Using REXX during Assembly

Invoking REXX during High Level Assembly via SETCF

by ColeSoft, a leader in all things assembler

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence**.

SHARE in Seattle 2015

Introduction

Today we will talk about forming a bridge between the assembler's macro processing and the REXX language. By doing this we are able to add the power of REXX language to the assembler in order to perform complex operations that are difficult or impossible using the usual assembler constructs.

This is accomplished by using the assembler's SETCF ability to invoke a module, RXBRIDGE in our case, that in turn executes REXX Execs and returns their result to the assembler.

The first part of this presentation will discuss writing and calling REXX Execs and the second part will get into how we do it.

Introduction

Here are a few examples of why we might want to do this.

- Using REXX PARSE to decode macro arguments.
- Recording information from macros to an external file.
- Accessing external data sources to generate tables.
- Manipulating numbers beyond 2^31, the scope of SETA

What we will cover

1. Review of SETC and SETCF
2. The Big Picture of Assembler SETCF
3. Setting up for RXBRIDGE
4. Sample RXBRIDGE Exec
5. Sharing data among RXBRIDGE Execs
6. Down to the Nitty-Gritty
7. Other Possibilities

1. Review of SETC and SETCF

- SETC is used to place a value into a macro variable:

```
&MYVAR SETC 'myvalue'
```

- SETCF also places a values into a macro variable but you are able to identify a program that will generate it:

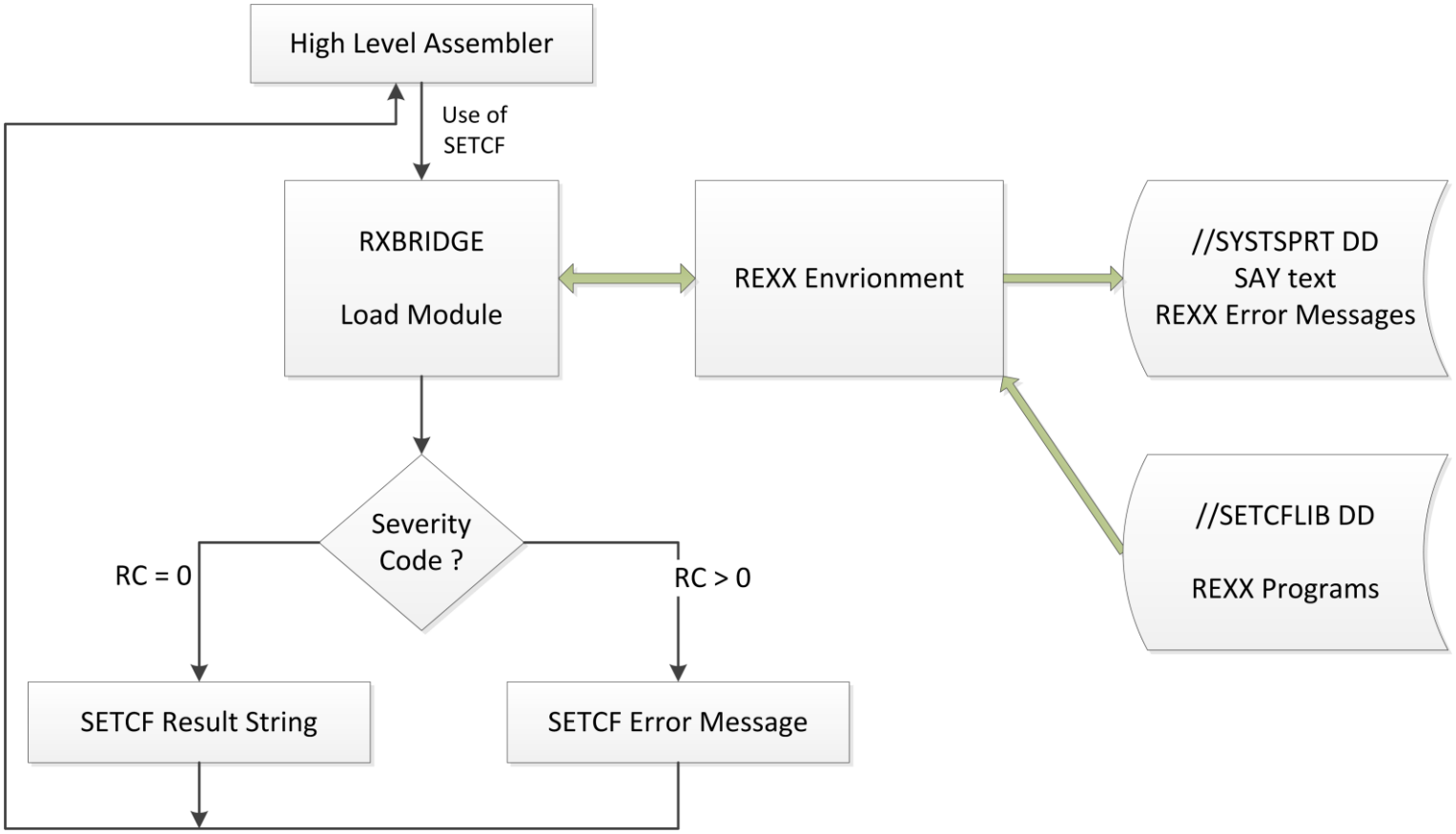
```
&MYVAR SETCF 'module','CFarg1','CFarg2',,,,
```

Review of SETC and SETCF

```
&MYVAR SETCF 'module','CFarg1','CFarg2',,,,
```

- Module is the name of a load module that is brought into memory by the assembler. In this presentation this is RXBRIDGE.
- To avoid reloading the module on every SETCF call the load module should be Link-Edited with the REUS or RENT attribute. When this is done the assembler will LOAD the module once and then just call it after that.
- RXBRIDGE is reusable.

2. The Big Picture of Assembler SETCF



3. Setting up for RXBRIDGE

RXBRIDGE is a SETCF module that invokes REXX.

The module is invoked like this:

```
&MYVAR SETCF 'RXBRIDGE','EXECNAME','RXarg1','RXarg2',,,
```

- From the SETCF point of view we have
  - The program load module is RXBRIDGE
  - CFarg1 = EXECNAME
  - CFarg2 = RXarg1
  - CFarg3 = RXarg2
  - …
- There may 0 to 200 RXargs.

Setting up for RXBRIDGE

The code in RXBRIDGE invokes a REXX Exec stored as a member in SETCFLIB. The following DD cards need to be added to your assembler JCL:

STEPLIB contains member RXBRIDGE

```
//STEPLIB  DD DSN=my.loadlib,DISP=SHR
```

SETCFLIB contains member EXECNAME

```
//SETCFLIB DD DSN=myfavs.rexx,DISP=SHR
```

SYSTSPRT contains SAY and REXX error messages.

```
//SYSTSPRT DD SYSOUT=*
```

SETCFLIB is hard-coded into RXBRIDGE, but you can change it to be pretty much whatever you want.

4. Sample RXBRIDGE Exec

Here is a sample REXX Exec, named REPLACE:

```
/* REXX SETCF PROGRAM TO REPLACE ONE STRING WITH ANOTHER */
/* THE OPERATION PROCEEDS FROM LEFT TO RIGHT.THE COMPARISON IS CASE
INSENSITIVE */

STRING =             ARG(1)
OLD    = TRANSLATE( ARG(2) )
NEW    =             ARG(3)

DO 1200
   P=POS(OLD,TRANSLATE(STRING)) /* WORK FROM LEFT TO RIGHT CASELESS */
   IF P<1 THEN LEAVE
/* WE HAVE LOCATED OLD INSIDE STRING */
   STRING = DELSTR(STRING,P,LENGTH(OLD))
   STRING = INSERT(NEW,STRING,P-1)
END
RETURN 0 STRING
```

Returning from an RXBRIDGE Exec

```
RETURN rc string
              |________| < This is the SETCF string
        |______________| < This is the returned value
```

- All RXBRIDGE Execs must return a value. REXXEXEC treats the first REXX "word" on the return string as the assembler's return code.
- When rc = 0
  - The text after the first space is returned as the SETCF value.
- When rc > 0
  - The text after the first space is returned as part of an assembler ASMA711W error message.
  - SETCF receives a zero-length string.

Sample invocation of REPLACE

```
         MACRO
&NAME    REPLACE_TEXT &STRING,&OLD,&NEW
&S       SETC  DEQUOTE('&STRING')
&O       SETC  DEQUOTE('&OLD')
&N       SETC  DEQUOTE('&NEW')
.*                     EXECNAME  ARG1 ARG2 ARG3
&DATA    SETCF 'RXBRIDGE','REPLACE','&S','&O','&N'
&NAME    DC    C'&DATA'
         MEND

         REPLACE_TEXT 'THIS is Some sHoRt Text','short','Longer'
+        DC    C'THIS is Some Longer Text'
```

REXX SAY

```
/* REXX Exec to echo text to //SYSTSPRT Dataset – SAYIT */
SAY ARG(1)
RETURN 0
```

You can use the REXX SAY instruction to place text into the //SYSTSPRT dataset. This can be very handy if you want to capture information during the assembly process and write it out to a flat file.

- You might have some tables that are defined with macros and want to capture that information without having to parse the assembler source yourself.
- You want to debug some macro logic but don't want to use MNOTE for some reason.

REXX SAY example

```
          MACRO
          SYSTSPRT &TEXT
&X        SETC  DEQUOTE('&TEXT')
&DATA     SETCF 'RXBRIDGE','SAYIT','&X'
          MEND
          SYSTSPRT 'This is a message'
```

Causes //SYSTSPRT to contain:

```
This is a message
```

Sample error invocation

```
/* REXX EXEC TO CAUSE A MINOR ERROR - MINORERR */
RETURN 4 'Minor ERROR'
```

The SETCF exit accepts the first REXX "word" on the return string as the assembler's return code. The text after the first space is returned as an error message instead of result string of the SETCF.

```
&DATA     SETCF 'RXBRIDGE','MINORERR'
** ASMA711W RXBRIDGE: MINORERR:Minor ERROR

External Function Statistics
 ----Calls----  Message   Highest  Function
 SETAF   SETCF    Count  Severity  Name
     0       1        1         4  RXBRIDGE
```

REXX Execution and Coding Errors

/* REXX Exec that has an error */

This is not good

return 0

If you have a REXX coding or execution error then you will see something like this in the //SYSTSPRT dataset:

```
2 *-* This is not good
  +++ RC(-3) +++
```

5. Sharing data among RXBRIDGE Execs

- Since each REXX Exec is invoked independently it would be helpful if there were a way to pass variables from one RXBRIDGE Exec to another.
- This is done by using variables named GLOBAL_xxxx
  - 64 character maximum name length
  - 512 byte maximum value length
  - 64K limit total
  - Not related to GBLC in any way

Sample Execs showing Globals in action

```
/* REXX Exec to save data into a global - GBL1*/
GLOBAL_ONE = ARG(1)
RETURN 0

/* REXX Exec to save data into a global - GBL2*/
GLOBAL_TWO = ARG(1)
RETURN 0

/* REXX Exec to return some global info - GETGBL12*/
RETURN 0 GLOBAL_ONE GLOBAL_TWO
```

Sample invocation of Execs with Globals

```
&X        SETCF 'RXBRIDGE','GBL1','This is text 1'
&X        SETCF 'RXBRIDGE','GBL2','This is text 2'
&X        SETCF 'RXBRIDGE','GETGBL12'
          DC    C'&X'
+         DC    C'This is text 1 This is text 2'
```

Using PARSE with RXBRIDGE

By using a pair of RXBRIDGE Execs we can access the REXX PARSE instruction with assembly-provided values and templates:

```
/* REXX - Sample RXBRIDGE PARSE Exec - DOPARSE */
INTERPRET "PARSE VALUE" ARG(1)"WITH" ARG(2)
RETURN 0
```

Which is invoked using

```
&X SETCF 'RXBRIDGE','DOPARSE','value','template'
```

Where the template consists of GLOBAL_ variables and controls.

Sample invocation of DOPARSE

```
&X       SETCF 'RXBRIDGE','DOPARSE',                    X
               '"March 4, 2015"',                       X
               'GLOBAL_M GLOBAL_D ", " GLOBAL_Y'
```

Causes REXX to execute

```
PARSE VALUE "March 4, 2015" WITH GLOBAL_M GLOBAL_D ", " GLOBAL_Y
            ---- ARG(1)----      ------------- ARG(2) ----------
```

Which assigns values to the variables GLOBAL_M, GLOBAL_D, and GLOBAL_Y into the RXBRIDGE global pool. These are now available to all future RXBRIDGE Execs.

Retrieving Global Variables

To get back the values of the global variables we can do this:

```
/* REXX Exec to retrieve a Global Variable - GETGBL */
INTERPRET "X = GLOBAL_"ARG(1)
RETURN 0 X
```

Which is invoked as

```
&X SETCF 'RXBRIDGE','GETGBL','varsuffix'
```

That would return the value of GLOBAL_varsuffix.

Sample invocation of GETGBL

Here we show "reading" the global variables that were produced by DOPARSE earlier:

```
&MONTH    SETCF 'RXBRIDGE','GETGBL','M'
&DAY      SETCF 'RXBRIDGE','GETGBL','D'
&YEAR     SETCF 'RXBRIDGE','GETGBL','Y'
          DC    C'&MONTH'    FROM GLOBAL_M
+         DC    C'March'
          DC    C'&DAY'      FROM GLOBAL_D
+         DC    C'4'
          DC    C'&YEAR'     FROM GLOBAL_Y
+         DC    C'2015'
```

Dumping the global variables

You can obtain a listing of the global variables by using this RXBRIDGE statement:

```
ADDRESS RXBRIDGE 'GLOBALS LIST'
```

In //SYSTSPRT:

```
GLOBAL_D='4'
GLOBAL_Y='2015'
GLOBAL_M='March'
      3 GLOBALS LISTED
```

Dumping the global variables

Or you can do it from a macro by having

```
/* REXX Exec to print the Globals - PRTGLOBL */
ADDRESS RXBRIDGE 'GLOBALS LIST'
RETURN 0
```

And coding

```
&X          SETCF 'RXBRIDGE','PRTGLOBL'
```

Setting Global Variables from a macro

To set the value of a global variable we can do this:

```
/* REXX Exec to set a Global Variable - SETGLOBL */
INTERPRET "GLOBAL_"ARG(1)"="""ARG(2)""""
RETURN 0

/* REXX Exec to print the Globals - PRTGLOBL */
ADDRESS RXBRIDGE 'GLOBALS LIST'
RETURN 0
```

Which is invoked by

```
&X SETCF 'RXBRIDGE','SETGBL','varsuffix','value'
```

Setting Global Variables from a macro

For example:

```
          MACRO
&NAME     SET_GLOBALS &DUMMY
&X SETCF 'RXBRIDGE','SETGLOBL','SYSDATE','&SYSDATE'
&X SETCF 'RXBRIDGE','SETGLOBL','SYSECT','&SYSECT'
&X SETCF 'RXBRIDGE','SETGLOBL','SYSMAC','&SYSMAC'
&X SETCF 'RXBRIDGE','SETGLOBL','SYSNDX','&SYSNDX'
&X SETCF 'RXBRIDGE','SETGLOBL','SYSTIME','&SYSTIME'
&X SETCF 'RXBRIDGE','PRTGLOBL'
          MEND
```

Setting Global Variables from a macro

Will produce in //SYSTSPRT:

```
GLOBAL_SYSDATE='03/04/15'
GLOBAL_SYSECT='TEST'
GLOBAL_SYSMAC='SET_GLOBALS'
GLOBAL_SYSNDX='0001'
GLOBAL_SYSTIME='19.15'
      5 GLOBALS LISTED
```

Resetting global variables

You can obtain a listing of the global variables by using this RXBRIDGE statement:

```
ADDRESS RXBRIDGE 'GLOBALS RESET'
```

Using the REXX Stack

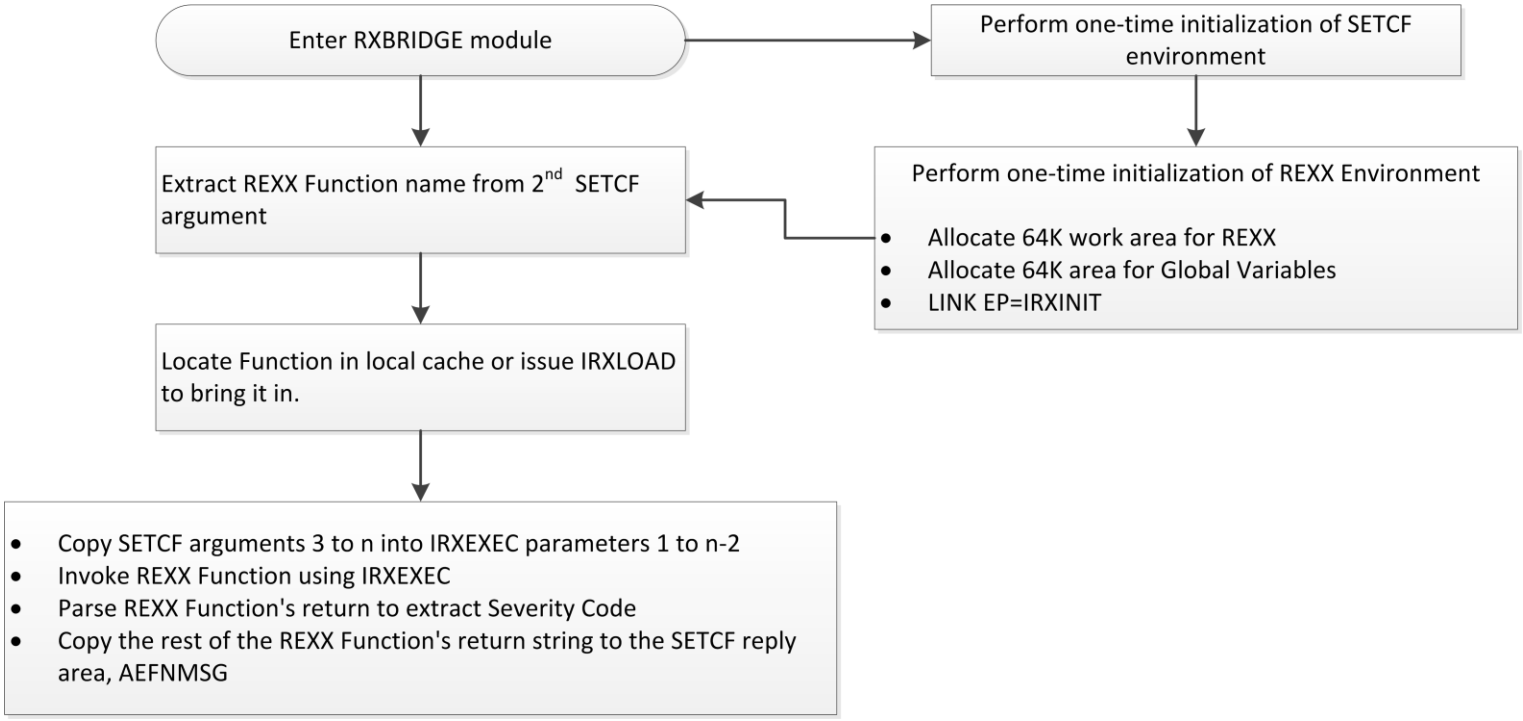
- You can use the REXX Stack to communicate between RXBRIDGE Exec calls.
  - PUSH (LIFO)
  - QUEUE (FIFO)
  - PULL
- I believe using Global variables is more flexible, but this works for simple interfaces and in some cases may be more convenient when dealing with unknown quantities of data.

6. Down to the Nitty-Gritty

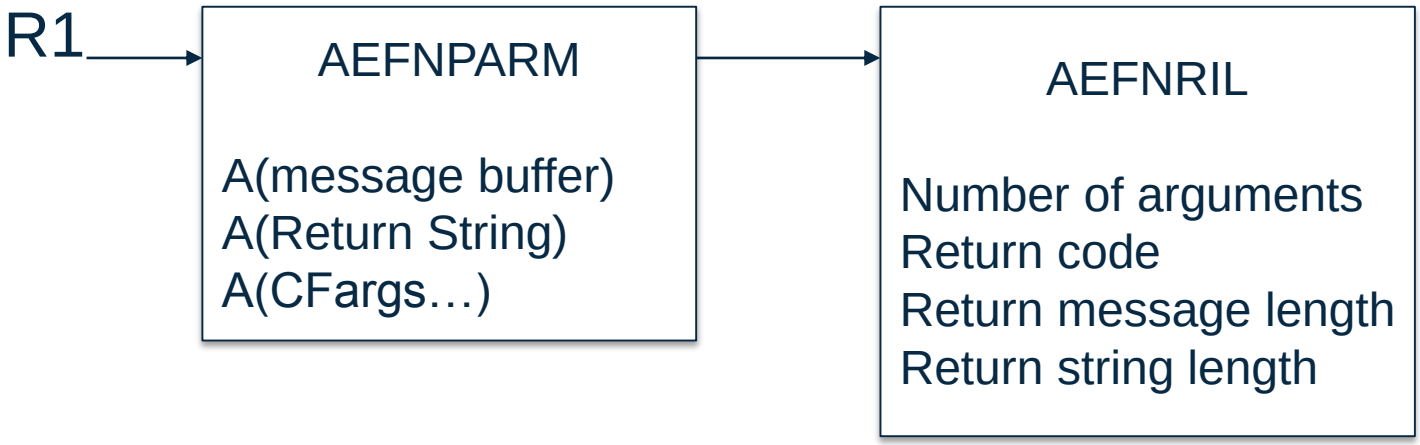
- Normal flow
- RXBRIDGE module inputs
- Initializing the REXX Environment - IRXINIT
- Invoking a Exec with IRXEXEC
- Outputs

Overview of flow under normal conditions

This flow chart shows the activities that occur within SETCF:



What the world looks like inside RXBRIDGE



The AEF control blocks are built by the assembler and passed to RXBRIDGE via R1. These DSECTs are mapped in HLA.SASMMAC1(ASMAEFNP).

Inputs to the routine

- AEFNPARM
  - AEFNRIP points to the AEFNRIL (below)
  - AEFNCF_PARMA[] array of pointers to the CFarg1, CFarg2, CFarg3,,, strings.
    - Number of elements is in AEFNUMBR.
- AEFNRIL
  - AEFNTYPE is 2 to indicate that this is a SETCF call
  - AEFN_PARMN_L[] array of lengths to the CFarg1, CFarg2, CFarg3,,, strings.
    - Number of elements is in AEFNUMBR.
    - Max argument length is 1024 bytes.

After the link to IRXINIT

```
TCB#5 RB#1 ------------------------------------XDC-CDF ISPF INTERFACE ------------ A.S. ROBRXTST
XDC ===> FORMAT .IRXINIT_PARMS
_  00000000_1F2117B8 8f (A.S.ROBRXTST) --- RXBRIDGE.RXBRIDGE+C68, @R12+C68, @R13+C68, RXBRIDGE+C68,
_           XPRIVATE+9117B8
_     .+C68                    IRXINIT_PARMS  DS    0D
_     .+C68 1F2117D4                          DC    A(IRXINIT_PARM1)
_     .+C6C 1F2117DC                          DC    A(IRXINIT_PARM2)
_     .+C70 1F2117E4                          DC    A(IRXINIT_PARM3)
_     .+C74 1F2117E8                          DC    A(IRXINIT_PARM4)
_     .+C78 1F2117EC                          DC    A(IRXINIT_PARM5)
_     .+C7C 1F2117F0                          DC    A(IRXINIT_PARM6)
_     .+C80 9F2117F4                          DC    A(X'80000000'+IRXINIT_PARM7)
_     .+C84 C9D5C9E3 C5D5E5C2  IRXINIT_PARM1  DC    CL8'INITENVB' CREATE A NEW ENVIRONMENT
_     .+C8C 40404040 40404040  IRXINIT_PARM2  DC    CL8' '        DEFAULT PARAMETER MODULE
_     .+C94 1F213D78           IRXINIT_PARM3  DC    V(IRXPARMS)   IN-STORE PARM LIST
_     .+C98 00000000           IRXINIT_PARM4  DC    A(0)          NO USER FIELD
_     .+C9C 00000000           IRXINIT_PARM5  DC    F'0'          RESERVED
_     .+CA0 0001BC90           IRXINIT_PARM6  DC    A(*-*)        ADDRESS OF ENVIRONMENT BLOCK
_     .+CA0                    ENVADDR        EQU   IRXINIT_PARM6,L'IRXINIT_PARM6
_     .+CA4 00000000           IRXINIT_PARM7  DC    A(*-*)        RETURN REASON CODE
```

We can see that we have been supplied with an Environment Block

Prior to the call to IRXEXEC

These are the parameters before the call to IRXEXEC

```
TCB#5 RB#1 ------------------------------------XDC-CDF ISPF INTERFACE --------------------------
XDC ===> FORMAT R1?
_  00000000_1F211878 8f (A.S.ROBRXTST) --- RXBRIDGE.RXBRIDGE+D28, @R1+0, @R15+9E8, @R9+ADA,
_           @R12+D28, @R13+D28, RXBRIDGE+D28, XPRIVATE+911878
_     .+D28                     IRXEXEC_PARMS  DS    0D
_     .+D28 1F2118A0                           DC    A(IRXEXEC_PARM1)
_     .+D28 1F2118A0          o@R1                .IRXEXEC_PARM1   *....*
_     .+D2C 1F2118A4                           DC    A(IRXEXEC_PARM2)
_     .+D30 1F2118A8                           DC    A(IRXEXEC_PARM3)
_     .+D34 1F2118AC                           DC    A(IRXEXEC_PARM4)
_     .+D38 1F2118B0                           DC    A(IRXEXEC_PARM5)
_     .+D3C 1F2118B4                           DC    A(IRXEXEC_PARM6)
_     .+D40 1F2118B8                           DC    A(IRXEXEC_PARM7)
_     .+D44 1F2118BC                           DC    A(IRXEXEC_PARM8)
_     .+D48 1F2118C0                           DC    A(IRXEXEC_PARM9)
_     .+D4C 9F2118C4                           DC    A(X'80000000'+IRXEXEC_PARM10)
_     .+D50 1F2119C8           IRXEXEC_PARM1   DC    A(EXECBLK)     EXECBLK GIVEN
_     .+D54 1F211168           IRXEXEC_PARM2   DC    A(ARGLIST)     ADDRESS OF ARG LIST
_     .+D58 50000000           IRXEXEC_PARM3   DC    B'01010000',XL3'0'
_     .+D58                    *                       ^^^^
_     .+D58                    *                       |||+------- RETURN EXTENDED RETCODES
_     .+D58                    *                       ||+-------- SUBROUTINE CALL        (NO)
_     .+D58                    *                       |+--------- EXTERNAL FUNCTION CALL (YES)
_     .+D58                    *                       +---------- NOT A COMMAND
_     .+D5C 1F2145F0           IRXEXEC_PARM4   DC    A(0)           INSTBLK ADDRESS (0=NOT PRELOADED)
_     .+D60 00000000           IRXEXEC_PARM5   DC    A(0)           NO CPPL SINCE BATCH MODE
_     .+D64 1F211A08           IRXEXEC_PARM6   DC    A(EVALBLOK)    ADDRESS OF EVAL BLOCK
_     .+D68 1F21193C           IRXEXEC_PARM7   DC    A(WORKAREA)    WORK AREA PROVIDED
_     .+D6C 00000000           IRXEXEC_PARM8   DC    A(0)           NO USER FIELD
_     .+D70 00000000           IRXEXEC_PARM9   DC    A(0)           ENVBLOCK IN R0
_     .+D74 00000000           IRXEXEC_PARM10  DC    F'0'           RETURN CODE AREA
```

Prior to the call to IRXEXEC

The IRXEXEC 2nd parameter points to the arguments (one in this case) being passed to the Exec

```
TCB#5 RB#1 ------------------------------------XDC-CDF ISPF INTERFACE --------------------------
 XDC ===> FORMAT .IRXEXEC_PARM2? 5
 _  00000000_1F211168 8f (A.S.ROBRXTST) --- RXBRIDGE.RXBRIDGE+618, @R9+3CA, @R12+618, @R13+618,
 _           RXBRIDGE+618, XPRIVATE+911168
 _     .+618 1F286D48 0000000E  ARGLIST  DS    (200+1)D            ROOM FOR UP TO &MAXARGS ARG
 _     .+618                    ARG_CLEAR_LEN EQU *-ARGLIST
 _     .+620 FFFFFFFF FFFFFFFF o                                   *........*
 _     .+628 00000000 00000000 o                                   *........*
```

And here we see the argument that is being passed…

```
TCB#5 RB#1 ------------------------------------XDC-CDF ISPF INTERFACE ------
XDC ===> DISPLAY .IRXEXEC_PARM2?? 5
_  00000000_1F286D48 8f (A.S.ROBRXTST) --- , @R7+828, @R6+848, XPRIVATE+986D48
_  1F286D48  8f                    E38889A2 4089A240          *This is *
_  1F286D50  8f  A385A7A3 40F10000 00000000 00000000  *text 1.........*
```

When IRXEXEC returns

We can see the RETURN string from the Exec:

```
XDC ===> DISPLAY .EVDATA 5
_  00000000_1F211A18 8f (A.S.ROBRXTST) --- RXBRIDGE.RXBRIDGE+EC8, @R1+1A0, @R9+C7A, @R12+EC8,
_           @R13+EC8, RXBRIDGE+EC8, XPRIVATE+911A18
_     .+EC8  8f                   F040E388 89A24089          *0 This i*
_     .+ED0  8f  A240A385 A7A340F7 40E38889 A24089A2  *s text 7 This is*
_     .+EE0  8f  40A385A7 A340F840 40404040 40404040  * text 8         *
```

and we convert that to the SETCF Severity Code and message

Outputs from the routine

- When we receive RC = 0 from the Return we
  - Set AEFNMSGS (severity) = 0
  - Return string to assembler by copying into buffer supplied in AEFNCF_SA word.
- When we receive RC > 0 from the Return we
  - Set AEFNMSGS (severity) = rc
    - Sets assembly return code, typically 4,8,or 12
  - Return message to assembler by copying into buffer supplied in AEFNMSGA word.

7. Some other possibilities

- You can do most anything that REXX can do.
- Write any data to //SYSTSPRT with SAY
- Open an FTP Socket
- Issue an MVS command (if authorized)
- Access z/OS or HFS datasets during macro processing
- Perform complex text editing
- Look up records in a database

Summary

- We have shown that REXX Execs can be invoked during the assembly process by using the SETCF assembler statement.
- You can write nearly any Exec that you wish and all the REXX built-in functions are available.
- You can communicate among Execs using GLOBAL_ variables.
- You can write any data you wish to the //SYSTSPRT dataset.
- You can modify the program to add more features if you desire. This is just a framework for the basics.

Full Source Code

You can download the full source code for RXBRIDGE from

http://www.colesoft.com/SHARE-March2015

You will be asked to agree to the usual disclaimers, etc.

References

SA32-0972 TSO/E REXX Reference

SC26-4940 High Level Assembler Language Reference
Chapter 9 – How to write conditional assembly instructions

SC26-4941 High Level Assembler Programmer's Guide
Chapter 5 – Providing External Functions

Proceedings of the REXX Symposium for Developers and Users, 1992,
Page 231, Interfacing with REXX, Anthony Rudd

Questions?