REXXGBLV - REXX function - Global variable support

Contents

Changes in this build (072)

- The AS(newname) operand may now be a mask.

Short description

REXXGBLV addresses the lack of global storage in REXX. While REXX does provide some means of passing data to and retrieve data from subroutines, there is no easy method for storing data for later retrieval.
REXXGBLV provides a variable store external to the currently running REXX program. Variables can be saved and later retrieved by the same or another REXX process. The saved variables can be written to and retrieved from a disk file, so can be used across logons and even be shared amongst users.
The data is stored in a named pool in the home address space above the 16MB border.
Any number of pools can be created, till the virtual storage runs full.

The maximum length of a variable name is 240. There is no maximum data length. Export and import will split the data over more records if needed.
The program is re-entrant, so can be installed in LPALIB. See the 'Performance' section later for other options.
Additionally, REXXGBLV can index variables, copy variables and list variables (useful for debugging).

Syntax

REXXGBLV must run as a function:

```
n= REXXGBLV('function parameters')
```

Refer to the section 'Parameter description' later for a detailed description of the parameters.

Store and load variables to/from pool(s)

Save REXX variables to a pool

```
SAVE VAR(vardef) [DATA(mask)] [POOL(pool)] [AS(name)]
```

See sample ‘Save and retrieve REXX variables’.

Clear an entire pool, then save REXX variables in the pool

```
CLRSAVE VAR(vardef) [POOL(pool)]
```

This is a combined DEL * and SAVE.
*deprecated, use DROPSAVE instead.

Delete pool variables, then save REXX variables in the pool

```
CLRVSAVE VAR(vardef) [DATA(mask)] [POOL(pool)]     *deprecated*
DELSAVE VAR(vardef) [DATA(mask)] [POOL(pool)]
```

This is a combined DEL and SAVE.
This call will delete all pool variables matching ‘name’, then save REXX variables matching ‘name’ in the pool. It can be useful if you previously have stored many variables and now wish to start from scratch.

Delete a pool, then save REXX variables to the pool

```
DROPSAVE VAR(vardef) [DATA(mask)] POOL(pool) [AS(name)]
```

This is a combined DEL * and SAVE.
Poolname is required. You cannot delete the internal pools.

Load pool variables to REXX

```
LOAD VAR(vardef) [DATA(mask)] [POOL(pool)] [AS(name)]
```

REXX variables can optionally be renameded, otherwise they are loaded (and replaced) with the pool variable name.

See sample ‘Save and retrieve REXX variables’.

Load pool variables to REXX, then delete the variables from the pool

PULL VAR(vardef) [DATA(mask)] [POOL(pool)] [AS(name)]
This is a combination of LOAD and DEL.
REXX variables can optionally be renameded, otherwise they are loaded (and replaced) with the pool variable name.

Load pool variables to REXX, then drop the pool

LOADDROP VAR(vardef) [DATA(mask)] [POOL(pool)] [AS(name)] LISTSTEM(stemname)]
REXX variables can optionally be renamedd, otherwise they are loaded (and replaced) with the pool variable name.
Use LISTSTEM to save a list of variables loaded, note that this is only supported for a generic load.

Add data directly to a pool variable

ADD VAR(name) VALUE(data) | VALVAR(varname) [POOL(pool)]

‘data’ can be a string of max 200 characters without parenthesis or other characters which can invalidate the parm string.
‘varname’ is the name of a variable containing the data. Use this instead of VALUE(data) if you are unsure of the data or if the data length is more than 200.

If the last chars of the name are '.+' then a new stem variable is created with suffix = name.0 + 1 and name.0 is updated. If name.0 does not exist it is created. This is intended for making a simple log feature.

See sample ‘Add a pool variable directly’.

Return the value from a pool variable

RETURN VAR(name) [POOL(pool)]
Null is returned if the variable is null, if the variable does not exist, or if the pool does not exist.

Pool handling

List contents of a pool

LIST VAR(vardef) [DATA(mask)] [POOL(pool)] [LENGTH] [NW(name-width)] [DW(data-width)]
[LISTSTEM(listname)]

See the DW, NW, LENGTH and LISTSTEM operands later.

Build a REXX stem containing the names of variables in a pool

INDEX VAR(vardef) [DATA(mask)] STEM(stem) [POOL(pool)]
Build a stem containing the names of variables in a pool matching 'name'. Note the stem is built in REXX, not in the pool.

Delete pool variables

DEL VAR(vardef) [POOL(pool)]
The VAR parameter is required.

Copy pool variables

POOLCOPY POOL(from-pool) POOL2(to-pool) [VAR(vardef)]
Copy some or all variables from one pool to another. to-pool may be new.

Delete a pool

DROP POOL(pool)
Releases storage.
Poolname is required. You cannot delete the internal pools.

Save and load variables to/from external file

Export from a pool to file

EXPORT DD(fileref) [POOL(pool)] [VAR(vardef)] [DATA(mask)] [AS(name)]

Save REXX variables to a pool, then export the pool variables to file

SAVEEXP VAR(vardef) [DATA(mask)] [DD(fileref)] [POOL(pool)] [AS(name)]
This is a combination of SAVE and EXPORT.

Export REXX variables directly to file

REXPORT VAR(vardef) [DATA(mask)] [DD(fileref)] [AS(name)]

Update pool variable(s) in an Export dataset

EXPORTU DDname(fileref)| DSname(dataset) POOL(pool) [VAR(vardef)] [DATA(mask)]

Update REXX variable(s) directly in an export dataset

REXPORTU VAR(vardef) [DATA(mask)] [DDname(fileref)] | DSname(dataset)

Imports file to a pool

IMPORT DD(fileref) [POOL(pool)] [VAR(vardef)] [AS(name)]
File must have been created by the EXPORT/REXPORT function.

Import file to a pool, then load the pool variables to REXX

IMPLOAD DD(fileref) [POOL(pool)] [VAR(vardef)] [DATA(mask)] [AS(name)]
This is a combination of IMPORT and LOAD.
File must have been created by the EXPORT/REXPORT function.

Imports REXX variables directly from file

RIMPORT [VAR(vardef)] [DATA(mask)] [DD(fileref)] [AS(name)]
File must have been created by the EXPORT/REXPORT function.

REXX variable manipulation

Copy REXX variables

COPY VAR(vardef) [DATA(mask)] [AS(as)]
'name' must be generic, like MYVAR. or MYVAR* .
'as' – see the description of the AS parameter later.

Build a stem containing the names of selected REXX variables

RINDEX VAR(vardef) [DATA(mask)] STEM(stem)
Create a listing of the variable names matching 'var' in stem 'stem'.

See sample 'Build a stem of REXX variable names'.

List contents of REXX variables

RLIST VAR(vardef) [DATA(mask)] [LENGTH] [NW(name-width)] [DW(data-width)]
LISTSTEM(listname)]
See the DW, NW, LENGTH and LISTSTEM operands later.
Default is to list to terminal, LISTSTEM can be used to save the list.

Index a REXX stem

STEMIX STEM(stem.)
Indexes a stem by taking the basename and putting a running number as the new suffix, plus .0 of course. This is intended to create an index over a stem with non-numeric suffixes.

Notes
- the stem is not sorted
- if the stem already has entries with numeric suffixes they may be overwritten.

See sample 'Index a REXX stem'.

Index a stem (old)

VARINDEX | VARIX VAR(vardef) STEM(stem) [POOL(pool)]
See INDEX for parameter syntax and usage notes.
deprecated, use INDEX instead

Other

Show build info

VERSION
Shows build id and date/time of assembly.
sample:
ver=REXXGBLV('version')

Map the internal memory storage structure

MAP [XTENDED] XTENDED will map storage entries. For debugging.

Parameter description

AS(newname) Descrete variable, rename to 'as' value.
Generic variables, replace the 'var' name with the 'as' value.
i.e. var(KILROY.) and AS(WASHERE.) => variable KILROY.A -> WASHERE.A
i.e. var(KILROY.) and AS(WASHERE) => variable KILROY.A -> WASHEREA
It is highly recommended that you use a mask instead, this generic feature may be deprecated in a later build. To use a mask to replace the stem name, do AS('NEWID.'>'.'>*) .
Mask, using mask characters as follows:

%	String character is copied to the output.
+c	Insert character.
+'text'	Insert text.
-	Delete character.
-'text'	Delete up till, but not including text.
*	Copy remainder to output.
*'text'	Copy string characters from the current position to 'text'. 'text' itself is not copied.
?text?if-found?not-found?	If text matches string at cursor then use if-found mask, else use not found mask. Either mask may be null.
?>text?if-found?not-found?	If text matches string anywhere at or after cursor then use if-found

mask, else use not-found mask. Either mask may be null.

\ The char following in the mask is copied to output, this allows you to copy a mask character - i.e. \%

other Mask character is copied.

Maximum newname parameter length is 60.

See members RXGVSAMP and RXGVIVP1 for samples.

DATA(mask) Data filter when name is generic.
Mask characters are * for all or none, % for single.
Note that the mask is case sensitive.
i.e. DATA(K*r*)

DD(fileref) Optional fileref for export and import. Default is RXGVEXP.

DW(data-width) Width of data - LIST/RLIST functions. Default=85
If specified as 0 then the actual data width is used.

NW(name-width) Width of name - LIST/RLIST functions. Default=35
If specified as 0 then the actual name width is used.

LENGTH Show name- and data length for LIST and RLIST.

POOL(pool) Pool name, max length=16. The name is translated to uppercase, it may contain any characters, but may not begin with '-', '(' or ')'.

STEM(name) Stem name – do include the dot.
Maximum length is 240 bytes.

TRACE Writes a lot of diagnostic info.

VALUE(data) Data to store for the VAR(name) variable. Used with the ADD function.
Maximum length is 200 bytes, do not use parenthesis or quotes.
Not valid with VAR(list).

VALVAR(name) Name of variable containing data. Used with the ADD function when the VALUE parameter would conflict with the syntax parser.
Not valid with VAR(list).

VAR(name | mask | list) Variable definition. Can contain wildcards '*' for multiple or null characters and '%' for a single character. Maximum length is 240 bytes.
A trailing dot (.) defines a stem base.
'list' can be a combination of names and masks.

DD statements

RXGVEXP Fileref for the EXPORT, IMPORT, REXPORT and RIMPORT functions.
RECFM=VB, LRECL should be half-track size. Make LRECL=27994 and BLKSIZE=27994 and you should be ok on a 3390 (like) disk.

Alternative DDname can be defined by the DD operand.

Variables set at termination

Some information is externalized through the variables below. They are all returned, though most will contain zeroes.

REXXGBLV_EXPORTN Number of entries exported to pool.

REXXGBLV_IMPORTN Number of entries imported from pool.

REXXGBLV_MAXDL	Max data length
REXXGBLV_MAXNDL	Max name+data length
REXXGBLV_MAXNL	Max name length
REXXGBLV_MSG	Failure message
REXXGBLV_RLISTN	Number of REXX variables listed.
REXXGBLV_RLOADN	Number of REXX variables loaded.
REXXGBLV_VDELN	Number of pool variables deleted.
REXXGBLV_VLISTN	Number of pool entries listed
REXXGBLV_VLOADN	Number of REXX variables loaded.
REXXGBLV_VSAVEN	Number of REXX variables saved.
REXXGBLV_STATS	Other statistics.

Principles of operation

REXXGBLV creates a storage area and uses the IEANTCR services to build name/token pairs to save the information for the areas. The pool name can be set in the call to REXXGBLV, thus allowing a virtually unlimited number of variable stores.

Storage

Pool storage is obtained from storage pool 131, so it survives till the TSO online- or batch session is terminated (step end) unless specifically freed.
All storage, except 64K used for I/O operations, is requested from above the line. Using 64-bit storage or data spaces has been deemed unnecessary so far.
The initial amount of storage for a pool is 1452 bytes. As the pool sees activity this will of course increase, actual amount depending on the number of variables saved and the combined size of the stored variables.
Storage is not necessarily released when a variable is deleted.
Depending on internal settings storage is released or moved to free queues from where it can be quickly reused. This is done as an attempt to avoid excessive getmain/freemain activity.
Not all storage is released when a pool is dropped. Some basic control blocks and a number of free elements are kept as an attempt to avoid excessive getmain/freemain activity. The maximum storage retained for a dropped pool is currently 11K.
There are 3 internal pools so assuming one user pool has been created used and dropped, the storage retained would be 44K.

Samples

Save and retrieve REXX variables

Note that REXX variables are not necessarily shown in alphabetical order.

```
parse value 'Kilroy was here' with p1 p2 p3
cc=REXXGBLV('save pool(p1) var(p*)')
say 'List pool.....'
cc=REXXGBLV('list pool(p1)')   /* list stored */
say 'List actual...'
```

```
cc=REXXGBLV('rlist var(p*)')   /* list actual */
p1='The president'
say 'List actual after setting P1'
cc=REXXGBLV('rlist var(p*)')   /* list actual */
cc=REXXGBLV('load  pool(p1)')  /* reload      */
say 'List actual after reload'
cc=REXXGBLV('rlist var(p*)')   /* list actual */
```

displays

```
List pool.....
P1                                Kilroy
P2                                was
P3                                here
# of records listed: 00000003
List actual...
P1                                Kilroy
P3                                here
P2                                was
List actual after setting P1
P1                                The president
P3                                here
P2                                was
List actual after reload
P1                                Kilroy
P3                                here
P2                                was
```

Add a pool variable directly

Callers code:

```
cc=RexxGblv('del var(log.)')
/*or cc=RexxGblv('add var(log.0) value(0)') */
Call Sub1
cc=RexxGblv('load var(log.)')
do n=1 to log.0
  say log.n
end
```

Sub1 code:

```
cc=RexxGblv('add var(log.+) value('time() 'Program started)')
  .. some code ..
cc=RexxGblv('add var(log.+) value('time() 'Program ended)')
Return 0
```

Will show something like this:

```
15:24:33 Program started
15:24:34 Program ended
```

Export and import directly to/from file

Fileref

```
//RXGVEXP  DD RECFM=VB,LRECL=27994,SPACE=(TRK,(5,5)),UNIT=SYSDA,..
```

Program

```
     Alpha    = 'Letter nr 01'
     Charlie  = 'Letter nr 03'
     India    = 'Letter nr 09'
     Juliet   = 'Letter nr 10'
     Romeo    = 'Letter nr 18'
     Sierra   = 'Letter nr 19'
     Tango    = 'Letter nr 20'
     Zulu     = 'Letter nr 26'
     cc=RexxGblv('rexport var(*ie*)')
     say REXXGBLV_EXPORTN+0 'variables exported'
     drop alpha charlie india juliet romeo sierra tango zulu
     say  alpha charlie india juliet romeo sierra tango zulu
     cc=RexxGblv('rimport var(*lie*)')
     say REXXGBLV_IMPORTN+0 'variables imported'
     say  alpha charlie india juliet romeo sierra tango zulu
     cc=RexxGblv('rimport')
     say REXXGBLV_IMPORTN+0 'variables imported'
     say  alpha charlie india juliet romeo sierra tango zulu
```

Shows

```
     3 variables exported
     ALPHA CHARLIE INDIA JULIET ROMEO SIERRA TANGO ZULU
     2 variables imported
     ALPHA Letter nr 03 INDIA Letter nr 10 ROMEO SIERRA TANGO ZULU
     3 variables imported
     ALPHA Letter nr 03 INDIA Letter nr 10 ROMEO Letter nr 19 TANGO ZULU
```

Build a stem of REXX variable names

```
     data.0   = 'zero'
     dataf    = 'sixth letter in the alphabet'
     foxtrot  = 'foxtrot is a dance'
     datad    = 'delta'
     faulty   = 'Faulty Towers'
     function = 'test filter'
     cc=REXXGBLV('rindex var(f*t*) stem(filter.)')
```

creates

```
     FILTER.0 = '00000003'
     FILTER.1 = 'FAULTY'
     FILTER.2 = 'FUNCTION'
     FILTER.3 = 'FOXTROT'
```

Index a REXX stem

```
     CAR.BMW   = 'german car make'
     CAR.FORD  = 'USA car make'
     CAR.MAZDA = 'japanese car make'
     cc=RexxGblv('stemix stem(car.)')
```

creates

```
     CAR.0  = '00000003'
     CAR.1  = 'BMW'
     CAR.2  = 'MAZDA'
```

```
CAR.3  = 'FORD'
```

Load variables, use the list stem and VALUE function to retrieve data.

```
cc=RexxGblv('load pool(T071) var(*li* *el*) liststem(lst.)')
do n=1 to lst.0
  say 'lst.'n left(lst.n,12) '->' value(lst.n)
end
```

See member RXGVSAMP for more samples.

Installation

See member REXXGBLV.
If you do not want to use the stay-in-JPAQ feature, then change the statement

```
CDEUCTZ2 dsect=N
```

to a comment.
Review and update the RXGVCUST member.

Initial verification

See members RXGVIVP (program) and RXGVIVP$ (JCL). The REXXGBLV installation step is set up to execute the RXGVIVP member as well.
The IVP job can also be used for samples, but the RXGVSAMP member is probably more readable.

Performance

REXXGBLV is fully reentrant, so can reside in LPALIB or linklist.
LPALIB is recommended. If run from JOBLIB or STEPLIB the program will reload itself, thus remaining in JPAQ - or in other words there will be only one program load for the entire session. See the 'Install' section for how to disable this feature.

Comments etc

Comments can be sent as email to

Willy(at)harders-jensen(dot)com
Substituting (at) with @ and (dot) with .

Web

The newest version of REXXGBLV, plus much much more, is found at https://harders-jensen.com

Have fun

Willy Jensen