADE FOR LO
     10     3056d8      305768      305648   BON JOVI                       LIVIN' ON A PRAYE
     11     305768      3057f8      3056d8   NIRVANA                        SMELLS LIKE TEEN
     12     3057f8      305888      305768   DEEP PURPLE                    SMOKE ON THE WATE
     13     305888      305918      3057f8   METALLICA                      NOTHING ELSE MATT
     14     305918      3059a8      305888   THE ROLLING STONES             (I CAN'T GET NO)
     15     3059a8      305a38      305918   BRUCE SPRINGSTEEN              BORN IN THE U.S.A
     16     305a38      304818      3059a8   QUEEN                          WE WILL ROCK YOU
     17     304818      305ac8      305a38   LYNYRD SKYNYRD                 FREE BIRD
     18     305ac8      305b58      304818   SURVIVOR                       EYE OF THE TIGER
     19     305b58      3047d8      305ac8   THE CLASH                      SHOULD I STAY OR
     20     3047d8      3049d8      305b58   JIMI HENDRIX                   HEY JOE
     21     3049d8      305be8      3047d8   FLEETWOOD MAC                  LITTLE LIES
     22     305be8      304998      3049d8   AC/DC                          HIGHWAY TO HELL
     23     304998           0      305be8   THE POLICE                     ROXANNE
Linked List contains 23 Entries
       List counter  23 Entries
Current active Entry 305138
```

LLGET (llist-number[option/llist-pointer])

Returns: the entry referred by the option or internal current pointer, or the specified llist-pointer. The internal current pointer (llcurrent) is not changed.

Parameters: **option** – NEXT sets it to the next element after llcurrent in the Linked List chain. If llcurrent was the last element 0 is returned. PREVIOUS sets it to the previous element of llcurrent in the Linked List chain. If llcurrent was the first element 0 is returned. FIRST sets it to the first element in the Linked List. LAST sets it to the last element in the Linked List.

Example:

```
ll1=llread("'pej.songs2'")          /* Create Linked List */
say copies('-',32)
say "Run Through Linked List"
say copies('-',32)
say llget(ll1,"FIRST")
do while llset(ll1,"NEXT")>0
   say llget(ll1)
end
call llfree ll1
```

Result:

```
--------------------------------
Run Through Linked List
--------------------------------
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
KISS                           I WAS MADE FOR LOVIN' YOU
BON JOVI                       LIVIN' ON A PRAYER
NIRVANA                        SMELLS LIKE TEEN SPIRIT
```

```
DEEP PURPLE                  SMOKE ON THE WATER
METALLICA                    NOTHING ELSE MATTERS
THE ROLLING STONES           (I CAN'T GET NO) SATISFACTION
BRUCE SPRINGSTEEN            BORN IN THE U.S.A.
QUEEN                        WE WILL ROCK YOU
LYNYRD SKYNYRD               FREE BIRD
SURVIVOR                     EYE OF THE TIGER
THE CLASH                    SHOULD I STAY OR SHOULD I GO
JIMI HENDRIX                 HEY JOE
FLEETWOOD MAC                LITTLE LIES
AC/DC                        HIGHWAY TO HELL
THE POLICE                   ROXANNE
```

LLSET (`llist-number`, `option`**[**, `sub-option`**]**)
Changes the internal current pointer according to the specified option and returns it as a pointer.

Parameters: **option** – NEXT sets it to the next element after llcurrent in the Linked List chain. If llcurrent was the last element 0 is returned. PREVIOUS sets it to the previous element of llcurrent in the Linked List chain. If llcurrent was the first element 0 is returned. FIRST sets it to the first element in the Linked List. LAST sets it to the last element in the Linked List. POSITION sets it to n.th entry, as defined in sub-option. If the specified number is not available it is set to the last entry. CURRENT returns the current internal current pointer. ADDRESS sets it according to the address defined in the sub-option.

LLCOPY (`llist-number`**[**, `from`**][**, `to`**][**, `existing-list`**][**, ”`list-name`”**]**)
Creates a copy of the Linked List. If an existing linked-list is specified, the entries are added after its existing entries.

Parameters:

- **from** – (optional) starts the copying process at from.th entry.
- **to** – (optional) ends the copying process with to.th entry.
- **existing-list** – (optional) appending an existing Source Array, else a new one will be created
- **list-name** – (optional) names the new/appended Link List

Returns: the newly created or appended Linked List Number(llist-number)

Example:

```
 1 max=10
 2 ll1=llcreate()                        /* Create Linked List */
 3 ll2=llcreate()                        /* Create Linked List */
 4 call time('r')
 5 do i=1 to max
 6    adr=lladd(ll1,i". Record")
 7 end
 8 call llList ll1
 9 do i=1 to 5
10    adr=lladd(ll2,i". Entry")
11 end
12 call llList ll2
13 ll3=llcopy(ll1,,,ll2,"Copied")
14 call llList ll3
```

Result:

```
      Entries of Linked List: 0 (UNNAMED)
 Entry Entry Address     Next    Previous      Data
 ------------------------------------------------------
     1     2e3258      2e3278           0   1. Record
     2     2e3278      2e3298      2e3258   2. Record
     3     2e3298      2e32b8      2e3278   3. Record
     4     2e32b8      2e32d8      2e3298   4. Record
```

```
     5     2e32d8      2e32f8      2e32b8   5. Record
     6     2e32f8      2e3318      2e32d8   6. Record
     7     2e3318      2e3338      2e32f8   7. Record
     8     2e3338      2e3358      2e3318   8. Record
     9     2e3358      2e3378      2e3338   9. Record
    10     2e3378           0      2e3358   10. Record
 Linked List contains 10 Entries
        List counter  10 Entries
 Current active Entry 2e3378

      Entries of Linked List: 1 (UNNAMED)
 Entry Entry Address     Next    Previous      Data
 ------------------------------------------------------
     1     2e3398      2e33b8           0   1. Entry
     2     2e33b8      2e33d8      2e3398   2. Entry
     3     2e33d8      2e33f8      2e33b8   3. Entry
     4     2e33f8      2e3418      2e33d8   4. Entry
     5     2e3418           0      2e33f8   5. Entry
Current active Entry 2e3418
Linked List contains 5 Entries
       List counter  5 Entries

     Entries of Linked List: 1 (Copied)
Entry Entry Address     Next    Previous      Data
------------------------------------------------------
    1     2e3398      2e33b8           0   1. Entry
    2     2e33b8      2e33d8      2e3398   2. Entry
    3     2e33d8      2e33f8      2e33b8   3. Entry
    4     2e33f8      2e3418      2e33d8   4. Entry
    5     2e3418      2e3458      2e33f8   5. Entry
    6     2e3458      2e3478      2e3418   1. Record
    7     2e3478      2e3498      2e3458   2. Record
    8     2e3498      2e34b8      2e3478   3. Record
    9     2e34b8      2e34d8      2e3498   4. Record
   10     2e34d8      2e34f8      2e34b8   5. Record
   11     2e34f8      2e3518      2e34d8   6. Record
   12     2e3518      2e3538      2e34f8   7. Record
   13     2e3538      2e3558      2e3518   8. Record
   14     2e3558      2e3578      2e3538   9. Record
   15     2e3578           0      2e3558   10. Record
Linked List contains 15 Entries
       List counter  15 Entries
Current active Entry 2e3578
```

LLENTRY (`llist-number [,llist-pointer]`))
Dumps the details of an entry either defined by the internal current pointer (llcurrent) or the llist-pointer.
Example:

```
-------------------------------------------
Linked List Entry
-------------------------------------------
Address  326f18
Data     42. Record
Next     326f58
Previous 326ed8
```

LLLIST (`llist-number`**[**, `from`**][**, `to`**]**)
Outputs a detailed list of all entries on a Linked list:

```
     Entries of Linked List: 0
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     326458      326498           0   1. Record
    2     326498      3264d8      326458   2. Record
    3     3264d8      326518      326498   3. Record
    4     326518      326558      3264d8   4. Record
    5     326558      326598      326518   5. Record
    6     326598      3265d8      326558   6. Record
    7     3265d8      326618      326598   7. Record
    8     326618      326658      3265d8   8. Record
    9     326658      326698      326618   9. Record
   10     326698      3266d8      326658   10. Record
   11     3266d8      326718      326698   11. Record
   12     326718      326758      3266d8   12. Record
...
```

LLDETAILS (`llist-number`, `option`)
Output statistics on the Linked List.

Parameters: **option** – COUNT returns the number of current entries in the Linked List. ADDED returns the number of added/inserted entries in the Linked List. DELETED returns the number of deleted entries in the Linked List. LIST returns the listed number of current entries in the Linked List. For this reason, it runs through the entire Linked List and counts the entries. LIST and COUNT should be equal, else there are inconsistencies in the Linked List. FULL print all available information

Example:

```
CALL LLDETAILS(0,'FULL')


Attributes of Linked List 0
-----------------------------------------------------
Entry Count     9999
    Listed     9999
    Added     10000
    Deleted     1
Current Pointer 326ed8
```

LLDELINK (`llist-number`**[**, `llist-pointer`**]**)
Similar to LLDEL an entry defined by the current entry or the specified llist-pointer is removed from the Link List but is kept in storage as an orphan, which might be later inserted in a different position in the same or a different Linked List. This is a fast way of moving elements.

Returns: the address of the orphaned entry.

If the operation was successful the internal current pointer (llcurrent) is set to the next entry, and if there is no one, to the last element.
The example is contained in the *LLLINK* sample.

LLLINK (`llist-number`, `llist-pointer`)
Links an orphaned entry to the Linked List prior to the current entry and sets the pointers accordingly.
If the operation was successful, the internal current pointer (llcurrent) is set to the newly inserted entry.
Example:

```
1 max=10
2 ll1=llcreate()                      /* Create Linked List */
3 ll2=llcreate()                      /* Create Linked List */
4 do i=1 to max
5    adr=lladd(ll1,i". Record")       /* add new entry      */
6 end
7 call llList ll1
```

```
 8 posadr=llset(ll1,"POSITION",7)       /* set to 7. Entry    */
 9 deladr=lldelink(ll1,posadr)          /* DELINK it          */
10 say "is now de-linked,ADDR "d2x(deladr)
11 call llList ll1
12 say "Insert one entry to LL2 "d2x(llinsert(ll2,"1. Entry"))
13 call llList ll2
14 say "LINK into new LList "d2x(llLink(ll2,deladr))
15 call llList ll2
```

Result:

```
     Entries of Linked List: 0 (UNNAMED)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     2db238      2db258           0   1. Record
    2     2db258      2db278      2db238   2. Record
    3     2db278      2db298      2db258   3. Record
    4     2db298      2db2b8      2db278   4. Record
    5     2db2b8      2db2d8      2db298   5. Record
    6     2db2d8      2db2f8      2db2b8   6. Record
    7     2db2f8      2db318      2db2d8   7. Record
    8     2db318      2db338      2db2f8   8. Record
    9     2db338      2db358      2db318   9. Record
   10     2db358           0      2db338   10. Record
Linked List contains 10 Entries
       List counter  10 Entries
is now de-linked,ADDR 2DB2F8
     Entries of Linked List: 0 (UNNAMED)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     2db238      2db258           0   1. Record
    2     2db258      2db278      2db238   2. Record
    3     2db278      2db298      2db258   3. Record
    4     2db298      2db2b8      2db278   4. Record
    5     2db2b8      2db2d8      2db298   5. Record
    6     2db2d8      2db318      2db2b8   6. Record
    7     2db318      2db338      2db2d8   8. Record
    8     2db338      2db358      2db318   9. Record
    9     2db358           0      2db338   10. Record
Linked List contains 9 Entries
       List counter  9 Entries
Insert one entry to LL2 2DB3D8
     Entries of Linked List: 1 (UNNAMED)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     2db3d8           0           0   1. Entry
Linked List contains 1 Entries
       List counter  1 Entries
LINK into new LList 2DB2F8
     Entries of Linked List: 1 (UNNAMED)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     2db2f8      2db3d8           0   7. Record
    2     2db3d8           0           0   1. Entry
Linked List contains 2 Entries
       List counter  2 Entries
```

LLSORT (`llist--number`**[**, `ASCENDING/DESCENDING`**][**, `sort-offset`**]**)
Sorts the Linked List using the quick sort algorithm in ascending or descending order, default is ascending.
The sort offset defines the sorting scope up to the end of the item, any substrings before it are not treated. If you define for example 5, the array is sorted at offset 5 (up to the rest of the item). The sort-offset defaults to 1.

Returns: the Linked List Number(llist-number), it is the same as the entry list.

Example:

```
1 ll1=llread("'pej.songs2'")
2 call llList ll1
3 call llsort ll1
4 call llList ll1    /* sort from column 1, band name */
5 call llfree ll1
```

Result:

```
     Entries of Linked List: 0 (0)
Entry Entry Address     Next    Previous      Data
-------------------------------------------------------
    1     3371c8      337258           0   LED ZEPPELIN                 STAIRWAY TO HEAVE
    2     337258      3372e8      3371c8   EAGLES                       HOTEL CALIFORNIA
    3     3372e8      337378      337258   AC/DC                        BACK IN BLACK
    4     337378      337408      3372e8   JOURNEY                      DON'T STOP BELIEV
    5     337408      337498      337378   PINK FLOYD                   ANOTHER BRICK IN
    6     337498      337528      337408   QUEEN                        BOHEMIAN RHAPSODY
    7     337528      3375b8      337498   TOTO                         HOLD THE LINE
    8     3375b8      337648      337528   KISS                         I WAS MADE FOR LO
    9     337648      3376d8      3375b8   BON JOVI                     LIVIN' ON A PRAYE
   10     3376d8      337768      337648   NIRVANA                      SMELLS LIKE TEEN
   11     337768      3377f8      3376d8   DEEP PURPLE                  SMOKE ON THE WATE
   12     3377f8      337888      337768   METALLICA                    NOTHING ELSE MATT
   13     337888      337918      3377f8   THE ROLLING STONES           (I CAN'T GET NO)
   14     337918      3379a8      337888   BRUCE SPRINGSTEEN            BORN IN THE U.S.A
   15     3379a8      336258      337918   QUEEN                        WE WILL ROCK YOU
   16     336258      337a38      3379a8   LYNYRD SKYNYRD               FREE BIRD
   17     337a38      337ac8      336258   SURVIVOR                     EYE OF THE TIGER
   18     337ac8      336218      337a38   THE CLASH                    SHOULD I STAY OR
   19     336218      336418      337ac8   JIMI HENDRIX                 HEY JOE
   20     336418      337b58      336218   FLEETWOOD MAC                LITTLE LIES
   21     337b58      3363d8      336418   AC/DC                        HIGHWAY TO HELL
   22     3363d8           0      337b58   THE POLICE                   ROXANNE
Linked List contains 22 Entries
       List counter  22 Entries
Current active Entry 3363d8
     Entries of Linked List: 0 (0)
Entry Entry Address     Next    Previous      Data
-------------------------------------------------------
    1     300a38      3009a8           0   AC/DC                        BACK IN BLACK
    2     3009a8      300918      300a38   AC/DC                        HIGHWAY TO HELL
    3     300918      300888      3009a8   BON JOVI                     LIVIN' ON A PRAYE
    4     300888      3007f8      300918   BRUCE SPRINGSTEEN            BORN IN THE U.S.A
    5     3007f8      300768      300888   DEEP PURPLE                  SMOKE ON THE WATE
    6     300768      337b58      3007f8   JOURNEY                      DON'T STOP BELIEV
    7     337b58      337ac8      300768   KISS                         I WAS MADE FOR LO
    8     337ac8      337a38      337b58   NIRVANA                      SMELLS LIKE TEEN
    9     337a38      3379a8      337ac8   PINK FLOYD                   ANOTHER BRICK IN
   10     3379a8      337918      337a38   QUEEN                        BOHEMIAN RHAPSODY
   11     337918      337888      3379a8   TOTO                         HOLD THE LINE
   12     337888      3363d8      337918   EAGLES                       HOTEL CALIFORNIA
   13     3363d8      336418      337888   FLEETWOOD MAC                LITTLE LIES
   14     336418      3377f8      3363d8   JIMI HENDRIX                 HEY JOE
   15     3377f8      336218      336418   LED ZEPPELIN                 STAIRWAY TO HEAVE
   16     336218      337768      3377f8   LYNYRD SKYNYRD               FREE BIRD
   17     337768      3376d8      336218   METALLICA                    NOTHING ELSE MATT
   18     3376d8      337648      337768   QUEEN                        WE WILL ROCK YOU
   19     337648      3375b8      3376d8   SURVIVOR                     EYE OF THE TIGER
```

```
    20     3375b8      336258      337648   THE CLASH                      SHOULD I STAY OR
    21     336258      337528      3375b8   THE POLICE                     ROXANNE
    22     337528           0      336258   THE ROLLING STONES             (I CAN'T GET NO)
 Linked List contains 22 Entries
        List counter  22 Entries
 Current active Entry 337528
```

LLWRITE (`llist-number`, `dsn/ddname`)
Writes all entries of a Linked List into an external dataset.
The dataset can be either a fully qualified Dataset Name or a pre-allocated DD Name.

Returns: the number of written entries.

An example is contained in *LLREAD*

LLREAD (`dsn/ddname`)
Reads all entries of an external dataset. The dataset can be either a fully qualified Dataset Name or a pre-allocated DD Name.

Returns: the newly created Linked List Number(llist-number).

Example:

```
1 ll1=llread("'pej.songs2'")
2 call lllist ll1
3 say "Records written: "llwrite(ll1,"'pej.temp'")
```

Result:

```
      Entries of Linked List: 0 (0)
 Entry Entry Address     Next    Previous      Data
 ------------------------------------------------------
     1     3371c8      337258           0   LED ZEPPELIN                   STAIRWAY TO HEAVE
     2     337258      3372e8      3371c8   EAGLES                         HOTEL CALIFORNIA
     3     3372e8      337378      337258   AC/DC                          BACK IN BLACK
     4     337378      337408      3372e8   JOURNEY                        DON'T STOP BELIEV
     5     337408      337498      337378   PINK FLOYD                     ANOTHER BRICK IN
     6     337498      337528      337408   QUEEN                          BOHEMIAN RHAPSODY
     7     337528      3375b8      337498   TOTO                           HOLD THE LINE
     8     3375b8      337648      337528   KISS                           I WAS MADE FOR LO
     9     337648      3376d8      3375b8   BON JOVI                       LIVIN' ON A PRAYE
    10     3376d8      337768      337648   NIRVANA                        SMELLS LIKE TEEN
    11     337768      3377f8      3376d8   DEEP PURPLE                    SMOKE ON THE WATE
    12     3377f8      337888      337768   METALLICA                      NOTHING ELSE MATT
    13     337888      337918      3377f8   THE ROLLING STONES             (I CAN'T GET NO)
    14     337918      3379a8      337888   BRUCE SPRINGSTEEN              BORN IN THE U.S.A
    15     3379a8      336358      337918   QUEEN                          WE WILL ROCK YOU
    16     336358      337a38      3379a8   LYNYRD SKYNYRD                 FREE BIRD
    17     337a38      337ac8      336358   SURVIVOR                       EYE OF THE TIGER
    18     337ac8      336318      337a38   THE CLASH                      SHOULD I STAY OR
    19     336318      336518      337ac8   JIMI HENDRIX                   HEY JOE
    20     336518      337b58      336318   FLEETWOOD MAC                  LITTLE LIES
    21     337b58      3364d8      336518   AC/DC                          HIGHWAY TO HELL
    22     3364d8           0      337b58   THE POLICE                     ROXANNE
 Records written: 22
```

MCREATE (`rows`, `columns`)
Creates a (Float) matrix with size *[rows x columns]*. Returned is the Matrix number to be used in various matrix operations. You can have up to 128 matrixes, depending on the virtual storage available. Accessing a matrix is very fast as there is no overhead compared to STEM variables.

MSET (`matrix-number`, `row`, `column`, `float-value`)
Sets a certain element of the matrix with a float value.

MGET (`matrix-number`, `row`, `column`)

Gets (returns) a certain element of the matrix.

MMULTIPLY (`matrix-number-1`, `matrix-number-2`)
Multiplies 2 matrices and creates a new matrix, which is returned. Input matrices remain untouched. The format of matrix-1 is *[rows x columns]*, therefore the format of matrix-2 must be *[columns x rows]*. The format of the result matrix is rows x rows.

MINVERT (`matrix-number`)
Inverts the given matrix and creates a new matrix, which is returned. The input matrix must be squared and remains untouched. The format of the result matrix remains the same as the input matrix.

MTRANSPOSE (`matrix-number`)
Transposes the given matrix and creates a new matrix, which is returned. The input matrix remains untouched. If the format of the input matrix is [rows x columns] then the result matrix is columns x rows.

MCOPY (`matrix-number`)
Copies the given matrix and creates a new matrix, which is returned. The input matrix remains untouched. Formats of both matrices are equal.

MNORMALISE (`matrix-number`, `mode`)
Normalises the given matrix and creates a new matrix, which is returned. The input matrix remains untouched. Formats of both matrices are equal.

Mode	Description
STANDARD	row is normalized to mean=0 variance=1
ROWS	row value is divided by number of rows
MEAN	row value is normalized to mean=0, variance remains unchanged

MDELROW (`matrix-number`, `row-number`**[**, `row-number`**[**, `row-number...`**]]**)
Copies the given matrix without the specified rows-to-delete as a new matrix, which is returned. The input matrix remains untouched.

MDELCOL (`matrix-number`, `col-number`**[**, `col-number`**[**, `col-number...`**]]**)
Copies the given matrix without the specified columns-to-delete as a new matrix, which is returned. The input matrix remains untouched.

MPROPERTY (`matrix-number`**[**, `"FULL"/"BASIC"`**]**)
Returns the properties of the given matrix in BREXX variables:

_rows	number of rows of matrix
_cols	number of columns of matrix

If **FULL** is specified additionally the the following stem variables are returned:

Stem	Description
_rowmean.column-i	mean of rows of column-i
_rowvariance.column-i	variance of rows of column-i
_rowlow.column-i	lowest row value of column-i
_rowhigh.column-i	highest row value of column-i
_rowsum.column-i	sum of row value of column-i
_rowsqr.column-i	sum of squared row value of column-i
_colsum.row-i	sum of column values of row-i

MSCALAR (`matrix-number`, `number`)
Multiplies each element of a matrix with a number (float). The result is stored in a new matrix, which is returned. The input matrix remains untouched.

MADD (`matrix-number-1`, `matrix-number-2`)
Adds each element of a matrix-1 with the same element of matrix-2. The result is stored in a new matrix, which is returned. The input matrix remains untouched. Matrix-1 and matrix-2 must have the same dimensions.

MSUBTRACT (`matrix-number-1`, `matrix-number-2`)

Subtracts each element of a matrix-2 from the same element of matrix-1. The result is stored in a new matrix, which is returned. The input matrix remains untouched. Matrix-1 and matrix-2 must have the same dimensions.

MPROD (`matrix-number-1`, `matrix-number-2`)
Multiplies each element of a matrix-1 with the same element of matrix-2. The result is stored in a new matrix, which is returned. The input matrix remains untouched. Matrix-1 and matrix-2 must have the same dimensions.

MSQR (`matrix-number`)
Squares each element of the matrix. The result is stored in a new matrix, which is returned. The input matrix remains untouched.

MINSCOL (`matrix-number`)
Inserts a new column as the first column. The initial first column becomes the second column, etc. The result is stored in a new matrix, which is returned. The input matrix remains untouched.

MFREE (**[**, `matrix-number/integer-array-number`, `"MATRIX"/"INTEGER-ARRAY"`**]**)
Frees the storage of allocated matrices and/or integer arrays. If no parameter is specified all allocations are freed. To release a specific matrix or integer-array the matrix-number or integer-array-number must be used as the first parameter, followed by the type to release.

Conversions between String Arrays, Linked Lists, and STEMS

STEM2S (`"stem-name."`)
Copies a stem variable into a Source Array, stem-name.0 must contain the number of items.
The copy process takes stem-name.1, stem-name.2, … up stem-name.n (where n is contained in stem-name.0) and copies it into a String Array.

Returns: the number of the String Array.

Example:

```
 1 xmax=1000
 2 do i=1 to xmax
 3    fred.i=i". record"
 4 end
 5 FRED.0=xmax
 6 say "Set Time  "time('e')
 7 call time('r')
 8 s1=stem2s("fred.")
 9 say "Copy Time "time('e')
10 call slist s1,xmax-10,xmax
```

Result:

```
Set Time  0.130996
Copy Time 0.066642
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00990   990. record
00991   991. record
00992   992. record
00993   993. record
00994   994. record
00995   995. record
00996   996. record
00997   997. record
00998   998. record
00999   999. record
01000   1000. record
```

S2STEM (`"array-number"`, `"stem-name."`)
Copies a SARRAY into a stem

Returns: the number of the items in the stem (String Array).

Example:

```
 1 smax=1000
 2 s1=screate(smax)
 3 do i=1 to smax
 4    call sset(s1,,"Record "i)
 5 end
 6 call slist s1,smax-10,smax
 7 call time('r')
 8 call s2stem(s1,"Fred.")
 9 say "S2STEM "time('e')
10 do i=smax-10 to smax
11    say i fred.i
12 end
```

Result:

```
    Entries of Source Array: 0
Entry   Data
-----------------------------------------------------
00990   Record 990
00991   Record 991
00992   Record 992
00993   Record 993
00994   Record 994
00995   Record 995
00996   Record 996
00997   Record 997
00998   Record 998
00999   Record 999
01000   Record 1000
S2STEM 0.253646
990 Record 990
991 Record 991
992 Record 992
993 Record 993
994 Record 994
995 Record 995
996 Record 996
997 Record 997
998 Record 998
999 Record 999
1000 Record 1000
```

S2IARRAY ()
Copies a SARRAY into an integer array.

Returns: the array number of the created array (Integer Array).

S2FARRAY ()
Copies a SARRAY into a float array.

Returns: the array number of the created array (Float Array).

S2LL (`array-number`**`[`**, `from`**`][`**, `to`**`][`**, `existing-linked-list`**`][`**, `"list-name"`**`]`**)
Copy a String Array into Linked List.

Parameters:

- **from** – (optional) starts the copying process at from.th entry.
- **to** – (optional) ends the copying process with to.th entry.
- **existing-list** – (optional) appending an existing Linked List, else a new one will be created
- **list-name** – (optional) name of the new/appended Linked List

Returns: the Linked List Number(llist-number)

Example:

```
1 s1=sread("'pej.songs2'")
2 call sList s1
3 ll2=s2ll(s1,,,,"LL Songs")
4 call llList(ll2)
5 call sfree(s1)
6 call llfree(ll2)
```

Result:

```
     Entries of Source Array: 0
Entry   Data
-------------------------------------------------------
00001   LED ZEPPELIN                  STAIRWAY TO HEAVEN
00002   EAGLES                        HOTEL CALIFORNIA
00003   AC/DC                         BACK IN BLACK
00004   JOURNEY                       DON'T STOP BELIEVIN'
00005   PINK FLOYD                    ANOTHER BRICK IN THE WALL
00006   QUEEN                         BOHEMIAN RHAPSODY
00007   TOTO                          HOLD THE LINE
00008   KISS                          I WAS MADE FOR LOVIN' YOU
00009   BON JOVI                      LIVIN' ON A PRAYER
00010   NIRVANA                       SMELLS LIKE TEEN SPIRIT
00011   DEEP PURPLE                   SMOKE ON THE WATER
00012   METALLICA                     NOTHING ELSE MATTERS
00013   THE ROLLING STONES            (I CAN'T GET NO) SATISFACTION
00014   BRUCE SPRINGSTEEN             BORN IN THE U.S.A.
00015   QUEEN                         WE WILL ROCK YOU
00016   LYNYRD SKYNYRD                FREE BIRD
00017   SURVIVOR                      EYE OF THE TIGER
00018   THE CLASH                     SHOULD I STAY OR SHOULD I GO
00019   JIMI HENDRIX                  HEY JOE
00020   FLEETWOOD MAC                 LITTLE LIES
00021   AC/DC                         HIGHWAY TO HELL
00022   THE POLICE                    ROXANNE
     Entries of Linked List: 0 (LL Songs)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
    1     337138      3371c8           0   LED ZEPPELIN                  STAIRWAY TO HEAVE
    2     3371c8      337258      337138   EAGLES                        HOTEL CALIFORNIA
    3     337258      3372e8      3371c8   AC/DC                         BACK IN BLACK
    4     3372e8      337378      337258   JOURNEY                       DON'T STOP BELIEV
    5     337378      337408      3372e8   PINK FLOYD                    ANOTHER BRICK IN
    6     337408      337498      337378   QUEEN                         BOHEMIAN RHAPSODY
    7     337498      337528      337408   TOTO                          HOLD THE LINE
    8     337528      3375b8      337498   KISS                          I WAS MADE FOR LO
    9     3375b8      337648      337528   BON JOVI                      LIVIN' ON A PRAYE
   10     337648      3376d8      3375b8   NIRVANA                       SMELLS LIKE TEEN
   11     3376d8      337768      337648   DEEP PURPLE                   SMOKE ON THE WATE
   12     337768      3377f8      3376d8   METALLICA                     NOTHING ELSE MATT
   13     3377f8      337888      337768   THE ROLLING STONES            (I CAN'T GET NO)
   14     337888      337918      3377f8   BRUCE SPRINGSTEEN             BORN IN THE U.S.A
```

```
    15     337918      3362d8      337888   QUEEN                          WE WILL ROCK YOU
    16     3362d8      3379a8      337918   LYNYRD SKYNYRD                 FREE BIRD
    17     3379a8      337a38      3362d8   SURVIVOR                       EYE OF THE TIGER
    18     337a38      336618      3379a8   THE CLASH                      SHOULD I STAY OR
    19     336618      3365d8      337a38   JIMI HENDRIX                   HEY JOE
    20     3365d8      337ac8      336618   FLEETWOOD MAC                  LITTLE LIES
    21     337ac8      336598      3365d8   AC/DC                          HIGHWAY TO HELL
    22     336598           0      337ac8   THE POLICE                     ROXANNE
 Linked List contains 22 Entries
        List counter  22 Entries
 Current active Entry 336598
```

LL2S (`llist-number`**[**, `from`**][**, `to`**][**, `existing-array`**]**)
Copy a Linked List into a Source Array.

Parameters:
- **from** – (optional) starts the copying process at from.th entry.
- **to** – (optional) ends the copying process with to.th entry.
- **existing-array** – (optional) appending an existing Source Array, else a new one will be created

Returns: the Linked List Number(llist-number)

Example:

```
 1 max=8
 2 ll1=llcreate()
 3 do i=1 to max
 4    adr=lladd(ll1,i". Record")
 5 end
 6 call llList ll1
 7 s1=ll2s(ll1)
 8 say "Linked List copied into Source Array "s1
 9 call slist(s1)
10 call llfree(ll1)
11 call sfree(s1)
```

Result:

```
      Entries of Linked List: 0 (UNNAMED)
 Entry Entry Address     Next    Previous      Data
 ------------------------------------------------------
     1     2e3218      2e3238           0   1. Record
     2     2e3238      2e3258      2e3218   2. Record
     3     2e3258      2e3278      2e3238   3. Record
     4     2e3278      2e3298      2e3258   4. Record
     5     2e3298      2e32b8      2e3278   5. Record
     6     2e32b8      2e32d8      2e3298   6. Record
     7     2e32d8      2e32f8      2e32b8   7. Record
     8     2e32f8           0      2e32d8   8. Record
 Linked List contains 8 Entries
        List counter  8 Entries
 Current active Entry 2e32f8
 Linked List copied into Source Array 0
      Entries of Source Array: 0
 Entry   Data
 ------------------------------------------------------
 00001   1. Record
 00002   2. Record
 00003   3. Record
 00004   4. Record
 00005   5. Record
```

```
00006   6. Record
00007   7. Record
00008   8. Record
```

LL2STEM (“llist-number”)
Copies a Linked List into a stem :returns: the number of the items in the stem (Linked List entries).
Example:

```
 1 max=1000
 2 /* ---------------------------------------
 3  * Copy LLIST into STEM
 4  * ---------------------------------------
 5  */
 6 LL1=LLCREATE("LLIST")
 7 do i=1 to max
 8    call LLADD(LL1,'FRED 'i)
 9 end
10 call time('r')
11 call ll2stem(LL1,'myStem.')
12 say "LL2STEM "time('e')
13 do i=mystem.0-10 to mystem.0
14    say i mystem.i
15 end
```

Result:

```
LL2STEM 0.195885
990 FRED 990
991 FRED 991
992 FRED 992
993 FRED 993
994 FRED 994
995 FRED 995
996 FRED 996
997 FRED 997
998 FRED 998
999 FRED 999
1000 FRED 1000
```

STEM2LL (“stem-name.”)
Copies stem into a Linked List, stem-name.0 must contain the number of items.
The copy process takes stem-name.1, stem-name.2, … up stem-name.n (where n is contained in stem-name.0) and copies it into a Linked List.

Returns: the created Linked List number.

Example:

```
 1 max=1000
 2 /* ---------------------------------------
 3  * Copy STEM into LLIST
 4  * ---------------------------------------
 5  */
 6 do i=1 to max
 7    myStem.i=i". Record"
 8 end
 9 mystem.0=max
10 call time('r')
11 ll1=stem2ll('myStem.')
12 say "STEM2LL "time('e')
13 call lllist ll1,max-10,max
```

Result:

```
STEM2LL 0.080318
     Entries of Linked List: 0 (UNNAMED)
Entry Entry Address     Next    Previous      Data
-----------------------------------------------------
  990     359498      3594d8      359458   990. Record
  991     3594d8      359518      359498   991. Record
  992     359518      359558      3594d8   992. Record
  993     359558      359598      359518   993. Record
  994     359598      3595d8      359558   994. Record
  995     3595d8      359618      359598   995. Record
  996     359618      359658      3595d8   996. Record
  997     359658      359698      359618   997. Record
  998     359698      3596d8      359658   998. Record
  999     3596d8      359718      359698   999. Record
 1000     359718           0      3596d8   1000. Record
Linked List address  34b218
Linked List contains 1000 Entries
       List counter  1000 Entries
Current active Entry 359718
```

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, BREXX delivers the following functions.

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.

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

Results:

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

```
1 spool=JESQUEUE()
2 call slist spool
3 exit
```

Results:

```
     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

Parameters:

- **msg-number** – message number to be displayed
- **msg-level** – One of: I, W, E, C

message level can be:

Message Level	Description
I	for an information message
W	for a warning message
E	for an error message
C	for a critical message

Example:

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

Results:

```
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**:

Return Code	Description

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

Message Level	Description
I	for an information message
W	for a warning message
E	for an error message
C	for a critical message

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 (`'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.
Initialize the function with *DCL('$DEFINE','structure-name')* where:

- *$DEFINE* initialises the structure definition
- *structure-name* all following field definitions are associated with the structure-name.

Parameters:

- **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 either **CHAR** no translation takes place, CHAR is default or **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:

```
 1  n=DCL('$DEFINE','student')
 2  n=DCL('Name',1,32,'CHAR')
 3  n=DCL('FirstName',1,16,'CHAR')
 4  n=DCL('LastName',,16,'CHAR')
 5  n=DCL('Address',,32,'CHAR')
 6  recin='Fred            Flintstone      Bedrock'
 7  /*    '12345678901234567890123456789012345678901234567890 */
 8  call splitRecord 'student',recin
 9  say Name
10  say FirstName
11  say LastName
12  say Address
13  firstName='Barney'
14  LastName='Rubble'
15  address='Bedrock'
16  say setRecord('student')
```

Results:

```
FRED            FLINTSTONE
FRED
FLINTSTONE
BEDROCK
BARNEY          RUBBLE          BEDROCK
```

DAYSBETW (date1, date-2**[**, **[**, format-date1**][**, format-date2**]]**)
Return days between 2 dates of a given format.

Parameters:

- **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 DATE 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.
Example:

```
1 CALL DUMP 'THIS IS THE NEW VERSION OF BREXX/370 V2R5M3','DUMP LINE'
```

Results:

```
DUMP LINE
0000(0000)  THIS  IS  THE  NEW    VERS ION  OF B REXX
0000(0000)  ECCE 4CE4 ECC4 DCE4   ECDE CDD4 DC4C 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.
Example:

```
1 CALL LISTALC
```

Results:

```
STDOUT    *terminal
STDIN     *terminal
SYSPROC   SYS1.CMDPROC
SYSHELP   SYS1.HELP
          SYS2.HELP
SYS00002  UCPUB001
RXLIB     BREXX.V2R5M3.RXLIB
SYSEXEC   SYS2.EXEC
SYS00005  UCPUB000
ISPPROF   IBMUSER.ISP.PROF
ISPMLIB   SYSGEN.ISPF.MLIB
STDERR    *terminal
ISPSLIB   SYSGEN.ISPF.SLIB
ISPCLIB   SYSGEN.ISPF.CLIB
          SYSGEN.REVIEW.CLIST
ISPLLIB   SYSGEN.ISPF.LLIB
          SYSGEN.REVIEW.LOAD
ISPTABL   SYSGEN.ISPF.TLIB
ISPPLIB   SYSGEN.ISPF.PLIB
          SYSGEN.ISPF.RFEPLIB
ISPTLIB   SYSGEN.ISPF.TLIB
REVPROF   IBMUSER.ISP.PROF
```

```
SYS00012  SYSGEN.ISPF.LLIB
SYS00013  IBMUSER.CLIST
```

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: http://home.cern.ch/~bnv

QUOTE (`string`, `qtype`)
Enclose string in quotes, double quotes, or parenthesis,

Parameters: **qtype** – can be:

- ‘ single quote (default),
- “ double quote
- *(* bracket, the closing character is ‘)’
- *[* square bracket, the closing character is ‘]’

Example:

```
1  Mystring='string to be quoted'
2  Say QUOTE(mystring,'"')
3  Say QUOTE(mystring,"'")
4  Say QUOTE(mystring,'(')
5  Say QUOTE(mystring,'[')
```

Results:

```
'STRING TO BE QUOTED'
'STRING TO BE QUOTED'
(STRING TO BE QUOTED)
[STRING TO BE QUOTED]
```

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

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.
Example:

```
1 say sec2Time(345000)
2 say sec2Time(345000,'DAYS')
```

Results:

```
95:50:00
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.*

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 DATE output-format description.

UNQUOTE (`string`)
Remove from string leading and trailing quotes, double quotes, parenthesis and '<' and '>' signs.
Example:

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

Results:

```
'QUOTED-STRING'
ENTRY 1
ENTRY 2
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.

Building TSO Commands

A BREXX function can be converted to work as a TSO command by creating a clist and call the BREXX script. To perform the new clist, it must be stored in one of the pre-allocated clists libraries which are active in your TSO session; alternatively, you can use *SYS2.CMDPROC*. Once this is done, you can call it from TSO directly.

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 ';'.

```
1 REXX -
2 CALL LISTALC('PRINT')
```

WHOAMI Display current User Id

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

```
1 REXX -
2 SAY USERID()
```

TODAY

Display today's Date

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

USERS

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

```
1 REXX WHO
```

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.

```
1 RX REPL NOSTAE
```

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.

```
1 RX EOT NOSTAE
```

Callable External Functions

With the new External Function feature, you can call compiled programs written in conventional language, as PL1, Assembler, and maybe more.

We closely adapted IBM's TSO/E REXX programming services: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.ikja300/progsrv.htm

How it works:

BREXX Call an external Program

To call an external program, you call it in the same way as a normal BREXX function:

```
say load-module(argument-1,argument-2,...,argument-15)
```

you can pass up to 15 arguments to the external function. The size of the return value can be up to 1024 bytes.

Example:

```
Say RXPI()
```

RXPI is a load module that must be accessible within the link list chain. It does not have any arguments.

BREXX Programming Services

BREXX provides control blocks containing the arguments and a 1024 bytes return buffer.

Called Program

The program needs to match the BREXX calling conventions to manage the argument and return value handling. To ease it, we have isolated communication control blocks and internal functions in a copybook. Once included, it will transparently provide the functionality to the program.

Example, PI calculation:

```
RXPI:  PROCEDURE(EFPL_PTR) OPTIONS(MAIN);
  %INCLUDE RXCOMM;
...
```

Benefits

The performance of a compiled program is much higher than in BREXX. So if you have complex mathematical calculations, they will be significantly faster than code implemented in BREXX. In our testing, we implemented an algorithm for calculating PI with 500 digits. In comparison, it was over 600 times faster than the same algorithm implemented in BREXX.

Example, PI calculation:

```
 1 RXPI:    PROCEDURE(EFPL_PTR) OPTIONS(MAIN);                          00000101
 2   %INCLUDE RXCOMM;                                                  00000201
 3  /* -----------------------------------------------------------------00000301
 4   * CALCULATE PI USING THE ALGORITHM OF S. RABINOWICZ AND S. WAGON    00000413
 5   * INPUT VARIABLES                                                  00000501
 6   *   ARGNUM      CONTAINS NUMBER OF PROVIDED ARGUMENTS (MAX 15)     00000601
 7   *   ARG(I)      CONTAINS CONTENTS OF ARGUMENT I (1 TO 15)          00000701
 8   *   ARG_LEN(I)  CONTAINS LENGTH   OF ARGUMENT I (1 TO 15)          00000801
 9   * RETURN VARIABLES                                                00000901
10   *   RESULT      CONTAINS RESULT TO BE RETURNED TO BREXX            00001001
11   *               THE RETURN VALUE MUST NOT EXCEED 1024 BYTES        00001112
12   *   RESULT_LEN  CONTAINS LENGTH OF RETURNED STRING                 00001201
13   * -----------------------------------------------------------------00001301
14   */                                                                 00001401
15   DCL (N, LEN) FIXED BINARY;                                         00001505
```

```
 16   DCL PI CHAR(512) VARYING;                                          00001611
 17   DCL TEMPI CHAR(9);                                                 00001709
 18   DCL PREDIGIT BIN FIXED(15);                                        00001806
 19   N = 500;                                                           00001912
 20   LEN = 10*N / 3;                                                    00002012
 21   PI='';                                                             00002112
 22  BEGIN;                                                              00002212
 23   DECLARE ( I, J, K, Q, NINES) BIN FIXED(15);                        00002312
 24   DECLARE X FIXED BINARY (31);                                       00002412
 25   DECLARE A(LEN) FIXED BINARY (31);                                  00002512
 26                                                                      00002600
 27   A = 2; /* START WITH 2S */                                         00002712
 28   NINES, PREDIGIT =0; /* FIRST PREDIGIT IS A 0 */                    00002812
 29   DO J = 1 TO N;                                                     00002912
 30      Q = 0;                                                          00003012
 31      DO I = LEN TO 1 BY -1; /* WORK BACKWARDS */                     00003112
 32         X = 10*A(I) + Q*I;                                           00003212
 33         A(I) = MOD (X, (2*I-1));                                     00003312
 34         Q = X / (2*I-1);                                             00003412
 35      END;                                                            00003512
 36      A(1) = MOD(Q, 10); Q = Q / 10;                                  00003612
 37      IF Q = 9 THEN NINES = NINES + 1;                                00003712
 38      ELSE IF Q = 10 THEN DO;                                         00003812
 39         TEMPI=PREDIGIT+1;                                            00003912
 40         PI=PI||SUBSTR(TEMPI,9,1);                                    00004012
 41         DO K = 1 TO NINES;                                           00004112
 42            PI=PI||'0';                                               00004212
 43         END;                                                         00004312
 44         PREDIGIT=0;                                                  00004412
 45         NINES = 0;                                                   00004512
 46      END;                                                            00004612
 47      ELSE DO;                                                        00004712
 48         TEMPI=PREDIGIT;                                              00004812
 49         PI=PI||SUBSTR(TEMPI,9,1);                                    00004912
 50         PREDIGIT = Q;                                                00005012
 51         DO K = 1 TO NINES;                                           00005112
 52            PI=PI||'9';                                               00005212
 53         END;                                                         00005312
 54         NINES = 0;                                                   00005412
 55      END;                                                            00005512
 56    END;                                                              00005610
 57  END ; /* END BEGIN */                                               00005712
 58    TEMPI=PREDIGIT;                                                   00005812
 59    PI=PI||SUBSTR(TEMPI,9,1);                                         00005910
 60  RESULT='3.'||SUBSTR(PI,3);                                          00006012
 61  RESULT_LEN=LENGTH(PI);                                              00006112
 62  END RXPI;                                                           00006201
```

BREXX Version of the PI calculation program:

```
 1 /* ---------------------------------------------------------------------
 2  * PI USING THE ALGORITHM OF S. RABINOWICZ AND S. WAGON
 3  * ---------------------------------------------------------------------
 4  */
 5 RXPIR:
 6   N = 500
 7   LEN = (10*N/3)%1
 8   A.=2
 9   NINES=0
10   PREDIGIT = 0 /* FIRST PREDIGIT IS A 0 */
11   DO J = 1 TO N
```

```
12      Q = 0
13      DO I = LEN TO 1 BY -1 /* WORK BACKWARDS */
14         X = INT(10*A.I + Q*I)
15         A.I = INT(X//(2*I-1))
16         Q = X%(2*I-1)
17      END
18      A.1 = (Q//10)%1
19      Q = Q % 10
20      IF Q = 9 THEN NINES = NINES + 1
21      ELSE IF Q = 10 THEN DO
22         PI=PI||PREDIGIT+1
23         PI=PI||COPIES('0',NINES)
24         PREDIGIT= 0
25         NINES = 0
26      END
27      ELSE DO
28         PI=PI||PREDIGIT
29         PREDIGIT = Q
30         PI=PI||COPIES('9',NINES)
31         NINES = 0
32      END
33   END
34   PI=PI||PREDIGIT
35 RETURN '3.'SUBSTR(PI,3)
```

Comparison of both implementations:

PL1 Program::

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620
89986280348253421170679821480865132823066470938446095505822317253594081284811 17
45028410270193852110555964462294895493038196442881097566593344612847564823378 67
83165271201909145648566923460348610454326648213393607260249141273724587006606 31
55881748815209209628292540917153643678925903600113305305488204665213841469519 41
51160943305727036575959195309218611738193261179310511854807446237996274956735 18
85752724891227938183011949 1
```

Elapsed Time 0.49016099452972417 seconds

BREXX Program::

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620
89986280348253421170679821480865132823066470938446095505822317253594081284811 17
45028410270193852110555964462294895493038196442881097566593344612847564823378 67
83165271201909145648566923460348610454326648213393607260249141273724587006606 31
55881748815209209628292540917153643678925903600113305305488204665213841469519 41
51160943305727036575959195309218611738193261179310511854807446237996274956735 18
85752724891227938183011949 1
```

Elapsed Time 300.3606059551243 seconds

For the hardcore programmer The current content of the Communication Interface follows. The long-winded coding is caused by the functionality of the old PL1-360-F compiler.

```
 1 /* ----------------------------------------------------------------------00000500
 2  * REXX INTERFACE BLOCK  EFPL                                           00000600
 3  * ----------------------------------------------------------------------00000700
 4  */                                                                     00000800
 5   DCL EFPL_PTR PTR;                                                     00000900
 6   DCL 1 EFPL BASED(EFPL_PTR),                                           00001000
 7         2 EFPLCOM  FIXED BIN(31),                                       00001100
 8         2 EFPLBARG FIXED BIN(31),                                       00001200
 9         2 EFPLEARG FIXED BIN(31),                                       00001300
10         2 EFPLFB   FIXED BIN(31),                                       00001400
11         2 EFPLARG  PTR,                                                 00001500
```

```
12         2 EFPLEVAL PTR;                                                 00001600
13  /* ---------------------------------------------------------------------00001700
14   * ARGTABLE ENTRIES AND RELATED DEFINITIONS                             00001800
15   * ---------------------------------------------------------------------00001900
16   */                                                                     00002000
17    DCL EFPLARG_PTR      PTR;                                            00002100
18    EFPLARG_PTR        = EFPLARG;                                        00002200
19    DCL 1 ARGTABLE BASED(EFPLARG_PTR),                                   00002300
20          2 ARGTABLE_ENTRY(15),                                          00002402
21            3 ARGSTRING_PTR    PTR,                                      00002500
22            3 ARGSTRING_LENGTH FIXED BIN(31);                           00002600
23                                                                         00002704
24    DCL ARGNUM      BIN FIXED(31);                                       00002810
25    DCL ARG_LEN(15) BIN FIXED(31);                                       00002908
26    DCL ARG(15)     CHAR(255) VARYING;                                   00003010
27                                                                         00003104
28    DCL ARG_PTR PTR;                                                     00003200
29    DCL ARGSTRING CHAR(255) BASED(ARG_PTR);                              00003300
30  /* ---------------------------------------------------------------------00003400
31   * EVALUATION BLOCK: EVALBLOCK                                          00003500
32   * ---------------------------------------------------------------------00003600
33   */                                                                     00003700
34    DCL EFPLEVAL_ADR_PTR PTR;                                            00003800
35    DCL EFPLEVAL_PTR     PTR;                                            00003900
36    EFPLEVAL_ADR_PTR     = EFPLEVAL;                                     00004000
37                                                                         00004104
38    DCL 1 EVALBLOCK_ADR BASED(EFPLEVAL_ADR_PTR),                         00004200
39          2 EFPLEVAL_ADR PTR;                                            00004300
40                                                                         00004400
41    EFPLEVAL_PTR = EFPLEVAL_ADR;                                         00004500
42                                                                         00004600
43    DCL 1 EVALBLOCK BASED(EFPLEVAL_PTR),                                 00004700
44          2 EVALBLOCK_EVPAD1 FIXED BIN(31),                              00004800
45          2 EVALBLOCK_EVSIZE FIXED BIN(31),                              00004900
46          2 EVALBLOCK_EVLEN  FIXED BIN(31),                              00005000
47          2 EVALBLOCK_EVPAD2 FIXED BIN(31),                              00005100
48          2 EVALBLOCK_EVDATA CHAR(256);                                  00005200
49                                                                         00005300
50    DCL EVDATA_PTR PTR;                                                  00005400
51    DCL EVDATLN_PTR PTR;                                                 00005500
52                                                                         00005600
53    EVDATA_PTR = ADDR(EVALBLOCK_EVDATA);                                 00005700
54    EVDATLN_PTR = ADDR(EVALBLOCK_EVLEN);                                 00005800
55                                                                         00005900
56    DCL RESULT CHAR(1024) BASED (EVDATA_PTR);                            00006009
57    DCL RESULT_LEN    BIN FIXED(31) BASED(EVDATLN_PTR);                  00006100
58                                                                         00006200
59    RESULT_LEN = 1;                                                      00006300
60  /* ---------------------------------------------------------------------00006400
61   * COPY BREXX PARMS INTO PL1 STRUCTURE                                  00006503
62   * ---------------------------------------------------------------------00006600
63   */                                                                     00006703
64    DCL AI BIN FIXED(31);                                                00006804
65                                                                         00006904
66    DO AI=1 TO 15 ;                                                      00007004
67       ARG_PTR = ARGSTRING_PTR(AI);                                      00007103
68       ARG_LEN(AI) = ARGSTRING_LENGTH(AI);                               00007208
69       IF ARG_LEN(AI)<=0 THEN ARG(AI)='';                                00007308
70       ELSE DO                                                           00007403
71          ARG(AI) = SUBSTR(ARGSTRING,1,ARG_LEN(AI));                     00007508
```

```
 72          ARGNUM=AI;                                                  00007605
 73        END;                                                          00007705
 74    END;
```

VSAM User's Guide

The VSAM User's Guide contains the BREXX functions to access VSAM KSDS files.

The VSAM Interface is based on Steve Scott's VSAM API: https://sourceforge.net/projects/rxvsam/ .

We gratefully thank Steve for allowing us the integration in BREXX and his support to achieve it.

The underlying VSAM API allows full support for KSDS, RRDS and ESDS, but we focused just on the KSDS functionality, so there is no support for RRDS and ESDS. If this limitation is lifted in the future depends on user requests.

Integration of the VSAM Interface in BREXX

We decided to integrate the interface as host commands rather than BREXX functions. It is now similar to the EXECIO host command for sequential datasets. The host command name is VSAMIO. Host commands are typically enclosed in quotes or double-quotes.

Example:

"VSAMIO OPEN VSIN (UPDATE"

Limitations/Restrictions

The implementation has only tested with UNIQUE cluster definitions, not with type SUBALLOCATION (which requires in MVS 3.8 a DEFINE SPACE and DEFINE VSAM catalogue definition). The UNIQUE specification does not allow the REUSE CLUSTER definition, which would be necessary for the initial loading of an empty KSDS dataset.

Initialising empty VSAM Files

A program cannot directly process an empty VSAM file it must be initialised first. The procedure to achieve this is to use IDCAMS REPRO to write a "null"-record into it. After this exercise, the VSAM file can be updated by BREXX/370 with normal VSAM Write commands from BREXX.

Key of Records

The key must be part of the record, and nevertheless, you must additionally specify the key in the following commands:

- *"VSAMIO READ ddname (KEY key … "*
- *"VSAMIO LOCATE ddname (KEY key … "*
- *"VSAMIO WRITE ddname (KEY key … "*
- *"VSAMIO DELETE ddname (KEY key … "*

The key of a record must consist of a sequence of contiguous non-space characters; this means blanks are not allowed being part of a key. This limitation might be lifted in one of the forthcoming releases.

You can easily convert spaces in a key with the TRANSLATE function: *key=translate(key,'_',' ')*

Return Codes

Each VSAMIO command call returns two return codes:

- **RC** the usual return code, containing:

Return Code	Description

0	call was successful
4	call was not successful and ended with warnings, typically in record-not-found Situations
8	call ended with errors

- **RCX** The extended VSAM Return code, and it consists of a 9 character field with the following format: *rrr-vvvvv* **rrr** is the function return code, **vvvvv** is the VSAM return code

You can look up the details of the extended VSAM return code in IBM's MVS System Messages under message IDC3351I.

System Abend A03

The RXVSAM API runs as independent subtask within the address space. By the end of the REXX Script, an automatic shutdown of the subtask is performed. If the REXX script unexpectedly terminates, you possibly see a SYSTEM ABEND A03, which means the main task (BREXX) has been terminated and there is still a subtask in the background active. MVS forces the ABEND of the subtask with A03. There are no further actions required; there is no impact on the system or the VSAM datasets.

Random and Sequential Access

The used VSAM IO module distinguishes two access methods:

- Random Access always requires a key to read/write/delete a record
- Sequential Access allows to position to a particular record and reads/write/delete records from there sequentially

Both methods can be used concurrently, but it is essential to understand that they do not mutual interfere. Having read a record with random access does not allow to read from this record sequentially the next records, as this is sequential access. But you can perform a LOCATE command with a key and continue the read from there sequentially.

VSAM Dataset reference

Each VSAMIO command uses the DDNAME as a reference to the VSAM dataset. It must be pre-allocated via a JCL DD Statement or a TSO ALLOCATE command.

There are no plans to allow a dataset name (DSN) instead of the DDNAME!

REXX VSAM Debugging

By using BREXXDBG as the BREXX interpreter you can produce additional log entries in the operator's console, as well as in the spool output of a batch job:

Example:

```
 1 //BRXVSMKY JOB CLASS=A,MSGCLASS=H,REGION=8192K,
 2 //         NOTIFY=&SYSUID
 3 //*
 4 //* -------------------------------------------------------------
 5 //* READ STUDENT VSAM FILE VIA KEY
 6 //* -------------------------------------------------------------
 7 //*
 8 //BATCH EXEC RXTSO,BREXX='BREXXDBG',
 9 //         EXEC='$STUDENK',
10 //         SLIB='BREXX.V2R5M3.SAMPLES'
11 //SYSPRINT DD SYSOUT=*,
12 //         DCB=(RECFM=FBA,LRECL=133,BLKSIZE=133)
13 //SYSUDUMP DD SYSOUT=*
14 //
```

Results:

```
07.35.01 JOB 1466 $HASP373 PEJRXKEY STARTED - INIT 1 - CLASS A - SYS
TK4-
07.35.01 JOB 1466 IEF403I PEJRXKEY - STARTED - TIME=07.35.01
07.35.02 JOB 1466 +VSAMIO - STUDENTM ACCESS TRACE, REQUEST = OPEN
07.35.02 JOB 1466 +VSAMIO - KEY=NONE
07.35.02 JOB 1466 +VSAMIO - STUDENTM ACCESS TRACE, REQUEST = READU
07.35.02 JOB 1466 +VSAMIO - KEY=X"C1D5C4C5D9E2D6D55EC2C5D56D6D6D6D6D6D6D6D6D6
07.35.02 JOB 1466 +VSAMIO - STUDENTM ACCESS TRACE, REQUEST = READU
07.35.02 JOB 1466 +VSAMIO - KEY=X"C1D5C4C5D9E2D6D55EC7C1C2D9C9C5D36D6D6D6D6D6
07.35.02 JOB 1466 +VSAMIO - STUDENTM ACCESS TRACE, REQUEST = READU
07.35.02 JOB 1466 +VSAMIO - KEY=X"C2C1D3C4E6C9D55EC1D9D3C5D5C56D6D6D6D6D6D6D6
07.35.02 JOB 1466 +VSAMIO - STUDENTM ACCESS TRACE, REQUEST = READU
07.35.02 JOB 1466 +VSAMIO - KEY=X"E2E3C5D7C8C5D5E2D6D55ED7C1E3D9C9C3C9C16D6D6D6
07.35.02 JOB 1466 +VSAMIO - STUDENTM ACCESS TRACE, REQUEST = CLOSE
07.35.02 JOB 1466 +VSAMIO - KEY=NONE
07.35.02 JOB 1466 IEFACTRT - Stepname Procstep Program  Retcode
07.35.02 JOB 1466 PEJRXKEY   BATCH    EXEC     IKJEFT01 RC= 0000
07.35.02 JOB 1466 IEF404I PEJRXKEY - ENDED - TIME=07.35.02
07.35.02 JOB 1466 $HASP395 PEJRXKEY ENDED
```

VSAM Commands in BREXX

OPEN VSAM Dataset

"VSAMIO OPEN ddname ([READ/UPDATE] "

Example:

```
"VSAMIO OPEN VSIN1 (READ"
"VSAMIO OPEN VSIN2 (UPDATE"
```

VSIN1 is opened in reading mode, VSIN2 in UPDATE mode.

READ with KEY

Access-Type: Random:

```
"VSAMIO READ ddname (KEY key-to-read VAR rexx-variable"
```

If you want to update the record, you must prepare for it by adding the UPDATE keyword:

```
"VSAMIO READ ddname (KEY key-to-read UPDATE VAR rexx-variable"
```

The UPDATE keyword requires a File OPEN with UPDATE

Example:

```
"VSAMIO READ VSIN1 (KEY "key1" VAR record1"
"VSAMIO READ VSIN2 (KEY "key2" UPDATE VAR record2"
```

Read a record with key1/key2 (contained in a rexx variable) into the rexx variable record1/record2

READ NEXT

Access-Type: Sequential

After positioning with LOCATE to a particular record, you can read the next records sequentially. If no LOCATE has been previously performed, the first record is read.:

```
"VSAMIO READ ddname (NEXT VAR rexx-variable"
```

If you want to update the record, you must prepare for it by adding the UPDATE keyword:

```
"VSAMIO READ ddname (NEXT UPDATE VAR rexx-variable"
```

The UPDATE keyword requires a File OPEN with UPDATE

Example:

```
"VSAMIO LOCATE VSIN (KEY "key
Do until rc>0
    "VSAMIO READ VSIN (NEXT VAR record"
    Say record
End
```

Position to record key (contained in a rexx variable) and read all records from there into rexx variable record

LOCATE position to a certain record

Access-Type: Sequential

Position the record pointer in front of the provided key or a key prefix:

```
"VSAMIO LOCATE ddname (KEY [key-to-position/key-prefix]"
```

To subsequentially read the next records a READ NEXT is required. After a successful read, the position is shifted to the next record position.

Example refer to READ NEXT.

WRITE KEY

Access-Type: Random

To update a record, it must be priorly read with a READ KEY, regardless whether the record exists. If the record doesn't exist, it is inserted:

```
"VSAMIO WRITE ddname (KEY key-to-write VAR rexx-variable"
```

Example:

```
"VSAMIO READ VSIN (KEY "key" UPDATE VAR CURRENT"
say 'READ 'rc' Extended RC 'rcx
"VSAMIO WRITE VSIN (KEY "key" VAR RECORD"
if rc<>0 then say key' Error during Insert'
else say inkey' Record inserted'
say 'WRITE 'rc' Extended RC 'rcx
```

To insert a new record; the READ is mandatory to verify if a record is already defined.

WRITE NEXT

Access-Type: Sequential To update a record, it must be priorly read with a READ NEXT.:

```
"VSAMIO WRITE ddname (NEXT VAR rexx-variable"
```

DELETE KEY

Access-Type: Random

To delete an existing record.:

```
"VSAMIO DELETE ddname (KEY key-to-delete "
```

Example:

```
"VSAMIO OPEN VSERR (UPDATE"
say 'OPEN 'rc' Extended RC 'rcx
"VSAMIO DELETE VSERR (KEY 0000000"
say 'Delete Dummy Record 'rc' Extended RC 'rcx
```

DELETE NEXT

Access-Type: Sequential

To delete an existing record, it must be priorly read with a READ NEXT.:

```
"VSAMIO DELETE ddname (NEXT "
```

Example:

```
"VSAMIO LOCATE VSIN (KEY "prefix
say "LOCATE "rc
say "Extended RC "rcx
do forever
    "VSAMIO READ VSIN (NEXT UPDATE VAR INREC"
    if rc<>0 then leave
    say "record='"INREC"' RC "rc" Extended RC "rcx
    key=substr(inrec,1,8)
    "VSAMIO DELETE VSIN (NEXT "
    if rc=0 then reci=reci+1
    say 'DELETE RC 'rc' Extended RC 'rcx
end
```

CLOSE

"VSAMIO CLOSE ddname "

To close all open VSAM datasets you can also use *"VSAMIO CLOSE ALL "*

Example:

```
"VSAMIO CLOSE VSERR"
```

BREXX VSAM Example

The installation file contains in the dataset *BREXX.V2R5M3.JCL* a working example of a student database using fictitious student entries, containing first name, family name, birth date, the field of study, address.

You can submit the REXX scripts in batch out of *BREXX.V2R5M3.JCL*

Member	Description
STUDENTC	Creates the VSAM Cluster definition
STUDENTI	Inserts the student records into the VSAM dataset
STUDENTK	Read the VSAM dataset with KEYs
STUDENTN	Read the VSAM dataset sequentially

The REXX scripts are stored in *BREXX.V2R5M3.SAMPLES*

Member	Description
@STUDENI	insert student records
@STUDENK	read student records by key
@STUDENL	Query student records by using formatted screens
@STUDENN	read student records sequentially

The following example illustrates the definition and population of a VSAM dataset using BREXX.

1. Define a VSAM Cluster: Define a new VSAM Cluster and import a "Null"-Record, submit the job *STUDENTC*
2. Sample BREXX Program to update the VSAM Dataset: run the script *@STUDENI*
3. JCL Upate VSAM Dataset: The BREXX Program is updating the new VSAM Dataset. Submit the job *STUDENTI*

4. Using a Formatted Screen Application to Query the Student File: *TSO RX "BREXX.V2R5M3.SAMPLE(@SUTDENTL)"*

BREXX KEY/VALUE Database

Introduction

Accessing the underlying VSAM KSDS is made simple by using the Key/Value Database. You do not require extensive VSAM understanding nor do you need to be familiar with the database's intricate structure. The Key/Value Database is standardised.

The system consists of two VSAM files: a source and a reference database. The reference DB facilitates source record linking of the K/V database. This makes it possible to implement simple databases like hierarchical or graph databases.

Overview of the VSAM Key structure

The K/V functions handle everything; memorizing the VSAM structure is not required, but it should not be overlooked for completeness.

The VSAM structure in an overview, the partition, which will be called the room, is 2 bytes long, the qualifier is 10 bytes long, and the key itself is 32 bytes long, making the total key length 44 bytes. Therefore, having the same key but different qualifiers is possible. Being in multiple rooms allows you to make use of the same key and qualification.

Any changes you make to these default settings will necessitate modifications in the BREXX module DBPROF, which is stored in your RXLIB. To address records, use the key element, for example:

```
Call DBSET([qualifier.]key, record)
Call DBGET([qualifier.]key)
Say Result
```

The qualifier is always converted to **lowercase**. The key is case-sensitive unless you set in your database profile (usually DBPROF) **ddprof.keyupper to 1**.

The qualifier is optional and, if not specified, will be set to **any**. If the qualifier and key are shorter than their definitions, they will be suffixed with "_" to achieve the maximum length feasible.

Installing the Key/Value Database Functions

The Key/Value program is included with BREXX and does not require any additional installation. However, you must configure and submit the VSAM Clusters in the member $CREKEYV in the BREXX installation dataset. For more details, see Step 6 in the BREXX installation dataset manual.

Key/Value Modules

Library content:

```
DBOPEN   Key/Value REXX contains all procedures
DBPROF   Key/Value profile with essential information
SAMPLE1  Key/Value sample to test it
$CREKEYV as part of the BREXX installation library
```

Customisation (optional)

Aside from the fundamental parameters (names and lengths), the PDS member DBPROF allows you to modify the behaviour. Especially you can instruct the DBLINK function to construct dummy source records if one of the records to be connected does not exist. Another option is to have it translate the key into uppercase. This is the default setting.

```
ddprof.EnableDummy=1             /* allow links between records   */
                                 /* which do not exist, they will */
                                 /* create DUMMY source records   */
```

```
                                /* 0: no, 1: yes                */

ddprof.keyupper=0               /* upper case translation of the */
                                /* key. 0: no, 1: yes           */
```

Loading the sample Key/Value Database

A sample is included in the BREXX installation and can be loaded using the REXX: *BREXX.V2R5M3.SAMPLE(DBWORLD)*. It includes all countries in the world, a few trade unions, and the country's main cities. It is used in the following examples to demonstrate how the K/V and reference databases work.

> KV160I Check-in Standard Room (AA) KV120I Key/Value DB successfully opened 10:25:20.626631 Start loading Countries 10:25:29.178898 Loading Countries completed, 203 loaded 10:25:29.180352 Link them to Trade Unions 10:25:31.231292 Link to Trade Unions completed 10:25:31.603774 Start loading Cities and Link them to their Country 10:25:51.079262 loading Cities completed, 774 loaded

Key/Value Database Functions

To use the Key/Value database functions you need to import KEYVALUE (part of your RXLIB).

> CALL IMPORT KEYVALUE

Let us start with a simple example, it should be self-explanatory:

```
 1 call import keyvalue
 2 say "OPEN  "DBOPEN()
 3
 4 /* Add Continents  */
 5    Call DBSET('Continent.Europe','Continent Europe')
 6    Call DBSET('Continent.Asia','Continent Asia')
 7    Call DBSET('Continent.Africa','Continent Africa')
 8    Call DBSET('Continent.North America','Continent North America')
 9    Call DBSET('Continent.South America','Continent South America')
10    Call DBSET('Continent.Australia','Australia and Oceania')
11
12 /* Get Continents  */
13    call DBGET('Continent.Europe')
14    say dbresult
15    call DBGET('Continent.Asia')
16    say dbresult
17    call DBGET('Continent.Africa')
18    say dbresult
19    call DBGET('Continent.North America')
20    say dbresult
21    call DBGET('Continent.South America')
22    say dbresult
23    call DBGET('Continent.Australia')
24    say dbresult
25    say "CLOSE "DBCLOSE()
26 exit
```

Result:

```
KV160I    Check-in Standard Room  (AA)
KV120I    Key/Value DB successfully opened
OPEN 0
Continent Europe
Continent Asia
Continent Africa
Continent North America
Continent South America
```

```
Australia and Oceania
CLOSE 0
```

VSAM Functions

DBOPEN ()
Opens the key/value VSAM dataset.
Opens the standard Key/Value Database. If DBOPEN was successful an RC of zero is returned, else it failed.

```
1 say "OPEN  "DBOPEN() ->  OPEN  0
```

DBROOM (room-name)
Assigns a room.
The DBROOM function divides the Key/Value Database into distinct areas that can be projects, applications, or anything else. This means that you have many Key/Value Databases in a single VSAM dataset.
The definition of a room is optional if not defined it automatically switches into a standard room. Use the DBROOM command to define a new room or to reuse an existing one. If the room is not already there, it will be created and recorded in the Key/Value database's control record. There will always be an uppercase translation of the room name.
By designating a room, you can specify the space where the following commands take effect.
You can define the same Key of a record more than once by using various rooms. Since they cannot see one another, rooms cannot communicate.
For example, write records (with the same key) in different rooms and report them.

```
 1 call import 'KeyValue'
 2 call DBOPEN
 3 call DBROOM 'Beverages'
 4 say '*** Room Beverages ***'
 5 call DBSET('Beer','Munich Hofbraeuhaus')
 6 call DBSET('Whisky','Black Label')
 7 call DBGET('Whisky')
 8    say left(dbkey,8)': ' dbresult
 9 call DBGET('Beer')
10    say left(dbkey,8)': ' dbresult
11 call DBROOM 'Booze'
12 say '*** Room Booze     ***'
13 call DBSET('Beer','Guiness')
14 call DBSET('Whisky','Bushmills')
15 call DBGET('Whisky')
16    say left(dbkey,8)': ' dbresult
17 call DBGET('Beer')
18    say left(dbkey,8)': ' dbresult
19 call DBROOM 'Beverages'
20 say '*** Room Beverages ***'
21 call DBGET('Whisky')
22    say left(dbkey,8)': ' dbresult
23 call DBGET('Beer')
24    say left(dbkey,8)': ' dbresult
25 call dbclose
```

Result:

```
KV160I    Check-in Standard Room  (AA)
KV120I    Key/Value DB successfully opened
*** Room Beverages ***
Whisky  :  Black Label
Beer    :  Munich Hofbraeuhaus
*** Room Booze     ***
Whisky  :  Bushmills
Beer    :  Guiness
*** Room Beverages ***
```

```
Whisky  :  Black Label
Beer    :  Munich Hofbraeuhaus
```

DBCLOSE ()
Closes the Key/Value Database.

```
1 say "CLOSE "DBCLOSE()  ->        CLOSE 0
```

DBSET ([qualifier.]key, value)
Writes a key/value record to the DB.
The record indicated by the qualifier and key arguments will be saved into the Key/Value Database. If the record is already present, it will be overwritten.
A BREXX DCL data structure can be used to compose the record's various components. Go to the BREXX User's Guide for further information. *DCL('structure-name,'$DEFINE')*. The qualifier parameter defaults to "any".
Return codes: - 0 Put successfully - 8 Put failed
Variables: - *dbKey* contains the key - *dbFKey* contains the full key (including the qualifier, key) - *dbResult* contains the full record (including the full key) - *dbRC* return code of the function
For example, adding continents to the database:

```
1 /* Add Continents  */
2    SAY DBSET('Cont.Europe','Continent Europe')
3    SAY DBSET('Cont.Asia','Continent Asia')
4    SAY DBSET('Cont.Africa','Continent Africa')
5    SAY DBSET('Cont.North America','Continent North America')
6    SAY DBSET('Cont.South America','Continent South America')
7    SAY DBSET('Cont.Australia','Australia and Oceania')
```

DBGET ([qualifier].key)
Reads a key/value record from the DB.
A BREXX DCL data structure can be used to compose the record's various components. Go to the BREXX User's Guide for further information. DCL('structure-name,'$DEFINE'). The qualifier parameter defaults to "any".
Return code: - 0 Get successfully - 8 Get failed
Variables: - *dbKey* contains the key - *dbFKey* contains the full key (including the qualifier, key) - *dbResult* contains the full record (including the full key) - *dbRC* return code of the function
For example, reading the Continents

```
 1 /* Get Continents  */
 2 call DBGET('Cont.Europe')
 3 say dbresult
 4 call DBGET('Cont.Asia')
 5 say dbresult
 6 call DBGET('Cont.Africa')
 7 say dbresult
 8 call DBGET('Cont.North America')
 9 say dbresult
10 call DBGET('Cont.South America')
11 say dbresult
12 call DBGET('Cont.Australia')
13 say dbresult
```

Results:

```
Continent Europe
Continent Asia
Continent Africa
Continent North America
Continent South America
Australia and Oceania
```

DBDEL ([qualifier.]key)
Deletes a record specified by the qualifier and key parameters. The qualifier parameter defaults to "any".

Return code: - 0 Get successfully - 8 Get failed
Variables: - *dbKey* contains the key - *dbFKey* contains the full key (including the qualifier, key) - *dbResult* contains the full record (including the full key) - *dbRC* return code of the function

```
1 /* Remove Europe and Asia, add Eurasia instead */
2 say "Delete "DBDEL('Cont.Europe')
3 say dbresult
4 say "Delete "DBDEL('Cont.Asia')
5 say dbresult
6 call DBSET('Cont.Eurasia','Continent of Eurasia')
7 say dbresult
8 call DBGET('Cont.Eurasia','Continent of Eurasia')
9 say dbresult
```

Results:

```
Delete 0
CONT____Europe_____________________________Continent Europe
Delete 0
CONT____Asia_______________________________Continent Asia
CONT____Eurasia____________________________Continent of Eurasia
Continent of Eurasia
```

DBREMOVE (**[**, QUALIFIER/ALL/ONLY/ANY/CONTAINS**]**, string)
Removes records of a project.
REMOVE records of the actual room.
Keywords: - *ALL* removes all members of the active room - *QUALIFIER* removes all records of the qualifier defined in the string parameter - *ONLY* removes all records whose key starts with the given string. The key is the plain key without the qualifier - *ANY* removes all records whose key contains the string parameter. The key is the plain key without the qualifier - *CONTAINS* removes all records whose record content contains string parameter

> **Note**
>
> There is no function to remove all records of the VSAM dataset.

```
1 call import KeyValue
2 call dbmsglv 'N'
3 say "OPEN  "DBOPEN()              /* Open Key/Value Database */
4 say "ROOM  "DBROOM('WORLD')       /* switch to WORLD         */
5 call dbremove('QUA',"Continent") /* Remove records w. qualifier Continent */
6 call dbremove('ANY',"Mu")        /* Remove recs containing Mu in the key */
7 call dbremove('ONLY',"Wa")       /* Remove recs with starting key Wa */
8 call dbremove('ALL')             /* Remove all recs of the active room */
9 say "CLOSE "DBCLOSE()
```

Result:

```
OPEN  4
ROOM  0
Remove all records of room WORLD(AB) with Qualifier continent
------------------------------------------------------------------------
continent.Africa removed
continent.Asia removed
continent.Australia removed
continent.Europe removed
continent.North_America removed
continent.South_America removed
Number of records removed 6

Remove records of room WORLD(AB) containing Mu in key
------------------------------------------------------------------------
```

```
city.Multan removed
city.Mumbai removed
city.Munich removed
city.Murcia removed
city.Muscat removed
Number of records removed 5

Remove records of room WORLD(AB) with starting key Wa
------------------------------------------------------------------------
city.Warsaw removed
city.Washington removed
city.Washington_DC removed
Number of records removed 3

Remove all records of room WORLD(AB)
------------------------------------------------------------------------
city.`Ajman removed
city.Aarhus removed
city.Aba removed
city.Abidjan removed
city.Abomey-Calavi removed
city.Abu_Dhabi removed
city.Abuja removed
city.Accra removed
city.Ad_Dammam removed
city.Adana removed
city.Addis_Ababa removed
...
country.Zambia removed
country.Zimbabwe removed
union.BRICS removed
union.Common_Wealth removed
union.European_Union removed
union.North_American_Free_Trade removed
union.Southeast_Asian_Nations removed
Number of records removed 1036

CLOSE 0
```

DBLOCATE ([[qualifier.]key/key-prefix])

Positions to a key/value record.

Positions the VSAM to the first occurrence of the qualifier and key. If the key parameter does not exist or is the prefix of a key, LOCATE positions it to the key which comes next.

The records can be read sequentially by DBNEXT. The qualifier parameter defaults to “any”.

Return code: - 0 Get successfully - 8 Get failed

Variables: - *dbKey* contains the key - *dbFKey* contains the full key (including the qualifier, key) - *dbRC* return code of the function

DBNEXT (**[**, ALL**]**)

Reads the next key/value record.

Following a DBLOCATE, the first, or next, record will be read. It will be returned as the next record if it matches the key given in DBLOCATE; otherwise, ALL must be supplied as a parameter. Any records that come after will be returned. The ending must be controlled by your REXX script. After the last record, the EOF condition will be reported

The qualifier parameter defaults to “any”.

Return code: - 0 Get successfully - 8 Get failed

Variables: - *dbKey* contains the key - *dbFKey* contains the full key (including the qualifier, key) - *dbResult* contains the full record (including the full key) - *dbRC* return code of the function

```
1 /* Locate North America and read all continents following */
2 say DBLOCATE('Cont.North America')
```

```
3 do forever
4    if DBNEXT()>0 then leave
5    say DBRESULT
6 end
```

Results:

```
Continent North America
Continent South America
```

DBLINK ([qualifier.]key,[qualifier.]key, link-type)
Link two key/value records.
DBLINK produces a record in the Reference Dataset that contains both keys and the link-type as the reference key. Both records must exist. This allows the navigation between the source records.
The qualifier parameters default to “any”.
Return code : - 0 Both records successfully linked - 8 link failed, check the existence of source and target record
Variables: - *dbRC* return code of the function - *dbLHS* full-key of the source-record - *dbRHS* full-key of the target-record

DBDELREF ([qualifier.]key,[qualifier.]key, link-type)
de-link two key/value records
DBDELREF removes the link between two source records. It is the counterpart to DBLINK. The qualifier parameters default to “any”.
Return code : - 0 link successfully removed - 8 link remove failed
Variables: - *dbRC* return code of the function - *dbLHS* full-key of the source-record - *dbRHS* full-key of the target-record

DBDELREFALL ([qualifier.]key)
de-link all references of a source record
All links that were started from a source record are removed by DBDELREFALL. Links initiated from another source record to this one, are not removed. The default value for the qualifier is “any”.
Return code : - 0 links successfully removed - 8 links remove failed

DBPRINT ([qualifier.]key)
Prints VSAM KEY Value Record including created References. If an information model is defined it also splits the record in its attributes.
Example:

```
DBPRINT country.U.S.A

country.U.S.A
// ----------------------------------------
// Source available
//            Attributes
// ----------------------------------------
 *ACRONYM           USA
 *CAPITAL           Washington DC
 *DESCRIPTION       ?_
 *VISITED           ?_
// ----------------------------------------
//            Links to other Records
// ----------------------------------------
 >CAPITAL-IS        city.WASHINGTON DC
 >CONTAINED-IN      continent.NORTH AMERICA
 >MEMBER-OF         union.NORTH AMERICAN FREE TRADE
```

DBOUTARRAY (array-number)
stores all created output of a command in an array
Example:

```
s4=screate(250)                  /* create an array */
call setg('dbOutArray',s4)       /* assign it to dboutarray */
call dbprint(country,'DETAILS')  /* run command, output stored in array */
```

```
call setg('dbOutArray','')       /* reset assignment, next commands will be
                                    normally displayed */
```

DBSAY (`output-line`)
Prints the line (like SAY does), but DBOUTARRAY can also redirect it to an array. This lets you mix your output with any Key/Value command.

DBRCOUNT (`[qualifier.]key,'REFERENCES/USAGES'`)
Counts the references or usages of a specific Key/Value record.

Report and Maintenance Functions

DBLIST (**[**, **[**, `QUALIFIER/ONLY/ANY/CONTAINS`**]**, `string`**]**)
List all records using the keyword (1. Parameter) and string combination.
If DBLIST is run without any parameters, it returns all entries for the active room.
Keyword: - *QUALIFIER* lists all records of the qualifier defined in the string parameter - *ONLY* lists all records whose key starts with the string parameter. The key is the plain key without the qualifier - *ANY* lists all records whose key contains the string parameter. The key is the plain key without the qualifier - *CONTAINS* lists all records whose record content contains string parameter
The DBLIST Format:

```
List records of room WORLD(AB) with starting key Wa
------------------------------------------------------------------------
 city.Wad Medani                              Source 265124
 city.Warsaw                                  Source 1428379
 city.Washington                              Source 3693775
 city.Washington DC                           Dummy  DUMMY
List contains 5 records
```

- The first column contains the qualifier and the key
- The second column contains either Source or Dummy. - Source means a source record was inserted via DBSET (or equivalent command) - Dummy means no source record was inserted yet, but the name has been reserved by a
- DBREFERENCE command: The third column contains the source record

Some examples:

```
1 CALL DBLIST()
```

Result:

```
List all records of room WORLD(AB)
------------------------------------------------------------------------
 any.AUS                                      Dummy  DUMMY
 any.Canberra                                 Dummy  DUMMY
 any.Mainland of the Australian continent     Dummy  DUMMY
 any.USA                                      Dummy  DUMMY
 any.Washington DC                            Dummy  DUMMY
 any.YES                                      Dummy  DUMMY
 city.`Ajman                                  Source 376263
 city.Aarhus                                  Source 219041
 city.Aba                                     Source 1174779
 city.Abidjan                                 Source 3823793
 city.Abomey-Calavi                           Source 503669
 city.Abu Dhabi                               Source 1138691
 city.Abuja                                   Source 2894719
 city.Accra                                   Source 1833578
 city.Ad Dammam                               Source 693590
 city.Adana                                   Source 1355973
 city.Addis Ababa                             Source 2334972
```

```
 city.Adelaide                                  Source 994888
 city.Aden                                      Source 389562
 city.Aguadilla                                 Source 199889
...
```

```
1 call dblist('QUALIFIER',"Continent")
```

Result:

```
List all records of room WORLD(AB) with Qualifier continent
-----------------------------------------------------------------------
 continent.Africa                               Dummy  DUMMY
 continent.Asia                                 Dummy  DUMMY
 continent.Australia                            Dummy  DUMMY
 continent.Europe                               Dummy  DUMMY
 continent.North America                        Dummy  DUMMY
 continent.South America                        Dummy  DUMMY
List contains 6 records
```

```
1 call dblist('ONLY',"Wa")
```

Result:

```
List records of room WORLD(AB) with starting key Wa
-----------------------------------------------------------------------
 city.Wad Medani                                Source 265124
 city.Warsaw                                    Source 1428379
 city.Washington                                Source 3693775
 city.Washington DC                             Dummy  DUMMY
List contains 5 records
```

```
1 call dblist('ANY',"Mu")
```

Result:

```
List records of room WORLD(AB) containing Mu in key
-----------------------------------------------------------------------
 city.Multan                                    Source 1437257
 city.Mumbai                                    Source 19175018
 city.Munich                                    Source 2000981
 city.Murcia                                    Source 516575
 city.Muscat                                    Source 1091400
List contains 5 records
```

```
1 call dblist('CONTAIN',"265")
```

Result:

```
List records of room WORLD(AB) containing 265
-----------------------------------------------------------------------
 city.Ahvaz                                     Source 968265
 city.Bamako                                    Source 1542654
 city.Busan                                     Source 2651469
 city.Oskemen                                   Source 265766
 city.Peshawar                                  Source 1512657
 city.Tanch'on                                  Source 265573
 city.Wad Medani                                Source 265124
List contains 7 records
```

DBKEEP (**[**, **[**, QUALIFIER/ONLY/ANY/CONTAINS**]**, string**]**)

The functionality is similar to that of DBLIST, except that it saves the result in a SARRAY that can be utilised in FMTLIST to be displayed or any other post-process.

Returns: the created SARRAY number

Example:

```
1 s2=dbkeep('Qualifier','Country')
2 buffer.0='ARRAY 's1
3 Call fmtlist
```

DBNKEEP (**[**, **[**, QUALIFIER/ONLY/ANY/CONTAINS**]**, string**]**)

The functionality is the same as that of DBKEEP, but all messages are suppressed.

Returns: the created SARRAY number

DBKKEEP([[QUALIFIER/ONLY/ANY/CONTAINS],string]) lists records

The functionality is similar to that of DBKEEP, except just the KEY is saved as the result in a SARRAY.

Returns: the created SARRAY number

DBHOOD ([qualifier.]key)

Prints the neighbourhood (References) of a Record.

Example:

```
DBHOOD country.U.S.A
Result
  PART-OF           city.ATLANTA
  PART-OF           city.ATLANTA:U.S.A
  PART-OF           city.BOSTON
  PART-OF           city.BOSTON:U.S.A
  PART-OF           city.CHICAGO
  PART-OF           city.CHICAGO:U.S.A
  PART-OF           city.DALLAS
  PART-OF           city.DALLAS:U.S.A
  PART-OF           city.HOUSTON
  PART-OF           city.HOUSTON:U.S.A
  PART-OF           city.LOS ANGELES
  PART-OF           city.LOS ANGELES:U.S.A
  PART-OF           city.MIAMI
  PART-OF           city.MIAMI:U.S.A
  PART-OF           city.NEW YORK
  PART-OF           city.NEW YORK:U.S.A
  PART-OF           city.PHILADELPHIA
  PART-OF           city.PHILADELPHIA:U.S.A
  PART-OF           city.WASHINGTON
  PART-OF           city.WASHINGTON:U.S.A
   |  Refer(s) to country.U.S.A
   V
+-------------------------------------------------+
| country.U.S.A                                   |
+-------------------------------------------------+
   |  Reference(s) from country.U.S.A
   V
  CAPITAL-IS        city.WASHINGTON DC
  CONTAINED-IN      continent.NORTH AMERICA
  MEMBER-OF         union.NORTH AMERICAN FREE TRADE
```

DBREFERENCE ([qualifier.]key,[max-level],[REFS/DETAILS])

Navigates from a given qualifier.key combination to all referred records (forward direction). Max-level defines how many nested levels are allowed (the default is 99). With REFS only the referred entries will be reported. With DETAILS referred entries and the link type are reported.

The qualifier parameter defaults to “any”.

```
1 Call dbreference('country.USA')
```

Results:

```
References of Country.USA
------------------------------------------
Country.USA -- part of -->  North America

Which references constitute city.Munich, 99 levels down, report referred entries only:
Call dbreference('country.USA',99,'REFS')

city.Munich
  PART-OF           country.GERMANY
  CAPITAL-IS        city.BERLIN
  PART-OF           country.GERMANY
  CONTAINED-IN      continent.EUROPE
  MEMBER-OF         union.EUROPEAN UNION
  SIGHT             event.OCTOBERFEST
  BREWERY           any.HACKERBRAEU
  BREWERY           any.HOFBRAEU
  TYPE              any.DARK
  TYPE              any.EXPORT
  BREWERY           any.LOEWENBRAEU
  BREWERY           any.PAULANER
  SIGHT             location.HOFBRAEUHAUS
  BREWERY           any.HOFBRAEU
Elements found 14
```

Which references constitute city.Munich, 99 levels down, report referred and link type entries only:

```
1 Call dbreference('country.USA',99,'LINK')
```

Results:

```
city.Munich
  PART-OF           country.GERMANY
  CAPITAL-IS        city.BERLIN
  PART-OF           country.GERMANY
  CONTAINED-IN      continent.EUROPE
  MEMBER-OF         union.EUROPEAN UNION
  SIGHT             event.OCTOBERFEST
  BREWERY           any.HACKERBRAEU
  BREWERY           any.HOFBRAEU
  TYPE              any.DARK
  TYPE              any.EXPORT
  BREWERY           any.LOEWENBRAEU
  BREWERY           any.PAULANER
  SIGHT             location.HOFBRAEUHAUS
  BREWERY           any.HOFBRAEU
Elements found 14
```

Which references constitute city.Munich, 99 levels down (maximum):

```
1 Call dbreference('city.Munich',99)
```

Results:

```
References of city.Munich
------------------------------------------------------------------------
 1 city.Munich
 1 +- part-of        ->  country.GERMANY
 2 |  country.GERMANY
 2 |  +- capital-is     ->  city.BERLIN
 3 |  |  city.BERLIN
```

```
 3 |  |  +- part-of        ->  country.GERMANY
 4 |  |  |  |- capital-is     ->#  city.BERLIN
 4 |  |  |  |- contained-in   ->  continent.EUROPE
 4 |  |  |  country.GERMANY
 4 |  |  |  +- member-of      ->  union.EUROPEAN UNION
 2 |  |- contained-in   ->#  continent.EUROPE
 2 |  |- member-of      ->#  union.EUROPEAN UNION
 1 |- sight          ->  event.OCTOBERFEST
 2 |  event.OCTOBERFEST
 2 |  +- brewery        ->  any.HACKERBRAEU
 2 |  event.OCTOBERFEST
 2 |  +- brewery        ->  any.HOFBRAEU
 3 |  |  any.HOFBRAEU
2 |  +- brewery        ->  any.LOEWENBRAEU
 2 |  event.OCTOBERFEST
 2 |  +- brewery        ->  any.PAULANER
 1 city.Munich
 1 +- sight          ->  location.HOFBRAEUHAUS
 2 |  location.HOFBRAEUHAUS
 2 |  +- brewery        ->  any.HOFBRAEU
 3 |  |  |- type           ->#  any.DARK
 3 |  |  |- type           ->#  any.EXPORT
    ->#  references have been reported previously
Elements found 14
```

DBUSAGE ([qualifier.]key,[REFS/DETAILS)

Navigates from a given qualifier.key combination to all used records (backward direction). Max-level defines how many nested levels are allowed (the default is 99). With REFS only the used entries will be reported. With DETAILS referred entries and the link type are reported.

```
1 Call dbusage('country.USA')
```

Example:

```
Usages of Country.USA
-----------------------------------------------------
Country.USA -- Economy <--  NAFTA
```

DBROOMS ()

DBROOM displays all previously defined rooms.

Example:

```
List all defined rooms
  HILBERT'S LOBBY        SAA 0
  MOSHE'S FOOD TRUCK     SAF 0
  PETER'S BEACH BAR      SAD 0
  WORLD                  SAB 0
```

Key/Value Database Tailoring

Defining an Information Model (optional)

You may specify a record structure using an information model, which allows the record to be divided into distinct attributes. Every qualification has to have its model defined if one has been set up. The information model is active in the current room. Here is an illustration of the structure of the world database sample:

Sample information model in the world database:

The qualifier country has the following attributes:

```
call dbkvimadd 'country: Acronym Capital Description Visited'
```

The qualifier city has the following attributes:

```
call dbkvimadd 'city: population Description'
```

if all qualifiers (regard them as types) the information model must be built, which can be done by:

```
call dbkvimbuild
```

The information model should be defined at the initialisation process before the database is loaded. The following is the definition of the world database:

```
/* ------------------------------------------------------------
 * Store a simple Key/Value Information Model
 * Example:
 *    country: Acronym Capital Description Visited
 *      |      |       |       |           fourth-attribute
 *      |      |       |       third-attribute
 *      |      |       second-attribute
 *      |      first-attribute
 *      record-type
 * ------------------------------------------------------------
 */
call dbkvimadd 'country: Acronym Capital Description Visited'
call dbkvimadd 'city: population Description'
/* ------------------------------------------------------------
 * Build Information Model and save it in the Control Record
 * ------------------------------------------------------------
 */
call dbkvimbuild
```

Once activated the record can be automatically structured by some commands into its single attributes for example DBPRINT.

Defining additional Key/Value Databases

You can add additional Key/Value Databases as needed. Their definition lengths may vary (e.g. qualifier and key)

Setting up a Key/Value Profile

To personalise the profile REXX script, alter the following REXX variables; any valid dataset or dd name may be selected: define a profile, for example, *DBPROF1*

```
/* ------------------------------------------------------------
 * Private Key/Value Profile
 * ------------------------------------------------------------
 */
  ddprof.DDKEY='PRIVALUE'          /* DD Name of Key/Value DB*/
  ddprof.DDREF='PRIREFS'           /* DD Name of Key/Ref DB  */
  ddprof.DSNKEY='BREXX.PRIVALUE'   /* DSN of Key/Value DB    */
  ddprof.DSNREF='BREXX.PROREFS'    /* DSN of Key/REF DB      */
  ddprof.keylen =12                 /* Plain Key length              */
  ddprof.roomlen=2                  /* Room length                   */
  ddprof.quallen=4                  /* Qualifier/Project/Bucket      */
  ddprof.typelen=4                  /* Type length in Reference DB   */

  ddprof.keyupper=1                 /* upper case translation of the */
                                    /* key. 0: no, 1: yes            */
  ddprof.EnableDummy=1              /* allow links between records   */
                                    /* which do not exist, they will */
                                    /* create DUMMY source records   */
                                    /* 0: no, 1: yes                 */
 return 0
```

- **Keylen**: Key/Value key *length=ddprof.keylen+ddprof.projlen+ddprof.roomlen*. In our example *Keylen=18*.
- **Reflen**: Reference key *length=2*keylen+1+ddprof.typelen*. In our example *Reflen=41*.

Cluster Definition

1. Create a copy of the member $CREKEYV of the BREXX installation dataset. Change the cluster definition accordingly.
2. Change the Cluster sizes so that it fits your need.
3. Submit the JCL

For our Example:

```
//* ---------------------------------------------------------------
//* CREATE A NULL RECORD FOR THE KEY/VALUE VSAM FILES
//* ---------------------------------------------------------------
//EXEC     EXEC PGM=BREXX,PARM='RXRUN',REGION=6000K
//STEPLIB  DD   DSN=SYS2.LINKLIB,DISP=SHR
//STDIN    DD   DUMMY
//STDOUT   DD   SYSOUT=*,DCB=(RECFM=FB,LRECL=140,BLKSIZE=5600)
//STDERR   DD   SYSOUT=*,DCB=(RECFM=FB,LRECL=140,BLKSIZE=5600)
//NULLREC  DD DSN=&&NULLREC,DISP=(,PASS),UNIT=VIO,SPACE=(TRK,(1,1)),
//         DCB=(RECFM=FB,LRECL=255,BLKSIZE=255)
//RXRUN    DD   *
   NULLR.1=COPIES('9',255); NULLR.0=1
  "EXECIO * DISKW NULLREC (STEM NULLR. FINIS"
/*
//* ---------------------------------------------------------------
//* DELETE/DEFINE THE PROMOTE VSAM CHANGE (CCID) LIST AND PRIME IT
//* WITH THE CONTROL RECORD
//* ---------------------------------------------------------------
//KEYVALUE EXEC PGM=IDCAMS
//NULLREC  DD DSN=&&NULLREC,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  DELETE BREXX.PRIVALUE CLUSTER
  SET MAXCC = 0
  DEFINE CLUSTER                          -
           (NAME(BREXX.PRIVALUE)          -
            INDEXED                       -
            KEYS(18 0)                    -
            RECORDSIZE(64 8192)           -
            SHAREOPTIONS(2,3)             -
            CYLINDERS(600 50)             -
            VOLUMES(PEJ001)               -
            UNIQUE                        -
            SPEED)                        -
         DATA                             -
            (NAME(BREXX.PRIVALUE.DATA))   -
         INDEX                            -
            (NAME(BREXX.PRIVALUE.INDEX))
  IF LASTCC = 0 THEN -
     REPRO INFILE(NULLREC) ODS(BREXX.PRIVALUE)
/*                                                /*
//KEYREF   EXEC PGM=IDCAMS
//NULLREC  DD DSN=&&NULLREC,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  DELETE BREXX.PRIREFS CLUSTER
  SET MAXCC = 0
```

```
48   DEFINE CLUSTER                          -
49             (NAME(BREXX.PRIREFS)          -
50              INDEXED                      -
51              KEYS(41 0)                   -
52              RECORDSIZE(64 256)           -
53              SHAREOPTIONS(2,3)            -
54              CYLINDERS(50 25)             -
55              VOLUMES(PEJ001)              -
56              UNIQUE                       -
57              SPEED)                       -
58           DATA                            -
59              (NAME(BREXX.PRIREFS.DATA))   -
60           INDEX                           -
61              (NAME(BREXX.PRIREFS.INDEX))
62   IF LASTCC = 0 THEN -
63      REPRO INFILE(NULLREC) ODS(BREXX.PRIREFS)
64 /*
65 //
```

Usage of the new Cluster Definition

Once the definition is constructed, you may use it by specifying the profile name in the DBOPEN call, for example, DBPROF1:

```
CALL DBOPEN('DBPROF1')
```

There can only be one database open at the same time! If another database was opened during the same run, it must be closed using a *DBCLOSE*.

Formatted screens

The following document is a brief description of the new Formatted Screen (FSS) feature. It allows to set up simple screen definitions within a BREXX script.

For detail take a closer look at the FSS samples in the delivered Installation library *BREXX.INSTALL.SAMPLES*

Delivered Samples

The relevant FSS samples are prefixed with the #-sign:

Member	Description
#TSOAPPL	Shows in a detailed usage of all FSS functions how to set up a menu and “paint” a TK4 like design
#BROWSE	A pre-packed FSS application to display data in a List Buffer instead of using SAYs
#FSS1COL	A pre-packed FSS application to generate input requests (in one column)
#FSS2COL	A pre-packed FSS application to generate input requests (distributes in two columns)
#FSS3COL	A pre-packed FSS application to generate input requests (distributes in three columns)
#FSS4COL	A pre-packed FSS application to generate input requests (distributes in four columns)
#FSS4CLX	A pre-packed FSS application to generate input requests (distributes in four columns)

FSS Limitation

The FSS screen limitation has been dropped. Now large screen widths and heights are supported.

FSS supports just one FSS Screen definition at a time. If you need to display more than one FSS Screen in your REXX application, you must close the first and set up and display the next FSS definition. Using this method, you can easily switch between different FSS Screens. It is a good idea to separate the FSS definitions in different sub-procedures; this allows their display by calling it.

FSS Function Overview

To use FSS functions in BREXX, you must import the FSS API library from BREXX.V2R5M3.RXLIB, address and initialise it by a call to FSSINIT, be aware that FSS is a host command application that requires an ADRESS FSS command, it is sufficient to use it once at the beginning. From this time on all host, commands are directed to FSS. If it happens to be and you have to switch to another host API (e.g. ADDRESS TSO or ADDRESS SYSTEM), you can do so, but you must make sure to switch back to the FSS API by re-issuing an ADDRESS FSS command:

```
1 /* IMPORT THE API LIBRARY */
2 CALL IMPORT FSSAPI
3 /* ADDRESS THE FSS SUBSYSTEM */
4 ADDRESS FSS
5 /* SWITCH TO FULL-SCREEN MODE */
6 CALL FSSINIT
```

FSSINIT Inits the FSS subsystem

Initialise the FSS environment; this must be performed before any other FSS call: *CALL FSSINIT*

Principles of Defining Formatted Screens

You can define your formatted screen by using a series of FSSTEXT and FSSFIELD and/or some wrapped FSS functions as FSSMESSAGE, FSSCOMMAND, etc. in your REXX script. Essential parameters are, in all cases, the ROW and COLUMN positions. Be aware that consistency validations are very basic and not bulletproof at all. It is, for example, possible to accidentally re-use occupied ranges, which may lead to unwanted behaviour or results. Performing just necessary validations increases the performance of the screen handling. It is, therefore, essential that you carefully design your Formatted Screens.

FSSTEXT

`CALL FSSTEXT 'text',row,column,[text-length],attributes`
Display a text field

Parameters:

- **text** – text to be displayed in the screen
- **row** – row where text should be placed
- **column** – column where text should be placed.
- **text-length** – length occupied by the text, this is an optional parameter; it defaults to the text length.
- **attributes** – screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section

FSSFIELD

`CALL FSSFIELD 'field',row,column,[length],attributes[,init-value]`
Display an input field and associate it with a BREXX Variable

Parameters:

- **field** – field-name of an input area to be displayed on the screen
- **row** – row where text should be placed
- **column** – column where text should be placed.
- **length** – length occupied by the text, this is an optional parameter; it defaults to the text length.
- **attributes** – screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section
- **init-value** – what should be displayed as content of the input field. It defaults to blank.

Note

Important Notice on the Column Position
Each text or field definition starts with the defined attribute byte, which itself is invisible but tells how the text or field appears on the screen. Therefore the original text or field-definition start at column+1.

Note

Important Notice on Screen Definitions
Be aware that all definitions provided by FSSTEXT and FSSFIELD are stacked internally. They do not create a formatted screen on the fly.

Attribute Definition

The attribute definitions trigger the behaviour or colours of the Formatted Screen text or input elements.

Attribute	Description
#PROT	Definition is protected (default for fsstext)
#NUM	input field must be numeric
#HI	text is displayed high-lighted
#NON	text/field-input is invisible
#BLINK	text/field blinks
#REVERSE	background is set with defined colour text appears white
#USCORE	Underscore field

Colors:

Attribute	Description
#BLUE	text or input field is of blue colour
#RED	text or input field is of red colour
#PINK	text or input field is of pink colour
#GREEN	text or input field is of green colour
#TURQ	text or input field is of turquoise colour
#YELLOW	text or input field is of yellow colour
#WHITE	text or input field is of white colour

You can combine several attribute bytes by adding them. e.g. *#PROT+#BLUE* combining several colours is not allowed and may lead to unexpected errors.

FSSTITLE

Displays a centred Title in Screen line 1

`CALL FSSTITLE title-text[,attributes]`
Besides the title definition the right hand 25 bytes may contain a short message in case of errors, it overwrites the title part in error situations and automatically resets it, if the enter key is used.
The error field is named ZERRSM and maybe set also by your program.

FSSOPTION

`CALL FSSOPTION [row[,option-length[,attribute1,[attribute2]]]`
Creates an OPTIONs line, typically used in a menu to select a menu option:

```
OPTION ===> _____________________________________________________
```

Parameters:

- **row** – defaults to 2
- **option-length** – defines the line length to proved the option input, default is length of the remaining line
- **attribute1** – Attribute of “OPTION”, default is #PROT+#WHITE
- **attribute2** – Attribute of the option line,default is #HI+#RED+#USCORE

FSSCOMMAND

`CALL FSSCOMMAND [row[,option-length[,attribute1,[attribute2]]]`
Creates an input line for entering menu options or commands, it appears with the “COMMAND ===>” prefix and is typically located in row 2.:

```
COMMAND ===> _____________________________________________________
```

Parameters:

- **row** – defaults to 2
- **option-length** – defines the line length to provide the command input, default is length of the remaining line
- **attribute1** – Attribute of “COMMAND”, default is #PROT+#WHITE
- **attribute2** – Attribute of the command line,default is #HI+#RED+#USCORE

FSSTOPLINE

`CALL FSSTOPLINE prefix,[row[,option-length[,attribute1,[attribute2]]]`
Create an Option/Command Line. FSSTOPLINE is a variation of FSSCOMMAND which allows the free definition of the input line prefix. It is typically located in row 2.:

```
MY-OPTION ===> _____________________________________________________
```

Prefix:	String which should appear in front of the input line. In the example above it is “MY-OPTION”
Row:	defaults to 2
Option-length:	defines the line length to provide the command input; default is the length of the remaining line
Attribute1:	Attribute of “COMMAND”, default is #PROT+#WHITE
Attribute2:	Attribute of the command line, default is #HI+#RED+#USCORE

FSSMESSAGE

CALL FSSMESSAGE [row[,attribute]]

Creates a message line to display messages. The message line occupies a full-screen line.

param row: defaults to 3

param attribute: attribute of message line, default is #PROT+#HI+#RED

A call to FSSZERRLM sets the Message

FSSZERRSM

Set Error/Warning/Info Short Message. The message is set in Field ZERRSM. ZERRSM is automatically created by using an FSSTITLE definition; otherwise, it must be defined explicitly. If implicitly used with the FSSTITLE definitions, it starts on the right-hand side after the end of the message; its length is dependant on the length of the title.

```
CALL FSSZERRSM 'message'
```

FSSZERRLM

Set Error/Warning/Info Long Message. The message is set in Field ZERRLM, which has been defined on the screen by a CALL FSSMESSAGE.

```
CALL FSSZERRLM 'message'
```

FSSFSET

Set Field Content

```
CALL FSSFSET 'field',content
```

Make sure the field-name is enclosed in quotes; otherwise, there is a chance of unwanted substitution by its value!

FSSFGET

Get current Field Content .. function:: Value=FSSFGET(‘field’)

Make sure the field-name is enclosed in quotes; otherwise, there is a chance of unwanted substitution by its value!

FSSFGETALL

Get Contents of all Fields

```
Number=FSSFGETALL()
```

All field contents of the screen are fetched and stored in the associated BREXX fields defined by FSSFIELD(…)

FSSCURSOR

Set Cursor to a Field .. function:: CALL FSSCURSOR ‘field’

FSSCOLOUR

Change Colour of a Field

```
CALL FSSCOLOUR 'field',colour-attribute alternatively
```

```
CALL FSSCOLOR 'field' ,colour-attribute
```

FSSKEY

Return Key entered. When the user presses an action-key on a screen the used key value to return control can be accessed by FSSKEY. The optional parameter CHAR returns it in a translated readable form if not set the value returned is the decimal value assigned to the action key.

```
key=FSSKEY([CHAR])
```

By FSS supported keys:

REXX Variable	Numeric Value	Translated Value
#ENTER	125	ENTER
#PFK01	241	PF01
#PFK02	242	PF02
#PFK03	243	PF03
#PFK04	244	PF03
#PFK05	245	PF05
#PFK06	246	PF06
#PFK07	247	PF07
#PFK08	248	PF08
#PFK09	249	PF09
#PFK10	122	PF10
#PFK11	123	PF11
#PFK12	124	PF12
#PFK13	193	PF13
#PFK14	194	PF14
#PFK15	195	PF15
#PFK16	196	PF16
#PFK17	197	PF17
#PFK18	198	PF18
#PFK19	199	PF19
#PFK20	200	PF20
#PFK21	201	PF21
#PFK22	74	PF22
#PFK23	75	PF23
#PFK24	76	PF24
#CLEAR	109	CLEAR
#RESHOW	110	RESHOW

FSSDISPLAY

Displays or Re-Displays the active screen.

```
CALL FSSDISPLAY
CALL FSSREFRESH
```

Get Screen Dimensions

```
width=FSSWidth()
```

Returns: number of available columns defined by Emulation

```
height=FSSHeight()
```

Returns: number of available rows defined by Emulation

Close FSS Environment

Once the Screen Handling is finished it is recommended to terminate the FSS environment with one of:

`CALL FSSTERM`

`CALL FSSTERMINATE`

`CALL FSSCLOSE`

Creating a Dialog Manager

To handle user's action-keys, you can set up a simple Dialog Manager, as shown in this example:

```
 1 /* ---------------------------------------------------------------------
 2 * Display screen in primitive Dialog Manager and handle User's Input
 3 * ---------------------------------------------------------------------
 4 */
 5 do forever
 6     fsreturn=fssDisplay()           /* Display Screen */
 7     if fsreturn='PFK03' then leave  /* QUIT requested */
 8     if fsreturn='PFK04' then leave  /* CANCEL requested */
 9     if fsreturn='PFK15' then leave  /* QUIT requested */
10     if fsreturn='PFK16' then leave  /* CANCEL requested */
11     if fsreturn<>'ENTER' then iterate
12     call fSSgetD()                  /* Read Input Data */
13     /* Add input checking if needed */
14 end
15 call fssclose /* Terminate Screen Environment */
```

Simple Screen Applications

There is a simple way to create formatted screens using preformatted rexx scripts, and this allows an easy screen setup without coding all the screen definitions manually.

Screen with Attributes in one Column

From *BREXX.V2R5M3.SAMPLES(#FSS1COL)*

```
 1  /*          + ------------------- Screen with 1 column
 2   *          !
 3   *          !    + -------------- Title line of screen
 4   *          !    !     */
 5 frc=FMTCOLUM(1,'One Columned Formatted Screen',
 6      ,'1. First Name  ===>',
 7      ,'2. Family Name ===>',
 8      ,'3. UserId      ===>',
 9      ,'4. Department  ===>',
10      )
11 do i=1 to _screen.input.0
12    say "User's Input "i'. Input Field: '_screen.input.i
13 end
14 return
```

The above definition creates and displays this screen:

```
----------------------- One Columned Formatted Screen --------------------

 1. First Name  ===> ____________________________________________________
 2. Family Name ===> ____________________________________________________
 3. UserId      ===> ____________________________________________________
 4. Department  ===> ____________________________________________________
```

After entering input and pressing enter, you receive the provided input:

```
----------------------- One Columned Formatted Screen --------------------

 1. First Name  ===> Fred_____________________________________________
 1. Family Name ===> Flintstone_______________________________________
 2. UserId      ===> FL2311___________________________________________
 3. Department  ===> Quarry___________________________________________
```

The provided input is stored in *SCREEN.INPUT.xx* an can be used or printed as in this REXX script:

```
User's Input 1. Input Field: Fred_____________________________________
User's Input 2. Input Field: Flintstone_______________________________
User's Input 3. Input Field: FL2311___________________________________
User's Input 4. Input Field: Quarry___________________________________
```

Screen with Attributes in two Columns

From *BREXX.V2R5M3.SAMPLES(#FSS2COL)*

```
 1  /*           + ------------------- Screen with 2 columns
 2   *          !
 3   *          !    + -------------- Title line of screen
 4   *          !    !     */
 5 frc=FMTCOLUM(2,'Two Columned Formatted Screen',
 6      ,'1. First Name  ===>',
 7      ,'2. Family Name ===>',
 8      ,'3. UserId      ===>',
 9      ,'4. Department  ===>',
10      )
11 do i=1 to _screen.input.0
12    say "User's Input "i'. Input Field: '_screen.input.i
13 end
14 return
```

you get the attributes in two columns:

```
----------------------- Two Columned Formatted Screen --------------------

1. First Name  ===> ____________________ 2. Family Name ===> _________________
3. UserId      ===> ____________________ 4. Department  ===> _________________
```

Entered input is provided in the same way as in the one column screen example.

Screen with Attributes in three Columns

```
----------------------- Three Columned Formatted Screen ------------------

1. First Name  ===> _____ 2. Family Name ===> _____ 3. UserId      ===> ___
4. Department  ===> _____
```

Just change the number of columns to 3: *frc=FMTCOLUM(3,'Three Columned Formatted Screen',*

Screen with Attributes in four Columns

Last option is to place the attributes in four columns: *frc=FMTCOLUM(4,'Three Columned Formatted Screen',*

Screen special Attributes

You can tailor the appearance of formatted column screens, by setting **_screen.xxxx** variables:

Presetting Screen input fields

Use _SCREEN.INIT.n='input-value-as-default', n is the reference to the field in the FMTCOLUMN definition. 1 is first, 2 second, etc.

Example:

```
_SCREEN.INIT.1='FRED'
_SCREEN.INIT.3='Flintstone'
_SCREEN.INIT.4='FL2311'
_SCREEN.INIT.5='Quarry'
```

Calling the formatted screen, you get a pre-set Screen:

```
----------------------- One Columned Formatted Screen --------------------

1. First Name  ===> Fred_____________________________________________
1. Family Name ===> Flintstone_______________________________________
2. UserId      ===> FL2311___________________________________________
3. Department  ===> Quarry___________________________________________
```

Input field appearance

If not changed, the input fields appear with an underscore in the available length. You can change it by setting _screen.preset. If you set _screen.preset='+' (one character) the input field filled by the character you defined. If you use more than one character *_screen.preset='_ '* only the given string is displayed.

Input field length

The field length is, by default, delimited by the following field definition in the row, or by the end of the line.

If you want to limit it to a certain length by: *_SCREEN.LENGTH.n=field-length* n is the field number you want to set. It is sufficient to set just the field length you want to limit.

Input Field CallBack Function

Normally, if you press enter, the screen control is giving back to your rexx, and the variable content is returned. If you prefer to check the entered input while your formatted screen is still active, for example, to validate user's input, you can define a callback function:

```
_screen.ActionKey='internal-subprocedure'
```

The internal sub-procedure must be coded without a PROCEDURE statement; else you cannot use the screen input variables

```
_screen.ActionKey='checkInput'
frc=FMTCOLUM(2,'Two Columned Formatted Screen',
...
return
/* ---------------------------------------------------------------------
* Call Back Routine from FMTCOLUMN to check provided Input
* ---------------------------------------------------------------------
*/
checkInput:
  if _screen.input.1 = '' then do
    call FSSzerrsm 'Field 1 ist mandatory'
    call FSSzerrlm 'Please enter valid content in Field 1'
    return 1
end
if _screen.input.2 = '' then do
    call FSSzerrsm 'Field 2 is mandatory'
    call FSSzerrlm 'Please enter valid content in Field 2'
    return 1
end
...
```

In case of an error, your call back function can use the FSSzerrsm function, which displays a short message in the formatted screen's title line and/or the FSSzerrlm function to display a long message. The error message is displayed in the last line of Formatted Screen. Your callback sub-procedure signals with its return code how to proceed:

Return	Description
return 0	everything ok, leave screen an pass control back to calling rexx
return 128	something is wrong, re-display the screen
return 256	something is wrong, leave the screen
return n:	field n contains wrong input, re-display screen n >0 and n<128 represents the field number in error

FSSMENU Supporting Menu Screens

To ease the creation of menu screens, you can use the FSSMENU definition. It creates the screen layout as well as the dialogue handling part.

Defining a Menu Screen

`CALL FSSMENU 'option','note','description','action',[startRow],[startCol]`

Parameters:

- **option** – option code which leads to performing the associated action. The option can be a numeric or alphanumeric string and its length must not exceed 2.
- **note** – short description of the action to perform
- **description** – long description of the action to perform
- **action** – action is performed is associated option is selected TSO prefixes an action for a TSO function call or with CALL if a REXX procedure should be called.
- **startRow** – row in which the first menu should be placed, default is 12. This parameter is only validated for the first FSSMENU definition and automatically used for each subsequent call. To achieve a row centred menu appearance, you can use the following rexx coding before the first FSSMENU definition: menumax=5 /* number of Menu entries startRow=(FSSHeight()%2)-(menuMax%2+1)-3 and pass startRow as a parameter in the FSSMENU definition
- **startCol** – column in which the menu should be placed, default is 6. This parameter is only validated for the first FSSMENU definition and automatically used for each subsequent call. To achieve a column centred menu appearance, you can use the following rexx coding before the first FSSMENU definition: startcol=(FSSWidth()%2)-30 and pass startCol as a parameter in the FSSMENU definition

The FSS menu definitions can be included within a typical FSS Screen definition to add additional fields or text parts to the formatted screen. These parts can be dynamically updated if you specify a callback procedure in the FSSMENU Display call.
The FSSMENU definition relies on the existence of the following fields (FSSMENU does not automatically generate them); they must be defined separately, either implicitly or explicitly:

- ZCMD is defined by FSSTOPLINE or FSSCOMMAND
- ZERRSM is defined by FSSTITLE

Example defined in a REXX script:

```
1 ...
2 CALL FSSMENU 1,"RFE",     'SPF like" productivity tool',
3                ,"TSO CALL 'SYS2.CMDLIB(RFE)"
4 CALL FSSMENU 2,"RPF",     'SPF like" productivity tool','TSO RPF'
5 CALL FSSMENU 3,"IM",      'IMON/370 system monitor','TSO IM'
6 CALL FSSMENU 4,"QUEUE",   'spool browser','TSO Q'
7 CALL FSSMENU 5,"HELP",    'general TSO help','TSO HELP'
8 CALL FSSMENU 6,"UTILS",
9      ,'information on utilities and commands available','TSO HELP UTILS'
```

```
10 CALL FSSMENU 7,"TERMTEST" ,'verify 3270 terminal capabilities',
11      ,'TSO TERMTEST'
12 ...
```

FSSMENU Displaying a Menu Screen

To display the menu and handle the selected actions, FSSMENU must be called with the $DISPLAY parameter:

`returnkey=FSSMENU('$DISPLAY',[callback-procedure],[actionkey-procedure])`

Parameters:

- **returnkey** – key used to end the dialogue handling, it is either PF03, PF04, PF15, or PF16
- **$DISPLAY** – Display the menu defined before
- **callback-procedure** – optional own callback procedure (internal or external) to update FSS variables or other variables. This procedure is called just before the menu is displayed and re- displayed. Therefore the variables which are defined for the menu screen and modified in the procedure are displayed with their new content. The callback procedure needs the scope of the FSSMENU variables; therefore, it must not be defined with a PROCEDURE statement. Just define the callback name with a label.
- **actionkey-procedure** – optional own action key procedure (internal or external) to check user's input in the command line This procedure is called when the user pressed the enter key, and the command line contains input. This input could be a simple menu option or maybe a command, which you like to process. It is also called if a PF-Key was used to request an action. PF03, PF04, PF15 and PF16 are not passed to the procedure as they trigger the standard return action. The action key procedure is called with the parameters action-key and command-line. To receive them in your procedure use: parse arg action, command Name of the above variables is of course freely selectable. To return to the calling menu, it is essential to provide a return code; this allows the menu processing to decide on the next steps. Return codes: 0 input has been handled by the exit, re-display Menu 4 input has not been handled, continue with internal checks 8 exit Menu immediately

Example: Simple Display without any exits

```
1 rckey=FSSMENU('$DISPLAY')
2 say 'End Key 'rckey
3 ...
```

Example: Before Display update some variables via a callback procedure

```
 1 rckey=FSSMENU('$DISPLAY','UPDVAR')
 2 say 'End Key 'rckey
 3 ...
 4 /* ---------------------------------------------------------------------
 5 * Update some Variables before displaying the Menu
 6 * ---------------------------------------------------------------------
 7 */
 8 Updvar:
 9 MDate=date() /* assuming MDATE/MTIME are defined in the MENU */
10 MTime=time('L')
11 Return
12 ...
```

Example: Before Display update some variables via a callback procedure, and check command line input via an enter-exit

```
 1 rckey=FSSMENU('$DISPLAY','UPDVAR','CHECKKEY')
 2 say 'End Key 'rckey
 3 ...
 4 ...
 5 /* ---------------------------------------------------------------------
 6 * Update some Variables before displaying the Menu
```

```
 7 * ---------------------------------------------------------------------
 8 */
 9 Updvar:
10 MDate=date() /* assuming MDATE/MTIME are defined in the MENU */
11 MTime=time('L')
12 Return
13 /* ---------------------------------------------------------------------
14 * Check user's Input in command Line
15 * Return code handling:
16 *  0 input has been handled by exit, re-display Menu
17 *  4 input has not been handled, continue with internal checks
18 *  8 exit Menu immediately
19 * ---------------------------------------------------------------------
20 */
21 CheckKey:
22 Parse arg actionkey,usercommmand
23 If length(usercommand)>2 then do
24   Say usercommand' is not an Option'
25   Return 0 /* continue, command already checked */
26 End
27 Return 4
28 /* maybe an Option, continue to option check */
```

FMTMENU Fully Defined Menu Screens

Using FSSMENU, you can define the menu lines and generate the menu handling, but it must be incorporated in a normal REXX script containing the other parts of the screen definition and handling. **FMTMENU** allows you the definition of a menu screen in one step, but there are additional screen definitions in the menu possible.

Definition of the Menu

`CALL FMTMENU 'option','note','description','rexx-script'`

Parameters:

- **option** – option code which leads to performing the associated action. The option can be a numeric or alphanumeric string.
- **note** – the short description of the action to perform
- **description** – long description of the action to perform
- **rexx-script** – REXX script which performs the action when the option is selected. Note the difference, to FSSMENU, here it must be a REXX script, but it may also contain calls to TSO, etc.

An FMTMENU always contains a title line (first row) an option line (second row) a message line (last row -1) and a footer line (last row).

Displaying the FMTMENU Screen

To display the menu and handle the selected actions, FMTMENU must be called with the $DISPLAY parameter:

`returnkey=FMTMENU('$DISPLAY','menu-title')`

Parameters:

- **returnkey** – key which was pressed to end the dialogue handling, it is either PF03, PF04, PF15, or PF16
- **$DISPLAY** – Display the menu defined before
- **menu-title** – defining the menu title

Menu Tailoring

There are some settings, which allow you to tailor the menu layout. The usage of the stem *_screen* defines all settings .xxx. These settings are supported in FSSMENU as well as in FMTMENU.

Setting	Description
_screen.MenuRow	starting row of first Menu entry (default is 4)
_screen.MenuCol	Column of Option parameter (default is 6)
_screen.Menucol2	Column of note parameter (default is _screen.MenuCol+3)
_screen.Menucol3	Column of note parameter (default is _screen.MenuCol+14)

Note for FSSMENU:

there are separate parameters startrow and startcol in the menu definition

`CALL FSSMENU 'option','note','description','action',[startRow],[startCol]`

If they are defined, they take precedence over the screen.MenuRow and screen.MenuCol definition.

_screen.MenuFooter	defines the contents of a footer line (placed on the last row)

Setting just for FSSMENU (in FMTMENU they are managed automatically)

Setting	Description
_screen.MenuOption 1	adds an Option line, else it must be defined manually
_screen. MenuMessage 1	adds a message line (last row-1)
_screen. Menutitle' 1	adds a title line

Formatted List Output

The usage of SAY statements displays the standard output of a REXX script. The disadvantage you can not scroll in it. Alternatively, you can write it in a sequential file and view it after the script has ended. By using the FMTLIST command and passing a result buffer in a stem variable, you can browse in the output while your REXX script is still running.

Example REXX reads entire RXDATE Member and displays it:

```
1 /* REXX */
2 ADDRESS TSO
3 "ALLOC FILE(INDD) DSN('BREXX.V2R5M3.RXLIB(RXDATE)')"
4 "EXECIO * DISKR INDD (STEM BUFFER."
5 "FREE FILE(INDD)"
6 CALL FMTLIST
7 RETURN
```

Results:

```
CMD ==>                                                ROWS 00001/00199 COL 001 B01
***** ***************************** Top of Data ******************************
00001 /* REXX */
00002 /* ------------------------------------------------------------------------
00003  *  should not be used anymore, all date functions are integrated in
00004  *    DATE(<output-format>,<date>,<input-format>)
00005  * ------------------------------------------------------------------------
00006  *  RXDATE Transforms Dates in various types
00007  *  ............................. Created by PeterJ on 21. November 2018
00008  *  RXDATE(<output-format>,<date>,<input-format>)
00009  *  date is formatted as defined in input-format
00010  *    it defaults to today's date
00011  *  Input Format represents the input date format
00012  *    it defaults to 'EUROPEAN'
00013  *     Base      is days since 01.01.0001
```

```
00014  *     JDN       is days since 24. November 4714 BC
00015  *     UNIX      is days since 1. January 1970
00016  *     Julian    is yyyyddd    e.g. 2018257
00017  *     European  is dd/mm/yyyy e.g. 11/11/2018
00018  *     German    is dd.mm.yyyy e.g. 20.09.2018
00019  *     USA       is mm/dd/yyyy e.g. 12.31.2018
00020  *     STANDARD  is yyyymmdd   e.g. 20181219
...
```

Using the PF7 and PF8 you scroll upward and forward, PF10 and PF11 scroll left and right. M in the CMD line and PF7 moves buffer to the top, M and PF8 to the bottom. A number and PF7 or PF8 moves the buffer the specified lines up or down.

FMTLIST Prerequisites

FMTLIST always displays the content of the stem variable BUFFER. The buffer must have the general structure:

BUFFER.0	contains the number of entries in BUFFER
BUFFER.1	contains the first line
BUFFER.2	second line
…	
BUFFER.n	last line

As the name is fixed, it does not need to be passed to FMTLIST.

Alternatively, you can also display a String Array. Then you need to specify, in BUFFER.0:

```
BUFFER.0=″SARRAY ″array-number
```

FMTLIST calling Syntax

`FMTLIST` ()

Parameters:

- **length-line-area** – length of displayed line-area, default is 5
- **line-area-character** – character which should be displayed in the line area, default is none, then the line area contains the line number
- **header-1** – this is an optional header line which is shown as first-line the displayed buffer
- **header-2** – optional second header, only if header-1 is also defined
- **applicationID** – If you specify an application ID, the FMTLIST screen supports line commands. The Line commands must be defined and coded in the calling REXX script as a callback label: applicationID_linecommand.

If you use PF7/PF8 to scroll up and down, the two header lines are always displayed as the buffer top lines.

FMTLIST supported PF Keys and Scrolling commands

PF3/PF4	exit FMTLIST screen
PF7	scroll one page up
PF8	scroll one page down
PF10	shift buffer 50 columns left
PF11	shift buffer 50 columns right
PF12	Display last command

If you use a combination of a number in the command line and PF7 or PF8, the buffer scrolls the number of lines up or down.

Command-line functions:

TOP	displays the first line of the buffer
M and PF7	displays the first line of the buffer
BOTTOM	displays the last line of the buffer
BOT	displays the last line of the buffer
M and PF8	displays the last line of the buffer

FMTLIST Customising Options

By setting _SCREEN.xxxx, you can manipulate the appearance of FMTLIST in various ways:

Variable Name	Default	Allowed Values	Note
_screen.cmdchar	blank		Command Line character building the command line. Default is blank and creates an empty command line which is displayed with the 3270 attribute #USCORE
_screen.color.Cmd	#red	Attribute Definitions	Colour of Command Line
_screen.color.Stats	#white	Attribute Definitions	Colour of Statistics (line and buffer numbering)

_screen.color.Top1	#red	Attribute Definitions	Colour of line area first line
_screen.color.Top2	#blue	Attribute Definitions	Colour of line conten first line (Top of Data)
_screen.color.Bot1	#red	Attribute Definitions	Colour of line area last line
_screen.color.Bot2	#blue	Attribute Definitions	Colour of line content last line (End of Data)
_screen.color.List1	#white	Attribute Definitions	Colour of line area (content part)
_screen.color.List2	#green	Attribute Definitions	Colour of line content part

_screen.footer	undefined	Content of footer (PF 1 …)	Fixed Footer Line (at screen height)
_screen.color.footer	#white	Attribute Definitions	Colour of line content part
_screen.Message	undefined	1for defining message	Fixed Message Line (screen height-1)
_screen.TopRow	1	1 up to Screen height-3	Begin row of fmtlist, if it is 2 or more there are empty lines above FMTLIST
_screen.TopRow.proc	Undefined		Is a call-back proc name in the rexx calling FMTLIST. There you can define the line above the FMTLIST screen. They can be set with FSSfield or FSSText commands. The number of added rows must not exceed _screen.TopRow-1
_screen.BotLines	Lines reserve at bottom of FMTLIST	1 up to Screen height-3	As screen height is dynamic depending on the 3270 definitions.

_screen.BotLines.proc	Undefined		Is a call-back proc name in the rexx calling FMTLIST. There you can define the lines at the end of the FMTLIST screen. They can be set with FSSfield or FSSText commands. The first line number which can be set is passed as arg(1) parameter. For consistency reasons of call back parameters, it is enclosed in quotes. This means you must strip them off: *first=strip(translate(arg(1),'',"'"))*

FMTLIST calling other REXX scripts from the command line

If you want to play another REXX script from within the FMTLIST buffer you can do so, by entering: *rexx-script-name* in the command command line.

Simple REXX scripts

A simple Rexx script does not contain any call to an FSS Screen. A sequence of say statements may provide the result, or you can place it in a buffer.x stem. If you do so, the result displayed in the current FMTLIST buffer. Which means the existing content is overwritten.

```
1 Buffer.1='first line'
2 Buffer.2='second line'
3 Buffer.0=2
```

If you want to keep the contents of the current buffer, use the prefix command *LOOKASIDE rexx-script-name*, and a new stacked buffer is created residing on top of the previous buffer. The previous buffer can be re-activated by pressing the PF3 key; it destroys the current buffer and returns to the last buffer.

If the called rexx-script contains an FMTLIST, FSSMENU, or FMTMENU itself a new buffer is created automatically.

Formatted List Line and Primary Commands

The FMTLIST Buffer supports Line Commands if it is called with an applicationID. The line command is coded within the calling procedure (performing the FMTLIST) as a callback label, to keep the scope of the variables there must not be a PROCEDURE statement used. The callback label must be coded as: applicationID_linecommand. In the following example there is a line command S, U and D defined :

```
 1 /* REXX */
 2 ADDRESS TSO
 3 "ALLOC FILE(INDD) DSN('BREXX.V2R5M3.RXLIB(RXDATE)')"
 4 "EXECIO * DISKR INDD (STEM Buffer."
 5 "FREE FILE(INDD)"
 6 call fmtlist ,,,,MYLIST /* MYLIST is application ID */
 7 return
 8
 9 /* ---------------------------------------------------------------------
10 * Line commands are organised as "call-back' labels to the calling REXX
11 * Format is REXX name_linecmd
12 * ---------------------------------------------------------------------
13 */
14
15 mylist_s: /* line command S, just output selected line */
16   say Arg(1)
17   return 0 /* tell FMTLIST to proceed normally */
18
19 mylist_u: /* line command U, allow editing line */
20   newLine=lineedit(,arg(1))
21   return 4 /* tell FMTLIST, you changed line */
22
23 mylist_e: /* line command E, automatically change line */
24   newLine='new Line set'
```

```
25   zerrsm='update'
26   zerrlm='Line has been updated'
27   return 4 /* tell FMTLIST, line is changed line */
28
29 mylist_d: /* Delete Line */
30   return 5 /* tell FMTLIST to delete selected line */
```

RC Code actions

RC=0	means the line command was processed
RC=4	means the line command was processed; if the REXX variable NEWLINE contains a value, the selected line will be overwritten by this value.
RC=5	delete this line
RC=6	a completely new buffer.n stem has been provided and should be displayed immediately. The old buffer content will be removed. If you set a ZERRSM or ZERRLM message the message will be kept and displayed.
RC=7	a new buffer.n stem has been provided and should be displayed in a new FMTLIST buffer, which is stacked on top of the previous one. Once you return with PF3 you will see the old buffer content. If you set a ZERRSM or ZERRLM message the message will be kept and displayed.
RC=8	invalid line command

Additionally, you can change the colour of the line in the buffer; you have to set:

- SETCOLOR1 sets the colour of the selected line of the line area, e.g. *setcolor1=#green*
- SETCOLOR2 sets the colour of the selected buffer content line, e.g. *setcolor2=#red*

If none or just one of the colours have been set, the other field colour remains unchanged

Formatted List Special Call-Back labels

FMTLIST supports certain call-back labels (defined in the calling REXX) if FMTLIST is called with an applicationID.

Formatted List Special labels

FMTLIST also supports calling generic procedures. They must be explicitly activated to be called. The location is of your choice, they can be defined in the calling REXX or as independent REXX.

TOPROW Procedure

Allows you to embed an FMTLIST screen into a frame of your own. It must be activated by defining the beginning position of the FMTLIST screen, and the label which creates the top-line content. The Header must be provided with FSS Text definitions. It is not (yet) intended to allow input fields.

BOTLINES Procedure

Allows you to embed an FMTLIST screen into a frame of your own. It must be activated by defining the bottom lines of the FMTLIST screen, and the label which creates the bottom lines content. It is not (yet) intended to allow input fields.

The following example shows the definition of a frame consisting of 3 header and footer lines:

```
 1 _screen.TopRow=4
 2 _screen.TopRow.Proc="x34Header"
 3 _screen.BotLines=3
 4 _screen.BotLines.proc="X34Footer"
 5 call fmtlist ,,copies(' ',20)'Volumes of your MVS3.8','Volume  Unit Device','VOLUMES'
 6 return 0
 7 /* ------------------------------------------------------------
 8  * VOLUMES Frame Header
 9  * ------------------------------------------------------------
10  */
11 x34Header:
12   delim=copies("=",80)
13   hdr  =Center("Volume List derived from Hercules definitions",80)
14   Address FSS
15   'TEXT 1 2 #PROT+#HI+#White delim'
16   'TEXT 2 2 #PROT+#HI+#RED hdr'
17   'TEXT 3 2 #PROT+#HI+#White delim'
18 return 0
19 /* ------------------------------------------------------------
20  * VOLUMES Frame Footer
21  * ------------------------------------------------------------
22  */
23 x34Footer:
24   delim=copies("-",80)
25   cmt  =Center("Use Line Commands of your choice",80)
26   Address FSS
27   'TEXT 24 2 #PROT+#HI+#White delim'
28   'TEXT 25 2 #PROT+#HI+#BLUE cmt'
29   'TEXT 26 2 #PROT+#HI+#White delim'
30 return 0
```

Formatted List Samples

There are several scripts in *BREXX.V2R5M3.SAMPLES* illustrating the usage of FMTLIST.

FMTOPBOT	has an embedded FMTLIST with user-defined header and footer lines.
@STUDENTL	the front end of the VSAM student database example
#BROWSE	Displays the LISTALC command

Debugging Simple Screen Applications

If you need to debug the behaviour of simple screen applications, you can switch on a trace feature in the calling REXX script:

```
_screen.FTRACE=1
```

You get a trace of the performed step within the screen application.

```
 1 /* REXX */
 2 do i=1 to 35
 3     buffer.i='Buffer Line 'i
 4 end
 5 buffer.0=i-1
 6 /*
 7 _screen.color.top2=#yellow
 8 _screen.color.mylist=#red
 9 _screen.color.cmd =#blue
10 _screen.color.stats=#white
```

```
 */
_screen.footer='PF1 Help PF3 Return PF4 Return'
_screen.Message=1
CALL FMTLIST ,,'','','TEST'
```

Displaying Trace in TSO:

```
09:45:27.09 Entering FMTLIST
09:45:27.18 Display Screen
***
The screen is displayed, waiting for the next user action
09:45:56.65 User Action PF08
09:45:56.69 Command Line ''
09:45:56.71 Display Screen
***
The screen is displayed, waiting for the next user action
09:46:42.13 User Action PF07
09:46:42.17 Command Line '10'
09:46:42.20 Display Screen
***
The screen is displayed, waiting for the next user action
09:47:10.09 User Action PF03
09:47:10.09 Command Line ''
***
```

Formatted List Monitor FMTMON

By setting up a formatted list monitor you can monitor certain events on a timely basis. You can for example continuously view updated entries of the Master Trace Table

```
CALL IMPORT FSSAPI
/* ---------------------------------------------------------------------------
* FMTMON is an FSS application that refreshes itself every xxx milliseconds
* the refresh takes place in the call-back procedure MonTimeOut it must
* provide a new buffer or just return
* There is also an enter-key call-back procedure MonEnter where you can
* execute commands, e.g. CONSOLE and modify the buffer if wanted
* ---------------------------------------------------------------------------
*/
call fmtmon "MVS Trace Table",1000
return 0
```

FMTMON calling Syntax

`FMTMON header,[refresh-frequency]`

Parameters:

- **header** – is displayed as title in the FMTMON screen
- **refresh-frequency** – refresh timer in milliseconds

FMTMON Call-Back Procedures

FMTMON requires two call-back procedures, which must be implemented in the calling REXX procedure.

1. MONENTER: is called when has entered input and presses the enter-key

```
/* ---------------------------------------------------------------
* MONENTER Call Back PROC of FMTMON Enter key pressed, do something
* return 0 continue normally
*  4 continue normally, buffer is not touched
*  8 end monitor (as PF3)
*  12 end monitor (as PF4)
```

```
 7 * ---------------------------------------------------------------
 8 */
 9 MonEnter:
10 call CONSOLE arg(1)
11 /* action requested console command */
12 return 0
```

2. MONTIMEOUT: is called when the frequence-time-out has been reached

```
 1 /* ---------------------------------------------------------------
 2 * MONTIMEOUT Call Back PROC of FMTMON Enter key pressed, do something
 3 * Timeout in FSS, you can provide new content in
 4 * BUFFER.i i=1 to number of lines
 5 * BUFFER.0 must contain number of lines
 6 * return 0 continue buffer is unchanged
 7 * 1 continue new buffer provided
 8 * ---------------------------------------------------------------
 9 */
10 MonTimeout:
11 /* arg(1) entry count */
12 /* create new contents of FMTMON Buffer.
13 return
```

FMTMON provide data to display

FMTMON displays the content of the stem variable BUFFER, typically it is updated in the MONTimeout call-back procedure.

The buffer must have the general structure:

BUFFER.0	contains the number of entries in BUFFER
BUFFER.1	contains the first line
BUFFER.2	second line
…	
BUFFER.n	last line

As the name is fixed, it does not need to be passed to FMTMON.

FMTMON predefined Action Keys

- Help key: PF1
- Scrolling keys: PF7/PF8
- Commands: TOP/BOT/UP n(-lines)/DOWN n(-lines)

FMTMON Application display Master Trace Table

This example is stored in: *BREXX.V2R5M3.SAMPLES(MTT)*

FSS Functions as Host Commands

Alternatively to the FSS functions described in ”FSS Function Overview” you can use the FSS Host command API directly. In this case, all definitions, calculations, validations, etc. must be handled by your REXX script directly.

INIT FSS Environment

Initialise the FSS environment; this must be performed before any other FSS call:

```
ADDRESS FSS
'INIT'
```

Defining a Text Entry

```
ADDRESS FSS
'TEXT 'row column attributes text'
```

- **text**: text to be displayed on the screen
- **row**: row where text should be placed
- **column**: column where text should be placed.
- **attributes**: screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section

Defining a Field Entry

```
ADDRESS FSS
'FIELD 'row column attributes field flen [preset]'
```

- **text**: text to be displayed on the screen
- **row**: row where text should be placed
- **column**: column where text should be placed.
- **attributes**: screen attributes, like colours, protected, high-lighted etc. For details refer to the attributes section
- **field**: Screen field name
- **flen**: length of input area representing field name
- **preset**: content initially displayed (optional), defaults to blank

Getting Field Content

```
ADDRESS FSS
'GET FIELD field rexx-variable'
```

- **field**: Screen field name
- **rexx-variable**: variable receiving the field content

Setting Field Content

```
ADDRESS FSS
'SET FIELD field value'
```

or

```
ADDRESS FSS
'SET FIELD field 'rexx-variable'
```

- **field**: Screen field name
- **value**: new field content
- **rexx-variable**: variable containing the field content

Setting Cursor to a field

Sets the cursor to the beginning of the Screen Field

```
ADDRESS FSS
'SET CURSOR field'
```

- **field**: Screen field name

Setting Colour

Sets the Colour of a Screen Field

```
ADDRESS FSS
'SET COLOR field/text colour'
```

- **field**: Screen field name
- **colour**: Color definition,for details refer to the attributes section

Getting action Key

When the user presses an action-key on a screen, the key value can be fetched in a rexx-variable:

```
ADDRESS FSS
'GET AID rexx-variable'
```

- **rexx-variable**: variable receiving the action key

Display or Refresh Formatted Screen

Used to display the Formatted Screen the first time, or to refresh an active screen:

```
ADDRESS FSS
'REFRESH'
```

End or Terminates FSS Environment

Ends the Formatted Screen environment and releases all used main storage:

```
ADDRESS FSS
'TERM'
```

Get Terminal Width

```
ADDRESS FSS
'GET WIDTH rexx-variable'
```

- **rexx-variable**: variable receiving the action key

Get Terminal Height

```
ADDRESS FSS
'GET HEIGHT rexx-variable'
```

- **rexx-variable**: variable receiving the action key

Application Guide

The BREXX Application Guide describes some of the applications available with. They may not be fully developed, or tested, or foolproof, but they demonstrate the capabilities of BREXX and might be valuable to you.

Installation

After the installation of BREXX will find them in the RXLIB library and can be invoked directly from your scripts.

RXDIFT

RXDIFT compares two datasets and shows their differences.

RXDIFT (new-dsn, old-dsn**[**, option1**][**, option2**]**)

Parameters:

- **new-dsn** – This dataset is considered new how it evolved from old-dsn
- **old-dsn** – dsn which is the source for the compare
- **option1** – Either *ALL* (show all changed/deleted/unchanged lines) or *CHANGES* (show just changed/deleted lines). Default *CHANGES*.
- **option2** – Either *DETAILS* (show progress) or *SUMMARY* (show only summary). Default *SUMMARY*.

We have 2 scripts which slightly are different, with the following REXX we compare them and display the changes with FMTLIST

Example 1:

```
1 file1='PEJ.EXEC(dbdoc1)'
2 file2='PEJ.EXEC(dbdoc2)'
3
4 rarray=RXDIFF(file1,file2,'ALL','DETAILS')
5   buffer.0='ARRAY 'rarray
6   hdr1='New    Old     'file1'<-'file2
7   hdr2='Lino   Lino    Lines'
8  call FMTLIST ,,hdr1,hdr2
```

Results:

```
New    Old     PEJ.EXEC(DBDOC1)<-PEJ.EXEC(DBDOC2)
Lino   Lino    Lines
08:23:47.288   0.000  Compare PEJ.EXEC(DBDOC1) with PEJ.EXEC(DBDOC2)
08:23:47.329   0.040  Dataset PEJ.EXEC(DBDOC1) read
08:23:47.370   0.039  Dataset PEJ.EXEC(DBDOC2) read
08:23:47.376   0.005  Datasets Hashes created
08:23:47.389   0.011  Temporary Arrays created
08:23:47.390   0.000  Start Compare process
08:23:47.393   0.003  Large overlap found 11 lines
08:23:47.395   0.005  Compare process ended, differences determined
08:23:47.399   0.003  Sequences analysed
Differences of PEJ.EXEC(DBDOC1)(new) with PEJ.EXEC(DBDOC2)(old)
00001  00001   call import KeyValue
00002  00002   call dbmsglv 'N'
00003  00003   say "OPEN  "DBOPEN()           /* Open Key/Value Database */
00004  00004   say "ROOM  "DBROOM('WORLD')       /* switch to WORLD  */
00005  00005   call dbremove('QUA',"Continent")  /* Remove records with
00006  00006   call dbremove('ANY',"Mu")         /* Remove records
00007  00007   call dbremove('CONTAINS',"265")
00008  00008   call dbremove('ONLY',"Wa")        /* Remove records with a
00009  00009   call dbremove('ALL')              /* Remove all records of
00010  00010   call dblist('ANY',"Mu")
00011  00011   call dblist('QUA',"Continent")
**del  00012   call dblist('ONLY',"Wa")
**del  00013   call dblist('CONTAINS',"265")
**del  00014   say "CLOSE "DBCLOSE()
00012  **ins   say "CLOSE "DBCLOSE()
deleted  lines 3
```

```
inserted lines 1
moved    lines 0
08:23:47.403   0.004  Summary produced
08:23:47.405   0.001  Cleanup completed
08:23:47.406   0.117  Compare completed PEJ.EXEC(DBDOC1) with PEJ.EXEC(DBDOC2)
```

Example 2:

```
1 file1='PEJ.EXEC(dbdoc1)'
2 file2='PEJ.EXEC(dbdoc2)'
3
4 rarray=RXDIFF(file1,file2)
5   buffer.0='ARRAY 'rarray
6   hdr1='New    Old     'file1'<-'file2
7   hdr2='Lino   Lino    Lines'
8  call FMTLIST ,,hdr1,hdr2
```

Results:

```
New    Old     PEJ.EXEC(DBDOC1)<-PEJ.EXEC(DBDOC2)
Lino   Lino    Lines
Differences of PEJ.EXEC(DBDOC1)(new) with PEJ.EXEC(DBDOC2)(old)
**del  00012   call dblist('ONLY',"Wa")
**del  00013   call dblist('CONTAINS',"265")
**del  00014   say "CLOSE "DBCLOSE()
00012  **ins   say "CLOSE "DBCLOSE()
deleted  lines 3
inserted lines 1
moved    lines 0
```

RXCOPY

RXCOPY is a speedy dataset copy service which handles the copy utilizing the original MVS tools (REPRO and IEBCOPY).

RXCOPY (`new-dsn`, `old-dsn`**[**, `volume`**][**, `option`**]**)

Parameters:

- **new-dsn** – The dataset to be created
- **old-dsn** – the dataset you're copying
- **volume** – The volume serial name that will receive the copied dataset. If omitted, MVS chooses the volume.
- **option** – 'REPLACE'/'REPRO' REPLACE replaces any existing target dataset which is in the system catalogue. REPRO is used to duplicate sequential datasets. The DCB information from the source dsn is utilized to create the target dsn before the copy process.

```
call rxcopy('pej.temp','PEJ1.TEMP',,'REPLACE')
```

Results:

```
-------------------------------------------------------------------------
RXCOPY PEJ.TEMP INTO PEJ1.TEMP REPLACE
-------------------------------------------------------------------------
DSN PEJ.TEMP is sequential, invoke REPRO
Create 'PEJ1.TEMP' with DSORG=PS,RECFM=VBM,UNIT=SYSDA,LRECL=137,BLKSIZE=1692,PRI=1,SEC=1
'PEJ1.TEMP' successfully created
NUMBER OF RECORDS PROCESSED WAS 15
```

IEBCOPY copies partitioned datasets. The DCB information from the source dsn is used to generate the target dsn before the copy procedure. IEBCOPY must be in authorized mode, therefore if you run it within ISPF, it must be authorised. Plain TSO is authorised, so you may run it there.

```
call rxcopy('pej.temp80','PEJ1.TEMP',,'REPLACE')
```

Results:

```
-------------------------------------------------------------------------
RXCOPY PEJ.TEMP80 INTO PEJ1.TEMP80 REPLACE
-------------------------------------------------------------------------
DSN PEJ.TEMP80 is partitioned, invoke IEBCOPY
Target Dataset 'PEJ1.TEMP80' has been removed, due to remove option
Create 'PEJ1.TEMP80' with DSORG=PO,RECFM=FB,UNIT=SYSDA,LRECL=80,BLKSIZE=6400,PRI=25,SEC=3,DI
'PEJ1.TEMP80' successfully created
Prepare IEBCOPY
IEBCOPY completed, RC=0 0
1                                      IEBCOPY MESSAGES AND CONTROL STATEMENTS
-IEB167I  FOLLOWING MEMBER(S)  COPIED  FROM INPUT DATA SET REFERENCED BY SYSUT1   -
IEB154I  PEJ1     HAS BEEN SUCCESSFULLY  COPIED
IEB154I  PEJ2     HAS BEEN SUCCESSFULLY  COPIED
IEB144I  THERE ARE 0000024 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY SYSUT2
IEB149I  THERE ARE 0000000  UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
IEB147I  END OF JOB -00 WAS HIGHEST SEVERITY CODE
```

JES2 Spool Viewer

The JES2 Spool Queue Viewer wants to add some more functionality to the ISPF3.8 function.

You can run the spool viewer with the following command:

```
rx  'BREXX.V2R5M3.SAMPLES(JESQUEUE)'
```

Some examples of the tool:

Main Menu:

```
---------------------- JES2 Primary Option  Menu    ----------------------
Option ===>

         Type an Option and press Enter"

         LOG        Display the System Log
         DA         Display Active Users of the System
         I          Display Jobs in the JES2 Input Queue
         A          Display Jobs Executing
         O          Display Jobs in the JES2 Output Queue
         H          Display Jobs in the JES2 Held Queue
         SYS        Display System Details
         DASD       Display Available Volumes
```

Display Jobs (option *o*):

```
               JES2 Spool Queue of MVSC
SPOOL ==>
        Job Name   Number    QUEUE   STATUS   LINES
***** . ************************************************** Top of      Data    ********
00001 . IBMUSER    TSU00001  PRTPUN  ANY
00002 . INIT       STC00004  PRTPUN  ANY
00003 . INIT       STC00005  PRTPUN  ANY
00004 . INIT       STC00006  PRTPUN  ANY
00005 . INIT       STC00019  PRTPUN  ANY
00006 . INIT       STC00020  PRTPUN  ANY
00007 . INIT       STC00021  PRTPUN  ANY
00008 . INIT       STC00025  PRTPUN  ANY
00009 . INIT       STC00026  PRTPUN  ANY
00010 . INIT       STC00027  PRTPUN  ANY
```

```
00011 . INIT       STC00031  PRTPUN  ANY
00012 . INIT       STC00032  PRTPUN  ANY
00013 . INIT       STC00033  PRTPUN  ANY
00014 . INIT       STC00037  PRTPUN  ANY
00015 . INIT       STC00038  PRTPUN  ANY
00016 . INIT       STC00039  PRTPUN  ANY
00017 . INIT       STC00043  PRTPUN  ANY
00018 . INIT       STC00044  PRTPUN  ANY
00019 . INIT       STC00045  PRTPUN  ANY
00020 . INIT       STC00049  PRTPUN  ANY
00021 . INIT       STC00050  PRTPUN  ANY

Linecmd S view, SJ create JCL, P purge, O send to class, XDC export to      dsn
```

Display DASD (option *dasd*):

```
FSSAPI ==>
      MVS DASDs
***** ***************************************************** Top of Data ******************
00001 Active DASDs
00002 ------------------------------------------------------------------------
00003 UNIT TYPE STATUS  VOLSER VOLSTATE   UNIT TYPE STATUS  VOLSER VOLSTATE
00004 150  3350 S       MVSRES PRIV/RSDNT 151  3350 A       MVS000 PRIV/RSDNT
00005 152  3350 A       PAGE00 PRIV/RSDNT 153  3350 A       SPOOL1 PRIV/RSDNT
00006 180  3380 A       PUB000 PRIV/RSERV 190  3390 A       PUB001 PRIV/RSERV
00007 220  2314 O       SORTW1  PUB/RSERV 221  2314 O       SORTW2  PUB/RSERV
00008 222  2314 O       SORTW3  PUB/RSERV 223  2314 O       SORTW4  PUB/RSERV
00009 224  2314 O       SORTW5  PUB/RSERV 225  2314 O       SORTW6  PUB/RSERV
00010 250  3350 O       SMP000 PRIV/RSDNT 251  3350 A       WORK00 STRG/RSDNT
00011 252  3350 A       WORK01 STRG/RSDNT 253  3350 A       SYSCPK PRIV/RSDNT
***** ***************************************************** End of Data ******************
```

Data Exchange between different MVS Environments

There is an easy way to exchange data between MVS systems.

Starting the Stargate Server:

```
1 rc=stargate('RECEIVE',,3205)
2 say 'Stargate ended with RC='rc
3 return
```

13:14:54.884085 ..BASIC 3205 Stargate TCP Server start at Port: 3205

Launch the Stargate Client, which transmits and requests services and datasets:

Here are some screenshots, just an overview:

Tailoring the list of target MVSes

Edit *BREXX.V2R5M3.SAMPLE(SGTCPLST)*:

```
;; ---------------------------------------------------------------
;; Tailor the TCP Address you usually use to access Stargate Servers
;;    the format is
;;      IP-ADDRESS port-number comment
;;      comment is optional
;; ---------------------------------------------------------------
   xxxx1.yyyyyyy.dddd           3205   my system 1
   xxxx2.yyyyyyy.dddd           3205   my system 2
   xxxx3.yyyyyyy.dddd           3205   my system 3
   xxxx4.yyyyyyy.dddd           3205   my system 4
   xxxx5.yyyyyyy.dddd           3205   my system 5
```

Implementation Restrictions

The name of a variable or label, and the length of a literal string may not exceed 250 bytes. More characters than 250 will be truncated. (Can be changed from rexx.h)

Numbers follows C restrictions, thus integers are long and real numbers are held as double.

The FOR and simple counts on a DO instruction, and the right-hand term of an exponentiation may not exceed maximum long number.

The control stack (for DO, IF, CALL, etc.) is limited to a nesting level of 256 and from the internal stack of the Operating system.

Functions and subroutines cannot be called with more than 15 arguments (Can be changed from rexx.h).

Input and Output cannot be redirected for commands executed through INT2E.

Variables

Variables are held in a binary tree, where the tree is balanced when one branch starts to become very big. Even though the variables are stored as a bintree there is an internal cache system for the faster access.

Each variable in rexx is a length prefix string, and it is kept in memory in 3 different types, long, real, string according the last operation that affect that variable.

```
1  ie. a = 2   /* will be kept as string (Length-prefixed) */
2      a = 2 + 1/* will be kept as integer (long) */
3      a = 2 + 0.1/* will be kept as real (double) */
```

The advantage of the above scheme is that numerical operations are performed much faster than the other algorithms. The main disadvantage is on the integer operations. 32 bit integers have a maximum of 2billion, so if you try something like this

```
1      factorial = 1
2      do i = 1 to 50
3              factorial = factorial * i
4      end
```

will result to 0 instead of the factorial of 50! To find the correct result you have to fool the interpreter to think that factorial is real and not integer, this can be done if you write factorial = 1.0 ….

You can easilly translate a variable to any format you like with the following instructions

```
1      a = a + 0.0   /* will translate a to real */
2      a = trunc(a)  /* will translate a to integer */
3      a = a || ''   /* will translate a to string */
```

Sometimes it is very important to know how a variable is kept in memory (usually for the INTR function) so there is an extra option in DATATYPE function “TYPE” that returns the way one variable is hold.

```
1      DATATYPE(2,"TYPE")      -> "STRING"
2      DATATYPE(2+0.0,"TYPE") -> "REAL"
3      DATATYPE(2+0,"TYPE)     -> "INT"
```

C routines are used for the translation of string to number, so a string like ‘- 2’ will be reported by DATATYPE as a NUMber when rexx tries to evaluate it as a number it will return a value of 0 instead of -2, because of the spaces between the sign and the number.

Stems

substitution to stems may be anything including strings with any character. No translation to uppercase is done to subscripts

```
1      lower = 'ma'
2      stem.lower    -> 'STEM.ma'
3      upper = 'MA'
4      stem.upper    -> 'STEM.MA'
```

Stems can be initialized with a command like stem. = ‘Initial value’

Functions

TRANSLATE sometimes wont work properly for strings with characters above ASCII 127. Works OK for Greek character set.

VARTREE wont work properly with variables with non-printable characters

Migration and Upgrade Notices

This section covers the changes in the new version, migration instruction to upgrade from the previous Release. The installation process is separately described in the BREXX/370 Installation section.

Upgrade from a previous BREXX/370 Version

Before upgrading backup your system. The easiest way is creating a copy of your TK4- directory containing all of your settings DASD volumes. In the cases of errors or unwanted behaviour, you can easily recover to the backup version.

BREXX V2R1M0

Important Changes

Due to the extended calling functionality in the new version of BREXX/370 an import of required REXX scripts is no longer necessary. For this reason, all pre-defined import libraries have been removed from the JCL Procedures RXTSO and RXBATCH. The installation will update them in SYS2.PROCLIB. For similar reasons the CLISTs RX and REXX are no longer necessary and will be therefore removed from SYS2.CMDPROC, there will be an RX and REXX member in SYS2.LINKLIB which replaces the CLIST version.

> **Important**
>
> If you made changes or extensions in the PROCLIB Member RXTSO and/or RXBATCH save the changes to re-introduce them in the newly installed version.
>
> If you made changes or extensions in the CMDPROC Members RX and/or REXX save them to incorporate them in a newly created function of your own.

Libraries

The following BREXX libraries are necessary for running REXX:

- BREXX.JCL
- BREXX.SAMPLES
- BREXX.SAMPLIB **new with this version**
- BREXX.RXLIB

They will be delivered and created during the installation process, existing libraries will be overwritten!

> **Warning**
>
> If you made changes or added your own entries in one of these libraries, save them before beginning with the installation process!

Calling external REXX Scripts or Functions

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

```
CALL your-script parm1,parm2...
```

or a function call:

```
Value=your-script(parm1,parm2,...)
```

The called script will be sought by the following sequence:

- Internal subprocedure or label (contained in the running script)
- current library (where the calling REXX is originated)
- BREXX.RXLIB

Important

The variable scope slightly differs from IBM's REXX implementation. In IBM's implementation a call to an external REXX script, the called REXX is always treated as a procedure without access to the caller's variable pool. BREXX can access and update the caller's pool. If a PROCEDURE is used in the called BREXX script, variables can be made available by the EXPOSE statement.

Software Changes requiring actions

The STORAGE function has been changed to become compatible with IBM's z/OS REXX STORAGE version.

In IBM's REXX, the storage address must be in hex format, in BREXX it was decimal. With this version, we match with IBM's specification and allow only hex notation.

Important

If you have created REXX scripts using the STORAGE function and dislike to update all of them, you can replace them by the BSTORAGE function, which still works with decimal addresses. BSTORAGE is a REXX function being part of the BREXX: RXLIB library.

New Functionality

BREXX functions coded in REXX

`ABEND` (`abend-code`)
Terminate program with Abend-Code, produces an SYSUDUMP

`USERID` ()
Signed in UserId (available in Batch and Online)

`WAIT` (`wait-time`)
Stops REXX script for some time, wait-time is in hundreds of a second

`WTO` (`console-message`)
Write a message to the operator's console

`SYSVAR` (`request-type`)
a TSO-only function to retrieve certain TSO runtime information request-type:

Request Type	Description
SYSENV	FORE/BACK - Foreground TSO / Batch TSO
SYSISPF	ACTIVE/NOT ACTIVE
SYSPREF	TSO Prefix (only available in Foreground TSO)

SYSUID	Userid

Brexx has the capability code new functions or command in REXX. They are transparent and will be called in the same way as basic BREXX function.

Overview:

BSTORAGE (...)
Storage command in the original BREXX decimal implementation

LISTALC ()
Lists all allocated Datasets in this session or region

BRXMSG (...)
Standard message module to display a message in a formatted way, examples:

```
1 rc=brxmsg( 10,'I','Program has been started')
2 rc=brxmsg(100,'E','Value is not Numeric')
3 rc=brxmsg(200,'W','Value missing, default used')
4 rc=brxmsg(999,'C','Division will fail as divisor is zero')
```

will return::

BRX0010I PROGRAM HAS BEEN STARTED BRX0100E VALUE IS NOT NUMERIC BRX0200W VALUE MISSING, DEFAULT USED BRX0999C DIVISION WILL FAIL AS DIVISOR IS ZERO

DAYSBETW (date1, date-2, ...)
Return days between 2 dates

JOBINFO (request-type)
Return information about currently running job or TSO session.
Request-type:

- JOBNAME - returns job name
- JOBNUMBER - returns job number
- STEPNAME - returns step name
- PROGRAMNAME - returns running program name

LINKMVS (load-module, parms)
Starts a load module. Parameters (if any) will send in the format JCL would pass it.

MVSCBS ()
Allows addressing of some MVS control blocks. This function must be imported, then the following functions can be used: Cvt(), Tcb(), Ascb(), Tiot(), Jscb(), Rmct(), Asxb(), Acee(), Ecvt(), Smca()

PDSDIR (dsn)
Return directory entries in a stem variable

RXDATE (...)
Return and optionally convert dates in certain formats

RXDSINFO (dsn/dd-name, options)
Return dsn or dd-name attributes

3RXDYNALC (...)
Allows dynamic allocations of datasets or output

RXSORT (...)
Sorts a stemvariable with different sort algorithms

SEC2TIME (seconds)
Converts an amount of seconds into the format [days]hh:mm:ss

SORTCOPY (stem-variable)
Copies any stem variable into the stem SORTIN, which can be used by RXSORT

STEMCOPY (source-stem-variable, target-stem-variable)
Copies any stem variable into another stem variable

TODAY ()

Returns todays date in certain formats

BREXX V2R2M0

Software Changes requiring actions

OPEN ()
In the *OPEN(ds-name,, 'DSN')* function the third parameter DSN has been removed to achieve closer compatibility to z/OS REXX.
Change Required: Replace OPEN(ds-name,,'DSN') by OPEN('ds-name',) putting the ds-name in quotes or double-quotes signals BREXX that an open for a ds-name instead of a dd-name. If you use a BREXX variable instead of a fixed ds-name the quotes must be coded like this:

```
file='BREXX.RXLIB'
OPEN("'"file"'")
```

QUALIFY ()
The QUALIFY(ds-name) function added in TSO environments the user-prefix. This function has been removed as its functionality has been integrated into the OPEN function.

BSTORAGE ()
The BSTORAGE function has been removed as it was a temporary solution for users of the very first BREXX/370 version. If you plan to keep it, take a copy from the previous RXLIB library.

Reduction of Console Messages

In previous releases, console messages have been displayed during the search of a called REXX script in the BREXX search path. It reported the library name if it was not located in the specific library. This messaging has been significantly brought down. These messages only appear if the called member could not found anywhere in the search path.

Known Problems

Reading Lines fromsequential Dataset

Reading lines of sequential datasets always truncate trailing spaces. This may be an unwanted behaviour for fixed-length datasets. To circumvent this problem you can use the following method:

If the dataset is allocated via a DD statement:

```
X=LISTDSI('INFILE FILE')
fhandle=OPEN(infile,'RB')
Record=READ(fhandle,SYSLRECL)
```

If the dataset is used directly:

```
dsn='HERC01.TEMP'
X=LISTDSI("'"dsn"'")
fhandle=OPEN("'"dsn"'",'RB')
Record=READ(fhandle,SYSLRECL)
```

LISTDSI returns the necessary DCB information (SYSLRECL). The OPEN must be performed with OPTION 'RB' which means READ, BINARY. Read uses the record length to create the record.

BREXX FORMAT Function

The BREXX FORMAT function differs from the standard behaviour of REXX FORMAT:

> FORMAT rounds and formats number with before integer digits and after decimal places. expp accepts the values 1 or 2 (WARNING Totally different from the Ansi-REXX spec) where 1 means to use the "G" (General) format of C, and 2 the "E" exponential format of C. Where the place of the total width specifier in C is replaced by before+after+1. (expt is ignored!)

After determining the code we discovered that a complete re-write would be necessary. As the effort does not stand in proportion to the benefit, we decided to leave it as it is.

New Functionality

BREXX functions

- **Support of Formatted Screens** Refer to the section on formatted screens for more information.
- **Integration of VSAM I/O** Refer to the section on VSAM files for more information.
- **EXECIO Command** Allows accessing sequential datasets either fully or line by line.

`CLRSCRN` ()
Clears the TSO screen by removing all lines from the TSO Buffer.

`CEIL` ()
Returns the smallest integer greater or equal then the decimal number

`FLOOR` ()
Returns the greatest Integer less or equal then the decimal number

`DCL` ()
Enables definition of copybook like definitions of REXX Variables, including conversion from and to decimal packed fields.

`P2D` ()
Converts Decimal Packed Field into REXX Numeric value.

`D2P` ()
Converts REXX Numeric value into Decimal Packed Field.

BREXX V2R3M0

Authorised BREXX Version available

With this release, we ship a standard installation of BREXX as well as an authorised version, which allows to system programs as IEBCOPY, NJE38, etc. The decision about what to install must be made before installation.

BREXXSTD load module removed

We have straightened the load module structure and removed the BREXXSTD load module from the installation library. If you use JCL with an explicit BREXXSTD call, replace it by BREXX. During the installation process, any existing BREXXSTD module will be removed from SYS2.LINKLIB.

Call PLI Functions

Example compile jobs for callable PLI Functions can be found in BREXX.V2R5M3.JCL:

- RXPI calculate PI with 500 digits
- RXCUT Return every n.th character of a string

New and amended functionality

BREXX functions

`CEIL` (`decimal-number`)
CEIL returns the smallest integer greater or equal than the decimal number.

`D2P` (`number`, `length`, `fraction-digit`)
Converts a number (integer or float) into a decimal packed field. The created field is in binary format

`P2D` (`number`, `length`, `fraction-digit`)

Converts a decimal packed field into a number.

ENCRYPT (`string`, `password <`, `rounds>`)

DECRYPT (`string`, `password <`, `rounds>`)
Encrypts/Decrypts a string

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.

DUMPVAR (`'variable-name'`)
DUMPVAR displays the content of a variable or stem in Hex format-

FILTER (`string`, `character-table <`, `drop/keep>`)
The filter function removes all characters defined in the character-table

FLOOR (`decimal-number`)
FLOOR returns the smallest integer less or equal than the decimal number.

LISTIT (`'variable-prefix'`)
Returns the content of all variables and stem-variables starting with a specific prefix

RHASH (`string`, `<slots>`)
The function returns a numeric hash value of the provided string.

ROUND (`decimal-number`, `fraction-digits`)
The function rounds a decimal number to the precision defined by fraction-digits

UPPER (`string`)

LOWER (`string`)
UPPER returns the provided string in upper cases. LOWER in lower cases.

ROTATE (`string`, `position<`, `length>`)
The function returns a rotating substring

TIMESTAMP (**[**, `day`, `month`, `year`**]**)
TIMESTAMP returns the unix (epoch) time, seconds since 1. January 1970.

Dataset Functions

CREATE (`dataset-name`, `allocation-information`)
The CREATE function creates and catalogues a new dataset

DIR (`partitioned-dataset-name`)
The DIR command returns the directory of a partitioned-dataset

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.

REMOVE (`dataset-name`)
The REMOVE function un-catalogues and removes the specified dataset

REMOVE (`partitioned-dataset(member)`)
The REMOVE function on members of a partitioned dataset removes the specified member.

RENAME (`old-dataset-name`, `new-dataset-name`)
The RENAME function renames the specified dataset.

RENAME (`partitioned-dataset(old-member)`, `partitioned-name(new-member)`)
The RENAME function on members renames the specified member into a new one.

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.

FREE (`ddname`)
The FREE function de-allocates an existing allocation of a dd-name.

OPEN (dataset-name, open-option, allocation-information)
The OPEN function has now a third parameter, which allows creating ew datasets with appropriate DCB and system definitions.

TCP Functions

TCPINIT ()
TCPINIT initialises the TCP functionality.

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

TCPOPEN (host-ip, port-number**[**, time-out-secs**]**)
TCPOPEN opens a client session to a server.

TCPWAIT (**[**, time-out-secs**]**)
TCPWAIT is a Server function; it waits for incoming requests from a client.

TCPSEND (clientToken, message**[**, timeout-secs**]**)
SendLength=TCPSEND(clientToken, message[,time-out-secs]) sends a message to a client.

TCPReceive (clientToken**[**, time-out-secs**]**)
Receives a message from another client or server.

TCPTERM ()
Closes all client sockets and removes the TCP functionality

New BREXX functions coded in REXX

GETTOKEN ()
returns a token which is unique within a running MVS System or in this century

BAS64ENC ()
Encodes a string or binary string with Base64.

BAS64DEC ()
Decodes a base64 encoded string into a string or binary string Returns the hash number of a string

STIME ()
Time since midnight in hundreds of a second

BREXX V2R4M0

Functions with changed functionality

There is a major change in every time functions. We have increased the precision of the time format from hundreds of a second to milliseconds in some cases to microseconds. If you use them or rely on the format, please change your REXX scripts accordingly:

```
say TIME('L') /* 16:38:03.112765 */
call wait 100 /* now waits 0.1 seconds */
call wait 5000 /* waits 5 seconds */
```

New Functions

This sections contains all new or changed BREXX V2R4M0 functions

DATE (target-date-format, date, input-date-format)
The new date function has now the “used” formats provided by the original REXX.

DATETIME (target-format, timestamp, input-format)
Formats are:

```
 T is timestamp in seconds 1615310123
 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
```

Time (`'MS'/'US'/'CPU'`)
Time has gotten new input parameters:

- MS Time of today in seconds.milliseconds
- US Time of today in seconds.microseconds
- CPU used CPU time in seconds.milliseconds

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

LINKPGM (`load-module`, `parms`)
Start a load module. Parameters work according to standard conventions.

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

UNLOCK (`'lock-string'`)
Locks/unlocks a resource to avoid concurrent access to it

TIMESTAMP (**[**, `day`, `month`, `year`**]**)
TIMESTAMP returns the unix (epoch) time, seconds since 1. January 1970.

BREXX V2R4M1

Important Changes

RAKF restrictions lifted

We have removed the rigid RAKF checking during the BREXX startup, which caused unnecessary ABENDS for non-authorized users (e.g. HERC03, HERC04). Some of the BREXX functions which require access to system resources (SVC244, DIAGCMD) are no longer available to non-authorized users, they will be reported as unknown functions.

Matrix and Integer Arrays

Added are mathematical Matrix functions and integer arrays. Both allow high-performance access and large-sized matrices and integer arrays outside the standard stem notation.

About

BREXX has been developed and supported by Vasilis.Vlachoudis@cern.ch

BREXX/370

BREXX/370 is a ported version of BREXX to IBM's operation system MVS 3.8j. Jason Winter and Jürgen Winkelmann ported BREXX initially to MVS3.8j. The BREXX/370 releases have been created and are supported by the BREXX/370 team: Mike Grossmann and Peter Jacob.

License

BREXX is licensed under the GNU General Public License v2.0. See https://github.com/vlachoudis/brexx/blob/master/COPYING All rules mentioned above apply to BREXX/370!

BREXX/370 documentation

The essential BREXX documentation applies to BREXX/370: https://ftp.gwdg.de/pub/languages/rexx/brexx/html/rx.html

Please install according to the Installation Guide.

DISCLAIMER

THE SOFTWARE REFERENCED IS MADE AVAILABLE AS - IS. THE AUTHOR MAKES NO WARRANTY ABOUT THE SOFTWARE AND ITS CONFORMITY TO ANY APPLICATION. THE AUTHOR IS NOT RESPONSIBLE FOR ANY DAMAGE, LOSS OF DATA, OR LOSS OF MONEY CAUSED BY THIS PROGRAM.

Indices and tables

- **genindex**
- **modindex**
- **search**

This user's guide documents the BREXX standard functions from https://ftp.gwdg.de/pub/languages/rexx/brexx/html/rx.html as well as the changes and amendments to BREXX to be used on MVS 3.8j.

Credits

- BREXX has been developed by Vasilis Vlachoudis, who made it publicly available as freeware for non-commercial purposes.
- Jason Winter's JCC Compiler for compiled BREXX
- JCC and the JCC-Library are owned and maintained by him. While not being freeware, Jason allows non-commercial usage and distribution of Software created using JCC through a relaxed license, as long as the complete source code always accompanies those distributions.
- Vasilis and Jason explicitly consented to make the JCC based version of BREXX available on TK4-. Thanks to both for their significant valuable contribution to the TK4- MVS 3.8j Tur(n)key system.
- The VSAM Interface is based on Steve Scott's VSAM API.
- The FSS Part is based on Tommy sprinkle's FSS - TSO Full-Screen Services
- Daniel Gaeta contributed his EXECIO implementation.
- The NJE38DIR load module was extracted out of Bob Polmanter's NJE38 V2 modules

We wish to thank the following persons for patiently answering our questions and for their support and advice:

- Vasilis Vlachoudis
- Jürgen Winkelmann
- Jason Winter
- Wally Mclaughlin
- Greg Price
- Bob Polmanter
- Steve Scott

and many others!

BREXX/370 Source Code

The BREXX/370 Source Code can be found and downloaded at: https://github.com/mvslovers/brexx370/

Some Notes on BREXX Arithmetic Operations

BREXX stores numeric values in the appropriate type format. The benefit compared to save 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, 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 is 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 two a bit-exact reversibility is not always given. This effect may build up during arithmetic calculations.

Index

Symbols

A

B

C

D

T

U

V

W

X