



**(Partial) Findings from the 1st run of tests vs BReXX/370's Release I,  
Version V2R1M0**



by reggiemack



Hi all!

This document addresses several findings that revealed themselves while the 1<sup>st</sup> major fully operational release of BRexx/370 (i.e., version V2R1M0) is under evaluation. The porting of BRexx to BRexx/370 is pretty solid. Most of the findings are minor giving rise to the robustness of the port. Wunderbar!

I am looking forward to continuing to complete the evaluation of the rest of the language elements of BRexx/370's release 1 and then on to release 2 which is close at hand to be published.



## Table of Contents

Introduction.....	4
1. Running [tso] rx - .....	6
2. Running [tso] r[ex]x -a .....	8
3. GetEnv('PATH') .....	10
4. STREAM BiF's Rexx Standard Open Commands (see GitHub Issue #32) .....	10
5. Duplicate Labels.....	11
6. STREAM open commands of UPDATE and APPEND .....	12
7. STREAM open commands of UPDATEBINARY and APPENDBINARY.....	14
8. The SOUNDEX Function dysfunction .....	16
9. BSTORAGE() Function .....	18
10. ADDR/BADDR.....	18
11. FORMAT function is non-standard .....	19
<del>12. I/O Functions default to DDN vs DSN (See: GitHub Issue #26) .....</del>	<del>20</del>
13. Interesting Feature(s) .....	21



## Introduction

As we now know ...

BRexx - is an open source version of the classic Rexx developed by Vasilis Vlachoudis of the Cern Laboratory in Switzerland. Written as an ANSI C language program, BRexx is notable for its high performance. It also provides a nice collection of special built-in functions and function libraries that offer many extra features over most of the existing Classic Rexx implementations.

BRexx was originally written for DOS in the late eighties/early 1990s. Then, with the rise of Windows, it was revised for the 32-bit world of Windows and 32-bit DOS. BRexx also runs under the Linux and Unix family of operating systems, and has a good set of functions especially written for Windows CE. Other operating systems on which it runs include: MacOS, BeOS, and the Amiga OS.

One of the outstanding features, among so many, of BRexx is its minuscule footprint. The entire product, written in C, including full documentation and examples, takes up only a measly few hundred kilobytes. It is small enough to fit on a single, ol' school floppy diskette.

Hence, BRexx was prime for re-targeting to MVS/VM for TSO/CMS. And so it was. Recently, the BRexx Rexx Interpreter was made operational onto the MVS 3.8j platform as BRexx/370 version V2R1M0 and it was released in April of 2019. The MVS 3.8j (now emulated) mainframe platform which pre-dates IBM's official release of TSO/CMS Rexx published initially, in the next (i.e., XA) set of offerings of IBM mainframe OSs. However, the lack of having a Rexx significantly hampers the MVS 3.8j platform of not only the highly utilitarian features of Rexx as the powerful glue it provides as an operating system's command/macro language, tying all of the vast collection of system resources together programmatically, under one umbrella, but also prevents the users and the MVS 3.8j community from taking advantage of the treasure trove of Rexx-ware scripts/programs/applications that exists currently which greatly enhances the MVS 3.8j user's and MVS 3.8j community's mainframe computing experience, many-fold.

BRexx was made operational, as BRexx/370, targeting the MVS 3.8j platform by: Peter Jacob (PEJ), Mike Großmann (MIG) and Gerard Wassink (GAW). BRexx/370 now provides a Rexx scripting/command language for the MVS 3.8j platform. All accolades to these gentlemen for their efforts in bringing BRexx/370 to the fore. Thank-you so very, very much! However, aside from the BRexx/370 installation guide and a BRexx/370 reference for the new Built-In (BiFs) and library (RxLib) external functions for BRexx/370 in the Users' Guide, there exists very little BRexx/370 specific documentation, as yet, confirming which of the original BiFs were ported versus those that were not. A documentation effort that is a major, major undertaking! Thus, we are relegated to BRexx's original documentation. There are some platform specific BiFs that are obvious in the fact that they are not germane to the MVS 3.8j platform like the Unix/Linux or Windows/MS-DOS specific functions. However, there exists some other functions that have some subtleties that are not so glaring and either work differently under BRexx/370 or do not work at all or were not ported. Currently, the only way to determine the



BRexX/370 status of each of the BRexX/370 language elements especially the BiFs is work through all of the language elements through trial & error.

This called for addressing the BRexX/370 language elements (LEs) 1 by one and establishing whether or not each of the LEs addressed thus far were ported and what works and what doesn't. As mentioned before, the BRexX/370 interpreter is quite solid with few anomalies, thus far. The protocol used for evaluation is a priority of guidelines:

1. Adherence to the Rexx Standard (as much as possible)
2. LE Compatibility with IBM's Rexx370
3. Brexx specific LE compliance where apropos BRexx

And, in addition, to verify any new (i.e., non-BRexx) features/functionality implemented in BRexX/370.

TO BE CONTINUED ...

Although the review of release 1 of BRexX/370 has not been completed (there is quite a lot to review), Early publication was needed to submit item topic 4 ([here](#)) as an issue related to the GitHub issue #26 ([here](#)) that was recently closed and implemented for the soon to be published release 2 of BRexX/370 ...



## 1. Running [tso] rx -

The BRexx documentation on running BRexx describes the several ways to run BRexx from the syntax: `r[ex]x [-[trace]|-a|-F] rexx-program [args...]`. Most work for BRexx/370 under TSO (or RFE/RPF) with Vista to TK4- on Windows. For example:

- a. [TSO] rexx execname ...
- b. [TSO] rx - say sin(0.5)\*sqrt(3\*\*2+4\*\*2)
- c. [TSO] rexx -a execname ...
- d. [TSO] rexx -?r execname ...
- e. [TSO] rexx - do i = 1 to 10; say i; end

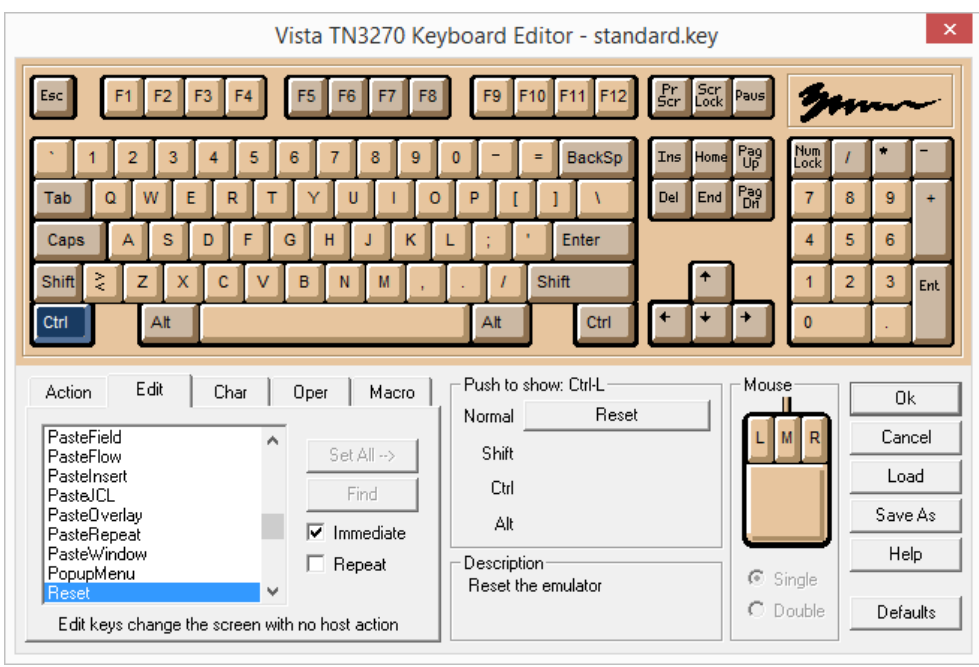
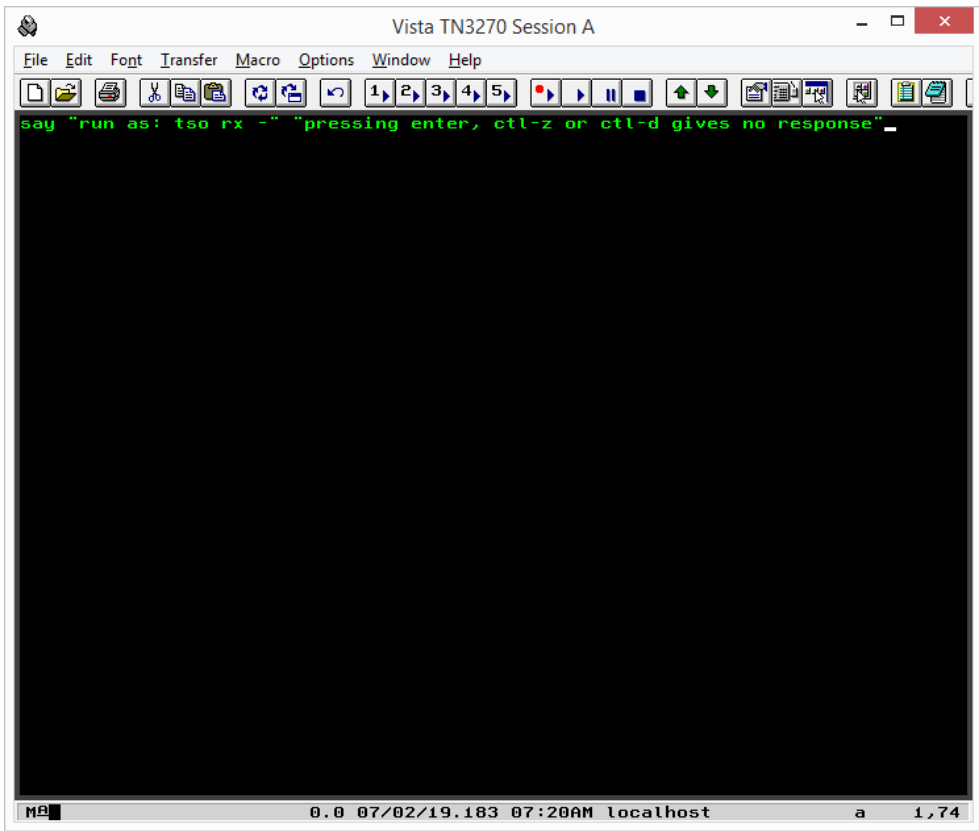
All of the above work beautifully (except for a glitch in c, see topic #2. next). However, the documentation also states the following:

“If there is no arguments following the ‘-’, then BRexx will wait for the user to type into **STDIN** the program. To end the program type *Ctrl-Z ...*”

As an example:

```
rexx - (press return, and type...)  
do x=0 to 6.28 by 0.1  
  y = trunc(39*(sin(x)+1)) + 1  
  say copies(' ',y) || '*'  
end  
Ctrl-Z
```

This does not appear to be working. When attempting, by typing “[tso] r[ex]x -” (without the quotes, of course) at TSO’s “READY” prompt or under RFE|RPF|[ISPF], pressing return to advance to the next line to continue program entry does not work. In fact, when pressing enter, absolutely nothing happens. Entering ctrl-z (or ctrl-d) with the standard keyboard set-up using Vista’s TN3270 on TK4- under Windows 8.1 gives the same response, *nothing*. Needless to say, other key[combinations]s are to no avail. The session, at this point, is hung. To continue, one has to utilize the MVS console to cancel the TSO session and then return to the terminal window to login, again. Just terminating the 3270 session and restarting it to reconnect to the TSO session returns one back to the point of stasis, HUNG! So I don’t believe that it’s a problem of terminal emulation. Without this feature functioning we are limited to 1-liners under the “[TSO] r[ex]x -” directive, a downer when doing BRexx/370 development and testing. What would be nice is allowing enter to take you to the next line and ctrl-z to end program entry and initiate execution. If this feature is not to be implemented, please advise. If anyone has run into (and hopefully, solved) this problem, any direction would be most welcome ...





## 2. Running [tso] r[ex]x -a

The BRexx documentation on running BRexx with the `-a` parameter describes the execution as utilized to break the argument string into multiple arguments. Like

```
[TSO] r[ex]x -a foo.r one "two three" four
```

with `foo.r` being:

```
do i=1 to arg(); say i":" arg(i);end
```

The execution of `foo.r` should result in the following display ...

```
1: one
2: two three
3: four
```

However, when tested with BrexX/370 there is a problem with the outcome. I coded the small test program called `FOO1` where `FOO1` is

```
/* Rexx */
do i=1 to arg(); say i":" arg(i);end;
exit
```

After running the script like:

```
[TSO] r[ex]x -a foo1 one "two three" four
```

the results display as:

```
1: one
2: two three"four
```

As you can see, the parameters haven't been parsed quite exactly as they should be. The following screenshots of the execution demonstrate the errant parsing ...





```
REVEDIT  HERC01.EXEC(F001) - 1.00          Columns 00001 00072
Command ==> tso rx -a fool one "two three" four_ Scroll ==> CS
***** *****Autosave***** Top of Data *****
000001 /* REXX */
000002 do i=1 to arg(); say i":" arg(i);end
000003 exit
***** *****Autosave***** Bottom of Data *****
```

7760K FREE

```
1: one
2: two three"four
***
```



### 3. GetEnv('PATH')

This function of BRexx/370 returns zilch (i.e., nothing). It's more of a DOS related function. If there are no future plans for this Built-In Function then maybe it should be removed? Or, maybe as a BRexx/370 BIF it can be retooled to return the session's dataset's high-level qualifier as USERID() which can be construed to be the defacto directory of sorts within a TSO/CMS user session? Although, we already have BIFs that will return the userid (such as USERID(), SYSVAR(), etc.) Or, maybe it's fine just as is? However, I think it deserved mentioning.

### 4. STREAM BiF's REXX Standard Open Commands (see GitHub Issue #32)

The STREAM BIF's file/dataset open commands are not the REXX standard open commands. This leaves a developer burdened with a significant amount of unnecessary coding when developing multi-platform REXX programs and addressing the BRexx/370 environment, especially when the program is heavily laden with STREAM I/O. Thank goodness that it is quite easily remedied (veritably a no-brainer) by applying an alias to the existing commands in the STREAM BiFs commands' case (switch) statement in 'rxfiles.c'. The STREAM BIF has the following syntax:

```
STREAM( streamid[, [option][, command]] )
```

To open a file/dataset stream, the [option] must be 'C' for command mode and then the file/dataset open directive as 'COMMAND' where, for BRexx/370 and its open commands to be standardized, 'COMMAND' becomes 'READ', 'WRITE', 'APPEND', 'UPDATE' or 'CREATE'.

The table here references the existing STREAM BIF's open commands to be aliased and then, associates the equivalent REXX standard STREAM BIF open, data stream, command string to be added as an alias.

BRexx/370 STREAM BIF Open Command	REXX STREAM BIF Standard Open Command (Alias)
'READ'	'OPEN'
'READ'	'OPEN READ'
'WRITE'	'OPEN WRITE'
'APPEND'	'OPEN WRITE APPEND'
'UPDATE'	'OPEN BOTH'
'CREATE'	'OPEN WRITE REPLACE'

For the BRexx/370 binary versions of the above, like 'READBINARY', BINARY is to be added to the end of its non-BINARY counterpart as an alias. So the BINARY version of "OPEN READ would be 'OPEN READ BINARY'. Likewise, APPENDBINARY's alias would then become 'OPEN WRITE APPEND BINARY' for example ...



## 5. Duplicate Labels

Under BRexx/370, in certain instances, one is allowed to have multiple labels with same name (i.e., duplicate labels). It's so blatant that I'm not sure as to whether this is a bug or a feature 😊. However, I am of the mind that most developers would want to be notified, if in fact, they'd coded duplicate labels in error. I can't think of a reason for which a developer would code duplicate labels on purpose. So, I vote for notification (Informational? Warning?) as long as another pass over the source or tokens is not required to do so. I've written a small BRexx program to demonstrate a duplicate label scenario. And, as you can see the program performed flawlessly regardless of the proliferation of duplicate labels.

```
REVEDIT  HERC01.EXEC(DUPLABEL) - 1.00          Columns 00001 00072
Command ==>  top_rx_DupLabel_                Scroll ==>  CS
***** *****Autosave***** Top of Data *****
000001 /* rexx */
000002
000003 parse source g.RxSystem g.RxCall g.RxExec
000004
000005 DupLabel_:
000006
000007   g := ""; g.Date.Time = DATE('S')'.D'DATE('D')'B'TIME('L')
000008
000009   say ""
000010   say g.RxExec": Start`Date.Time :=>" g.Date.Time
000011   say
000012
000013 DupLabel_Beg:
000014
000015   call DupLabel_End
000016
000017 DupLabel_End:
000018 DupLabel_End:
000019 DupLabel_End:
000020 DupLabel_End:
000021 DupLabel_End:
000022
000023   say ""
000024   say g.RxExec": Commence Date.Time ==>: ' g.Date.Time
000025   say g.RxExec": Terminal Date.Time ==>: DATE('S')'.D'DATE('D')'B'TIME('L')
000026   say g.RxExec": Elapsed Time ==>: TIME('E')
000027
000028   signal DupLabel_Exit
000029
000030 DupLabel_Exit:
000031 DupLabel_Exit:
000032 DupLabel_Exit:
000033 DupLabel_Exit:
000034 DupLabel_Exit:
000035
000036   exit
000037
000038 /***** End of Program *****/
***** *****Autosave***** Bottom of Data *****
7756K ERPF
```

```
: Start`Date.Time :=> 20190814.0226001:50:26.50
: Commence Date.Time ==>: 20190814.0226001:50:26.50
: Terminal Date.Time ==>: 20190814.0226001:50:26.50
: Elapsed Time ==>: 0.1490809917449952
*** _
```



## 6. STREAM open commands of UPDATE and APPEND

The versions of BREXX/370's STREAM function's OPEN commands test just fine except for the UPDATE and the APPEND versions. For some reason, when attempting to open a file/dataset as UPDATE or APPEND via BREXX/370's STREAM function, like:

STREAM(ddn, 'C', 'UPDATE')

-OR-

STREAM(ddn, 'C', 'APPEND')

A fatal error 57 is returned ...

I have coded a test script to illustrate the issue please see the screen-shots following ...

```
REVEDIT  HERC01.EXEC(APPENDIO) - 1.01                Columns 00001 00072
^Command ==> tso rx appendio brexx370.appendio_      Scroll ==> CS
***** *****Autosave***** Top of Data *****
000001 parse upper arg DSN
000002 say "DSN ==>" DSN
000003 DDN = "APPENDIO"
000004 say "DDN ==>" DDN
000005
000006
000007 /* READ BINARY */
000008
000009 'ALLOC DA('DSN') FI('DDN') SHR'
000010 x=STREAM(DDN,'C','READ')
000011 say "Open as Read Status of" DSN "==" x
000012 x=CLOSE(DDN)
000013 'FREE FI('DDN')'
000014
000015
000016 /* WRITE BINARY */
000017
000018 'ALLOC DA('DSN') FI('DDN') OLD'
000019 x=STREAM(DDN,'C','WRITE')
000020 say "Open as Write Status of" DSN "==" x
000021 x=CLOSE(DDN)
000022 'FREE FI('DDN')'
000023
000024
000025 /* UPDATE BINARY */
000026
000027 'ALLOC DA('DSN') FI('DDN') OLD'
000028 x=STREAM(DDN,'C','APPEND')
000029 say "Open as Update Status of" DSN "==" x
000030 x=CLOSE(DDN)
000031 'FREE FI('DDN')'
000032
000033 exit
***** *****Autosave***** Bottom of Data *****
7716K FREE
```



```
DSN ==> BREXX370.APPENDIO
DDN ==> APPENDIO
Open as Read Status of BREXX370.APPENDIO ==> READY
Open as Write Status of BREXX370.APPENDIO ==> READY
28 *-* x=STREAM(DDN,'C','APPEND')
Error 57 running APPENDIO, line 28: Cannot open file
*** _
```

The same results occur when running the test issuing the `STREAM(DDN,'C','UPDATE')` Stream I/O Built-in-Function, `STREAM`'s open `UPDATE` command option in BREXX/370. The other open options are working as documented.



## 7. STREAM open commands of UPDATEBINARY and APPENDBINARY

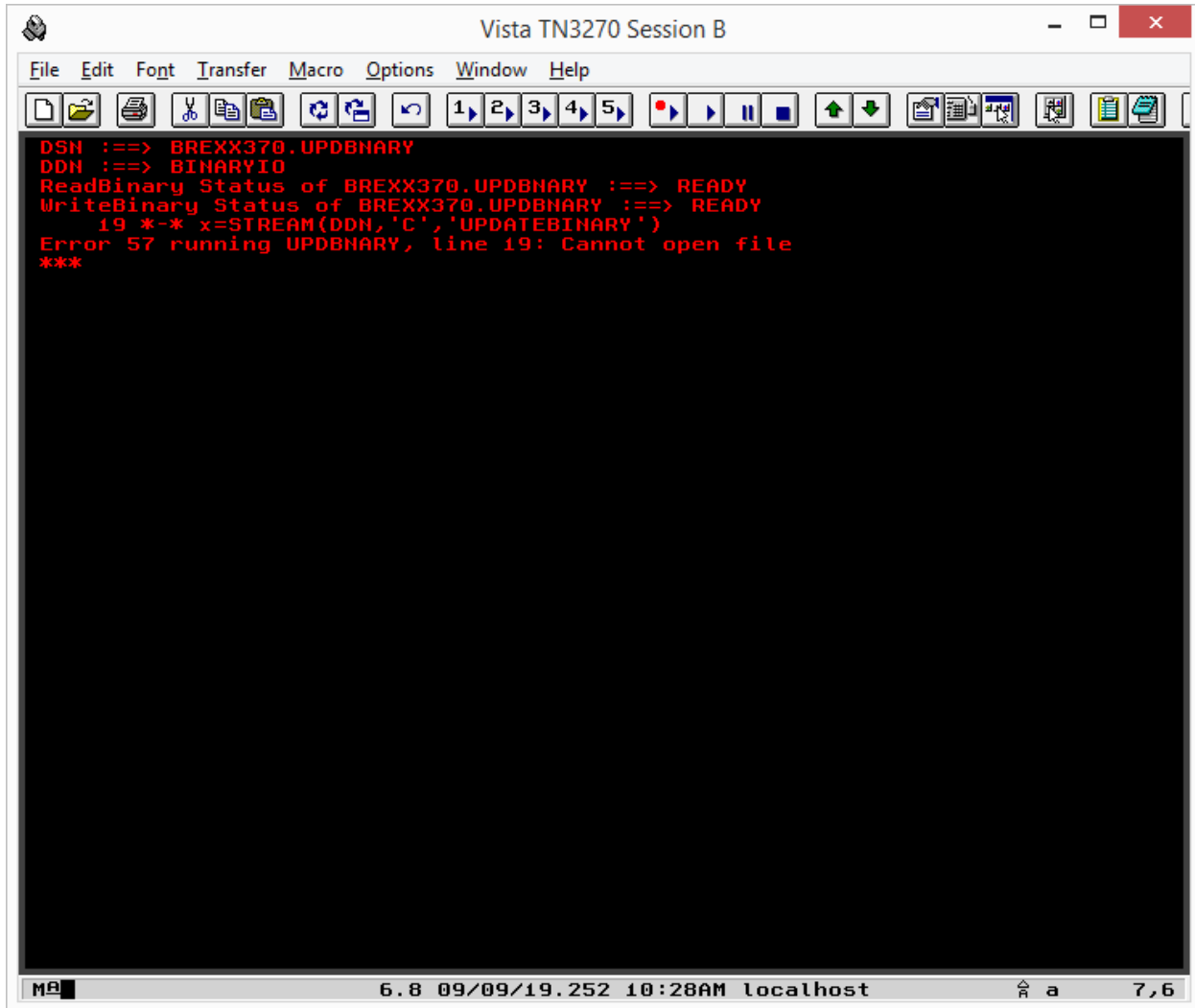
The BINARY versions of BRexx/370's STREAM function's OPEN commands test just fine except for the UPDATEBINARY and the APPENDBINARY versions. For some reason, when attempting to open a file/dataset as UPDATEBINARY or APPENDBINARY via BRexx/370's STREAM function, like:

```
STREAM(ddn, 'C', 'UPDATEBINARY')  
-OR-  
STREAM(ddn, 'C', 'APPENDBINARY')
```

A fatal error 57 is returned ...

I have coded a test script to illustrate the issue please see the screen-shots following ...

```
Vista TN3270 Session B  
File Edit Font Transfer Macro Options Window Help  
REVEDIT HERC01.EXEC(UPDBINARY) - 1.01 Columns 00001 00072  
1 Command ==> Scroll ==> CS  
***** *****Autosave***** Top of Data *****  
000001 parse upper arg DSN  
000002 say "DSN ==>" DSN  
000003 DDN = "BINARYIO"  
000004 say "DDN ==>" DDN  
000005  
000006  
000007 /* READ BINARY */  
000008  
000009 'ALLOC DA('DSN') FI('DDN') SHR'  
000010 x=STREAM(DDN,'C','READBINARY')  
000011 say "ReadBinary Status of" DSN " ==>" x  
000012 x=CLOSE(DDN)  
000013 'FREE FI('DDN')'  
000014  
000015  
000016 /* WRITE BINARY */  
000017  
000018 'ALLOC DA('DSN') FI('DDN') OLD'  
000019 x=STREAM(DDN,'C','WRITEBINARY')  
000020 say "WriteBinary Status of" DSN " ==>" x  
000021 x=CLOSE(DDN)  
000022 'FREE FI('DDN')'  
000023  
000024  
000025 /* UPDATE BINARY */  
000026  
000027 'ALLOC DA('DSN') FI('DDN') OLD'  
000028 x=STREAM(DDN,'C','UPDATEBINARY')  
000029 say "UpdateBinary Status of" DSN " ==>" x  
000030 x=CLOSE(DDN)  
000031 'FREE FI('DDN')'  
000032  
000033 exit  
***** *****Autosave***** Bottom of Data *****  
7724K FREE  
M 0.0 09/09/19.252 11:03AM localhost a 27,2
```

A screenshot of a terminal window titled "Vista TN3270 Session B". The window has a menu bar with "File", "Edit", "Font", "Transfer", "Macro", "Options", "Window", and "Help". Below the menu bar is a toolbar with various icons for file operations and navigation. The main area of the window is black with red text. The text shows the execution of a COBOL program with the following output:

```
DSN ==> BREXX370.UPDBNARY
DDN ==> BINARYIO
ReadBinary Status of BREXX370.UPDBNARY ==> READY
WriteBinary Status of BREXX370.UPDBNARY ==> READY
19 *-* x=STREAM(DDN,'C','UPDATEBINARY')
Error 57 running UPDBNARY, line 19: Cannot open file
***
```

At the bottom of the window, there is a status bar with the text "6.8 09/09/19.252 10:28AM localhost" and a small icon.

I get the same result when running the test issuing the `STREAM(DDN,'C',APPENDBINARY)` Stream I/O Built-in-Function `STREAM`'s open binary command options in BRExx/370. The other open binary options seem to be working, just fine.



## 8. The SOUNDLEX Function dysfunction

The SOUNDLEX function produces certain codes that represent the syllabic sounds of a word. Thereby, allowing the processing of words by how they sound (enunciated) vs how they are spelled. The BrexX/370 version of SOUNDLEX being ported from the original BrexX produces a code that differs from that established by the BrexX documentation. For example, the BrexX documentation indicates that 'monday' and 'mandei' have equivalent sounds (in English). After evaluation by the SOUNDLEX() Built-in-Function (BiF), they both should produce the same code, M530:

```
SOUNDLEX('monday') ==> SOUNDLEX('mandei') ==> 'M530'
```

However, in BrexX/370, the SOUNDLEX() BiF produces 2 different codes from each other as well as different from that produced from the original BrexX, per the documentation. When the SOUNDLEX() BiF is employed to evaluate the same 2 strings what is produced can only fully be displayed in hex.

```
SOUNDLEX('monday') = vx'M.&.'      x'D4205020'      1
                    D252                |
                    4000                | → EBCDIC
SOUNDLEX('mandei') = vx'M.&.'      x'D4175020'      1
                    D152                |
                    4700                |
```

The vx stands for vertical hex. I surmise that the differences may be due to a complication of the character encoding (i.e., ASCII vs. EBCDIC) somewhere within the Soundex algorithm. Anyway, take a look at the screen-shots following for a glimpse of the SOUNDLEX() function in action.





```
REVEDIT  HERC01.EXEC(TSTSNDX) - 1.00          Columns 00001 00072
Command ==> tso rx tstsndx_                Scroll ==> CS
*****  
*****Autosave***** Top of Data *****  
000001 /*  
000002 Returns a 4 character soundex code of word (in english) ...  
000003 say SOUNDDEX('monday') /* M530 */  
000004 say SOUNDDEX('Mandei') /* M530 */  
000005 */  
000006  
000007 x=SOUNDDEX('monday')  
000008 say "the SOUNDDEX() code for the string 'monday' is ==>" x  
000009  
000010  
000011 x=SOUNDDEX('monday')  
000012 say "the SOUNDDEX() code for the string 'mondei' is ==>" x  
000013  
000014 exit  
*****  
*****Autosave***** Bottom of Data *****
```

7760K FREE

```
the SOUNDDEX() code for the string 'monday' is ==> M:&:  
the SOUNDDEX() code for the string 'mondei' is ==> M:&:  
***  
_
```



## 9. BSTORAGE() Function

In the original BRexx STORAGE() BiF, which is now BSTORAGE(), one was allowed to insert data at what is now the BSTORAGE() location. The data parameter is completely ignored. The BRexx/370 Users' guide says:

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

Storage command in the original BREXX decimal implementation. The storage address is in decimal.

Did we miss this functionality or was the ability to update storage at the decimal-storage-address purposely left out? Also, what would now be BSTORAGE(), that is, without a parameter was allowed under BRexx but raises an error in BRexx/370. I'm assuming that a parameter-less BSTORAGE() is no longer allowed?

## 10. ADDR/BADDR

To be compatible with the STORAGE() & BSTORAGE() BRexx/370 functions where STORAGE() mimics IBM's Rexx370 utilizing hexadecimal addresses and BSTORAGE, the original STORAGE function in BRexx, I'd like to recommend BRexx/370 having an ADDR() & BADDR. ADDR() would function with hexadecimal addresses and BADDR() would function as the original with decimal addresses.

**If:**

a = "Hello" and a is located at decimal address 4096

**then:**

ADDR('a') should return '1000'x

**and:**

BADDR('a') from RxLib should return 4096

One (or 2) less data type conversions keeping one to less hair-pulling 😊.



## 11. FORMAT function is non-standard

The BRexx/370 FORMAT function, by definition under the Math functions documentation, is not the standard version of the Rexx FORMAT Built-in-Function as demonstrated in the BRexx documentation on FORMAT():

**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 */
```

I recommend for Rexx platform portability's sake, that *eventually*, we do like what was done with the STORAGE() BiF, re-establishing the current FORMAT() BiF as BFORMAT and implementing the FORMAT() BiF as a Rexx standard FORMAT() BiF. I believe that I ran across a RexxFormat() external function somewhere that looked like it might fit the bill and provide the standard Rexx FORMAT() BiF functionality.

Oh yes, I did find it in the BRexx distribution, a RxLib external function called RexxFormat(). However, it is not fully flushed out and is not operational at this time. It's probably 70% complete and eventually could become the standard (although external) version of the FORMAT() function for BRexx/370 with not too significant of an effort ...



## 12. I/O Functions default to DDN vs DSN (See: [GitHub Issue #26](#))

### This has been addressed in release 2 (V2R2M0) of BRexx/370

The Stream I/O Built-in-Functions (BiFs) of BRexx/370 only operate with the DDName of a dataset. To be compliant with the Rexx Standard and with IBM's Rexx370 (a.k.a., TSO Rexx) the Stream I/O commands also must be able to operate with the DSName of a particular file. The arguments for me for accommodating the dataset name as the target of the Stream I/O functions (CHARIN, CHAROUT, CHARS, LINEIN, LINEOUT, LINES, STREAM) are several:

- It's part of the Rexx standard and is provided for in z/OS's REXX370 under the STREAM I/O functions package.
- The ability to utilize the dataset name as the target of the Stream I/O functions under BRexx/370, taking advantage of the dynamic allocation of the dataset eliminating the need to code and evaluate the execution of the cumbersome file allocation command and the commands usually associated with it (DELETE, ATTRIB, FREE, etc.) will reduce time spent on development and maintenance.
- It will facilitate the ease of developing multi-platform Rexx applications especially when heavily data driven.
- One can still continue to use the DDName with Stream I/O and/or EXECIO if one chooses and not be affected at all. So its implementation would be backward compatible.

The implementation doesn't appear that overwhelming, at least on the surface:

"If the target of any of the Stream I/O function is not a valid, currently allocated, DDName then continue processing as a dataset name (DSName). Otherwise, the apropos error is raised."

#### THEN ...

Under the auspices of issue #26, this has now been addressed and implemented into the in the next release of BRexx/370 (V2R2M0). The StreamIO BiFs (CHARIN, CHAROUT, CHARS, LINEIN, LINEOUT, LINES, STREAM), will now accept, as the 1<sup>st</sup> argument, either a DSName or a DDName given the following criteria:

- a. Quoted file-names are treated as DSNames
- b. Unquoted file names are DDNames
- c. There will be no automatic prefixing of the DSNames (by User ID) in TSO

However, a sister requirement ([GitHub issue #32](#)) to the now closed [GitHub issue #26](#) is to bring to standard the command options of the STREAM BiF as outlined here in topic [#4](#).



### 13. Interesting Feature(s)

- When invoking a subordinate BRexx/370 script via call or as a function, it turns out that the functions and subroutines that reside within subordinate exec become available to the calling exec once the subordinate program has executed at least once. It's as if the subordinate script was IMPORT'd and not just invoked, which internally is probably part of the actual mechanism utilized to facilitate a subordinate invocation. I see this as a feature that can be manipulated to write more trim & efficient BRexx/370. Again, I just thought it was worth mentioning ...