Thank you for your interest in this topic. I hope to provide you with a 40 minute informative and interesting presentation and at the end of the presentation we will have a Q&A period. If you have questions during the presentation please raise your hand to be recognized.

If you have more in-depth questions after the presentation please feel free to contact me at the E-Mail address below.

Lets begin…

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.

[click to advance to each topic]

Introduction

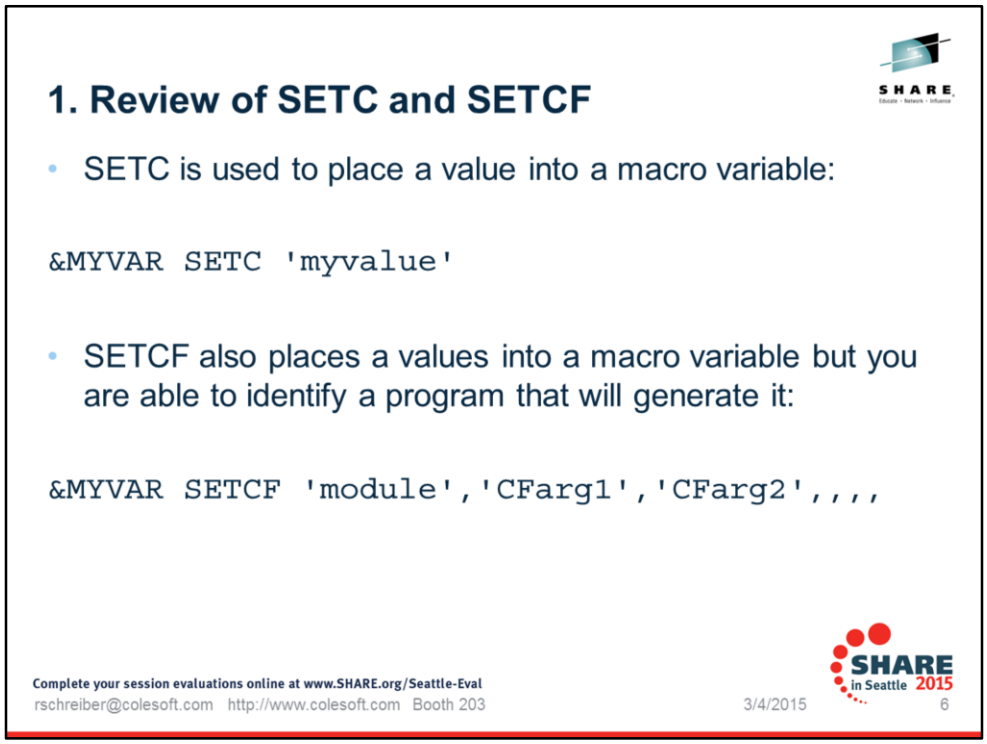
Here are a few examples of why we might want to do this.

- 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

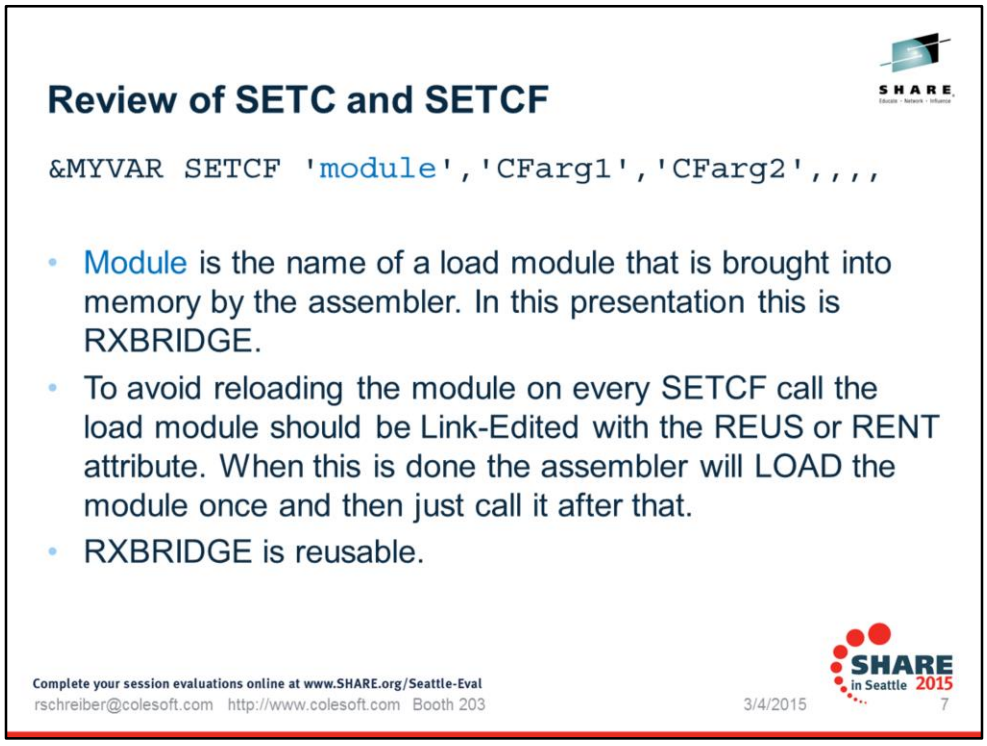
[click to advance to each topic]



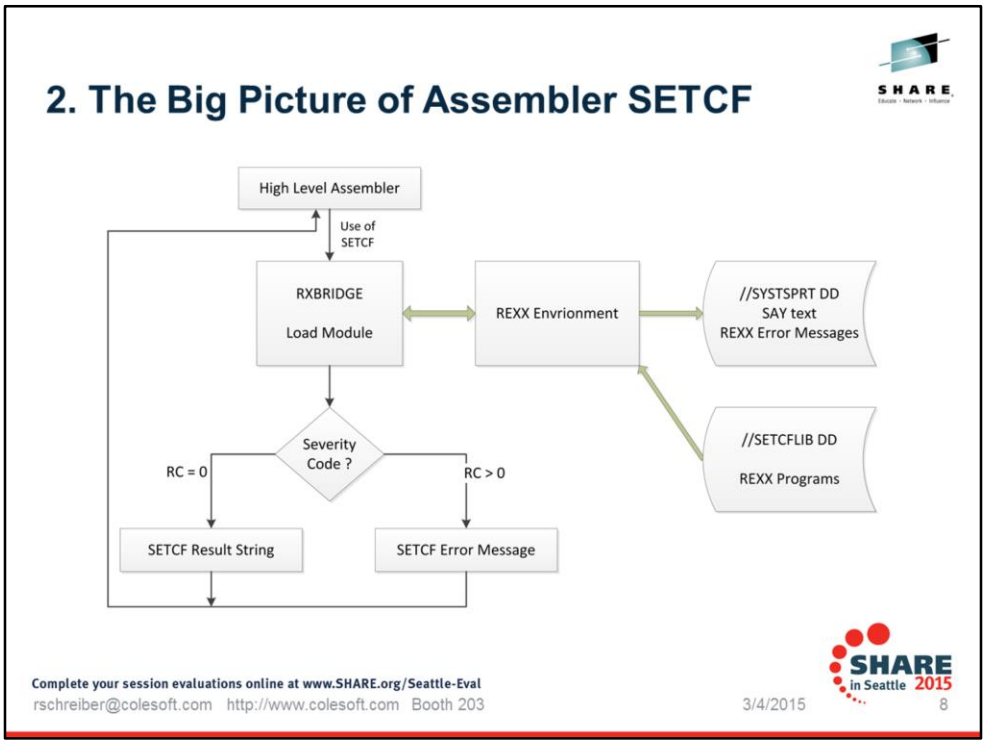
The first example is the SETC that we all know and love.

I suspect that not many people have used SETCF because it is a bit intimidating.

Not to worry.



Using REUS / RENT will have a dramatic effect on performance.



This diagram shows the overall relationship among the Assembler, the SETCF module, and the REXX Environment. We will be breaking down these pieces in the minutes ahead.

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.

The code sample has a limit to 20 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

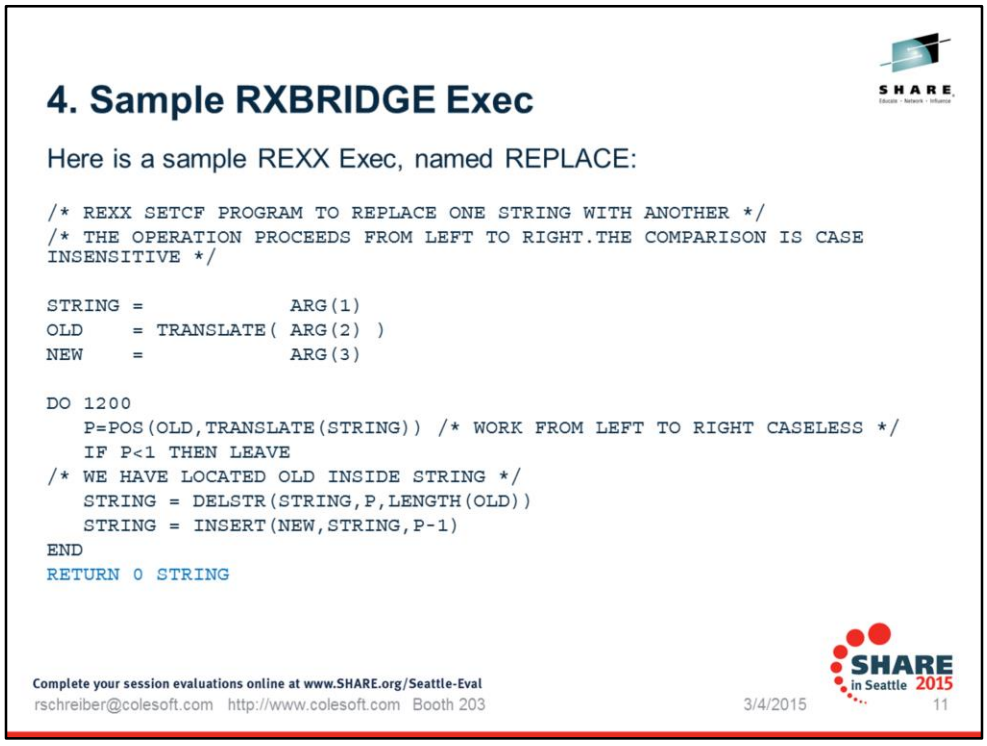
```
//SETCFLIB DD DSN=myfavs.rexx,DISP=SHR
```

SYSTSPRT contains SAY and REXX error messages.

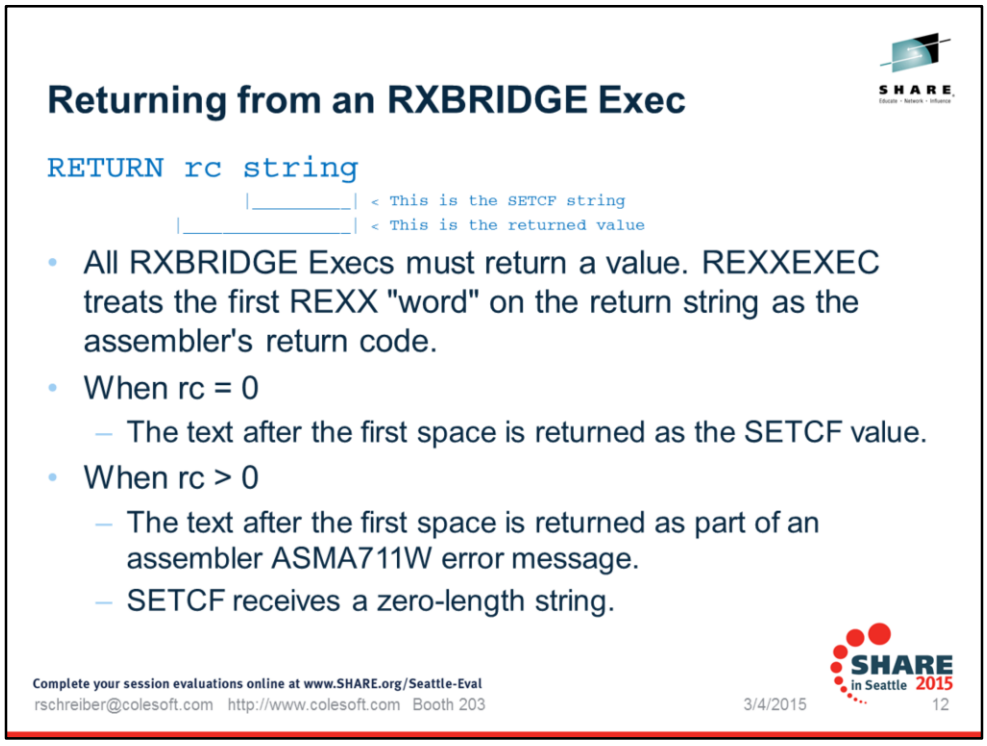
```
//SYSTSPRT DD SYSOUT=*
```

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

STEPLIB is optional. The RXBRIDGE module could be in the LINK PACK AREA or other locations.



This simple Exec does not check for the pathological case where the old and new intersect.
The point here is to show how arguments come in and a return string is formed.
A more robust implementation would make such a check and issue RETURN 4 or take some other action.



A REXX word is a blank delimited string of characters.
A return code may have leading zeros.
If you have something like RETURN 0 ' SOME TEXT' then the SETCF will return ' SOME TEXT'.
Note that leading and trailing blanks ARE significant in the returned string.
Return '0 Some Text' is just fine too.

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'
```


Here is some sample assembler code that invokes the REXX REPLACE Exec.

The use of DEQUOTE is required since the syntax of SETCF MUST have single quotes on the arguments.

&DATA SETCF 'RXBRIDGE','REPLACE',&STRING,&OLD,&NEW will cause an error since this does not act like
  DC C&STRING

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.

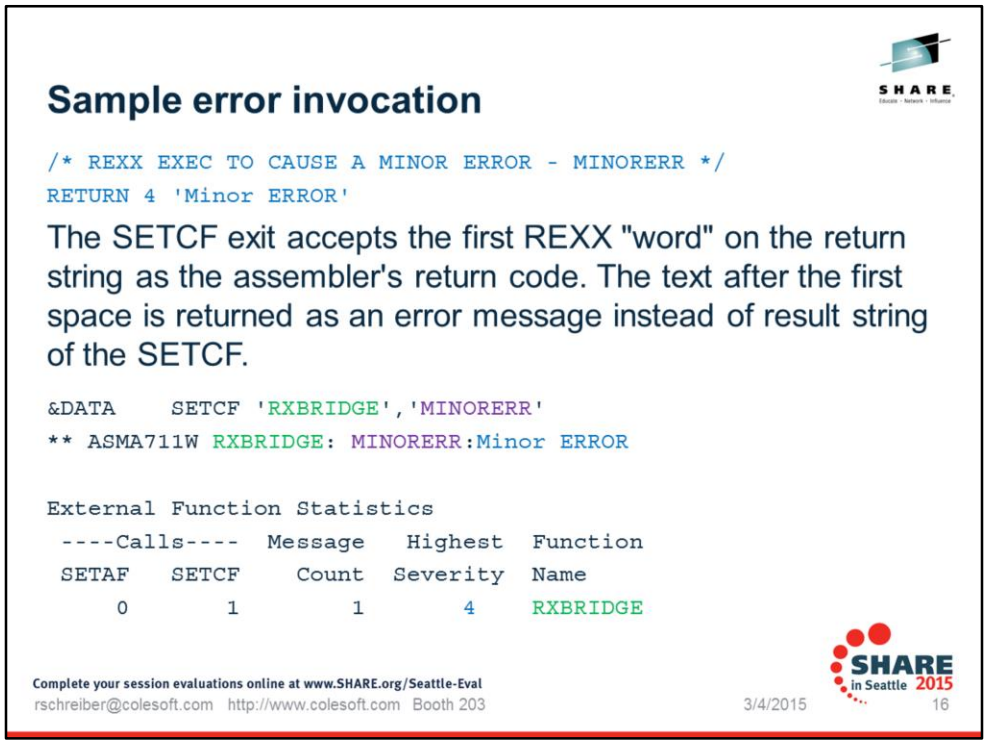
Can I use SAY in a Exec?
SURE! The result of SAY is sent to //SYSTSPRT

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
```



The ASMA711W message occurs in SYSPRINT, sort of like an MNOTE.

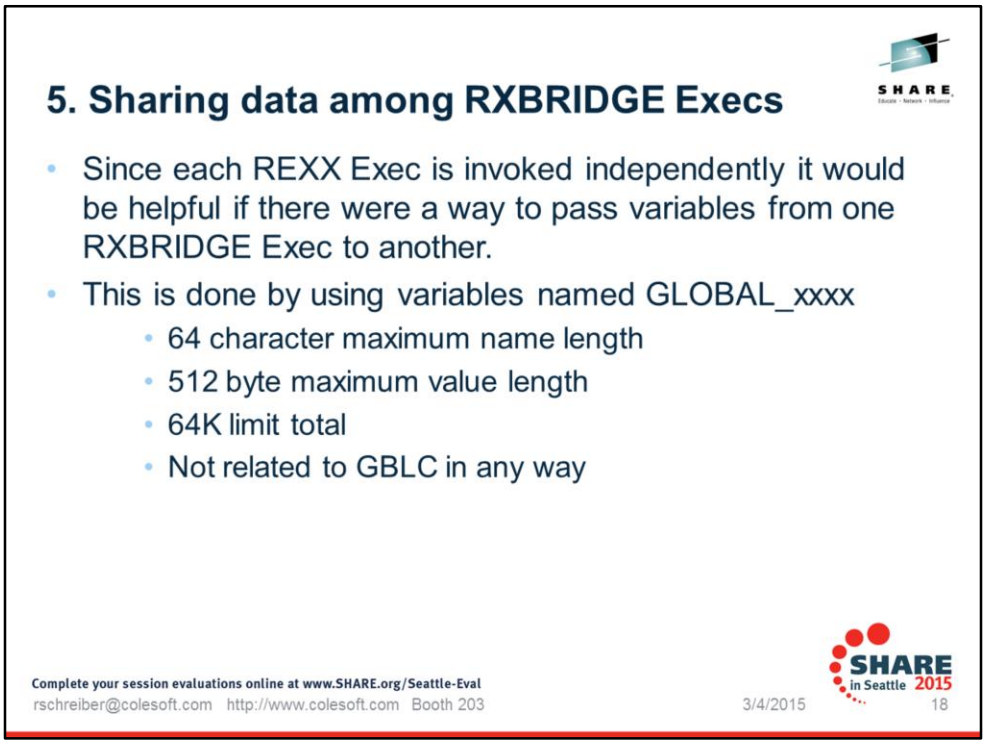
The ultimate return code of the assembly is the highest severity that was seen.

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) +++
```



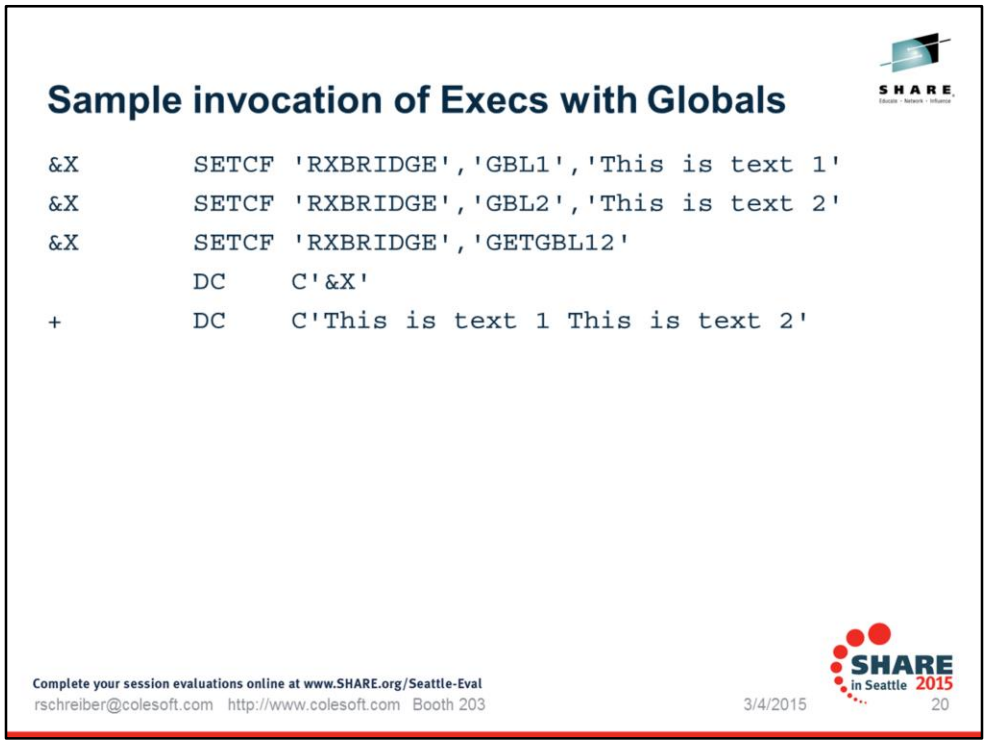
These are arbitrary limits. Small changes to the source code can accommodate different limits.

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
```



Note that there is a space between "…text 1" and "This is…".
That is because the Return statement in GETGBL12 put it there.
If you wanted the two string to be "abutted" then you would code
RETURN 0 GLOBAL_ONE || GLOBAL_TWO
In the GETGBL12 Exec.

Using PARSE with RXBRIDGE

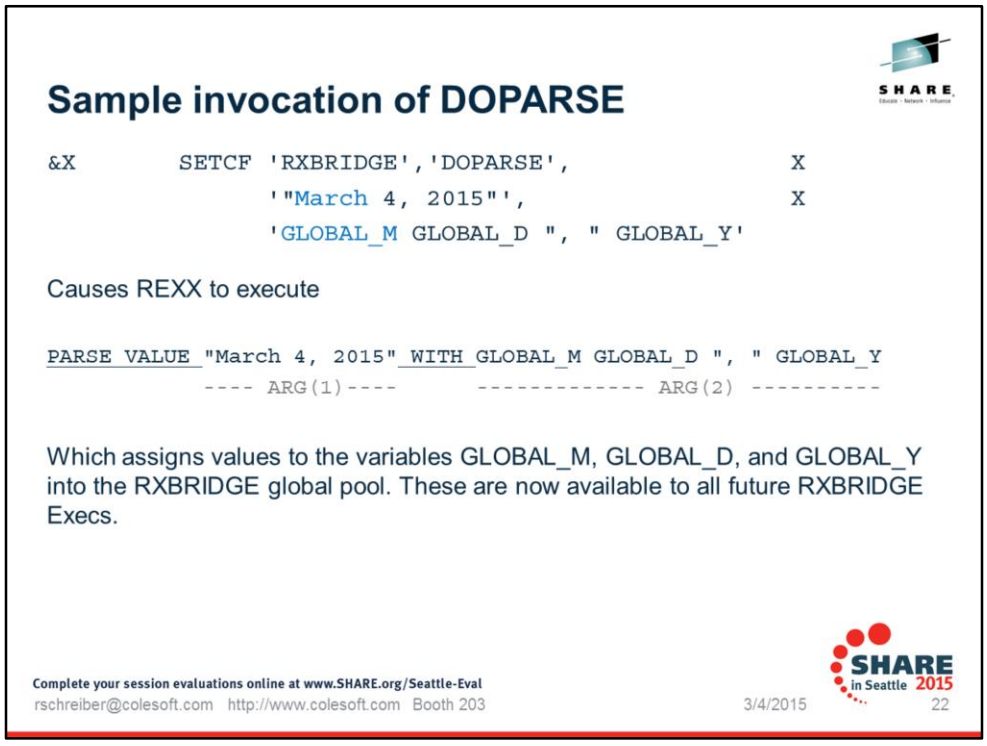
By using a pair of RXBRIDGE Execs we can access the REXX PARSE instruction with assembly-provided values and templates:

```
/* 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.



Note that the quotes are required on the value due to REXX syntactical requirements. Watch out when the value and template need to contain single quotes.

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.

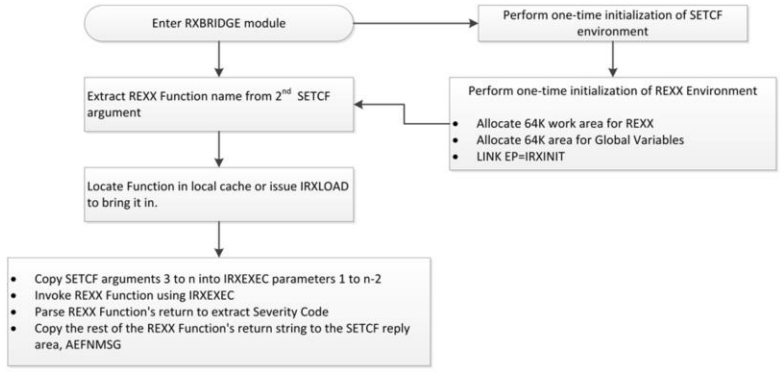
6. Down to the Nitty-Gritty

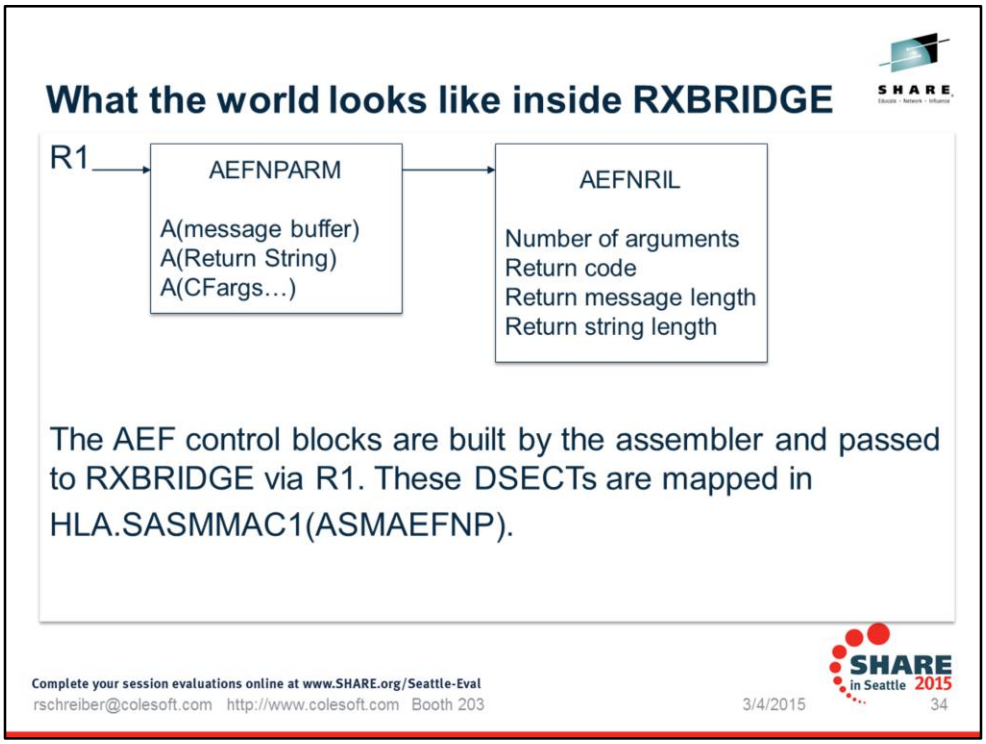
- Normal flow
- RXBRIDGE module inputs
- Initializing the REXX Environment - IRXINIT
- Invoking a Exec with IRXEXEC
- Outputs

OK. So how do we accomplish this?

Overview of flow under normal conditions

This flow chart shows the activities that occur within SETCF:

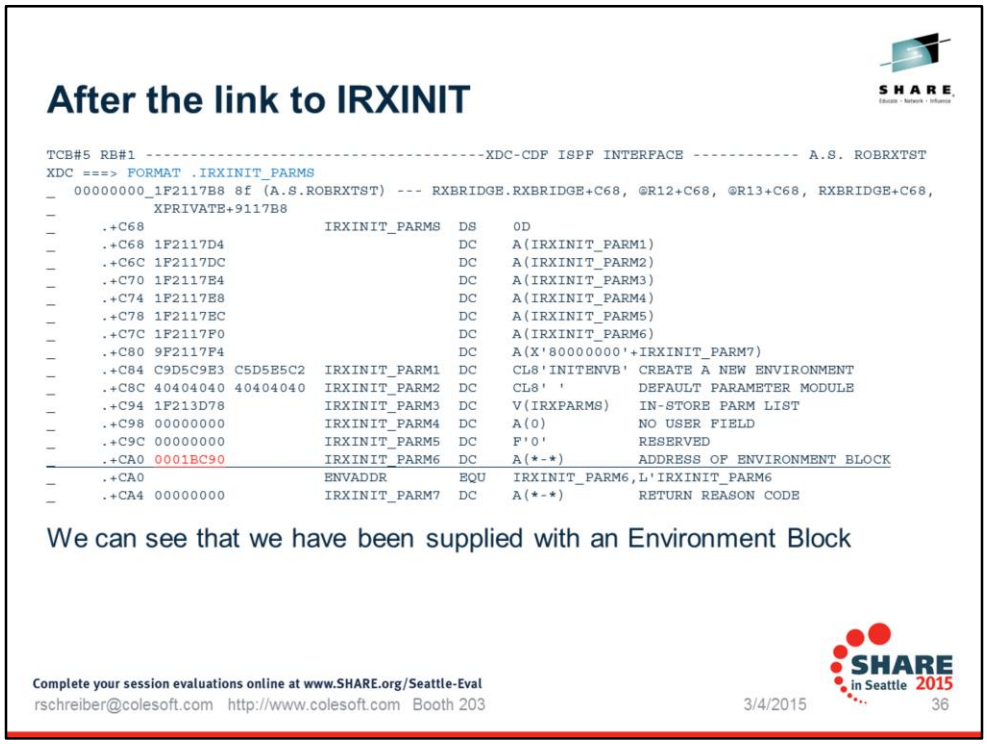




Upon entry to the SETCF module, R1 points to the AEFNPARM control block.
The AEFNRIP pointer points to the AEFNRIL.
The dataset name may be different on your system.

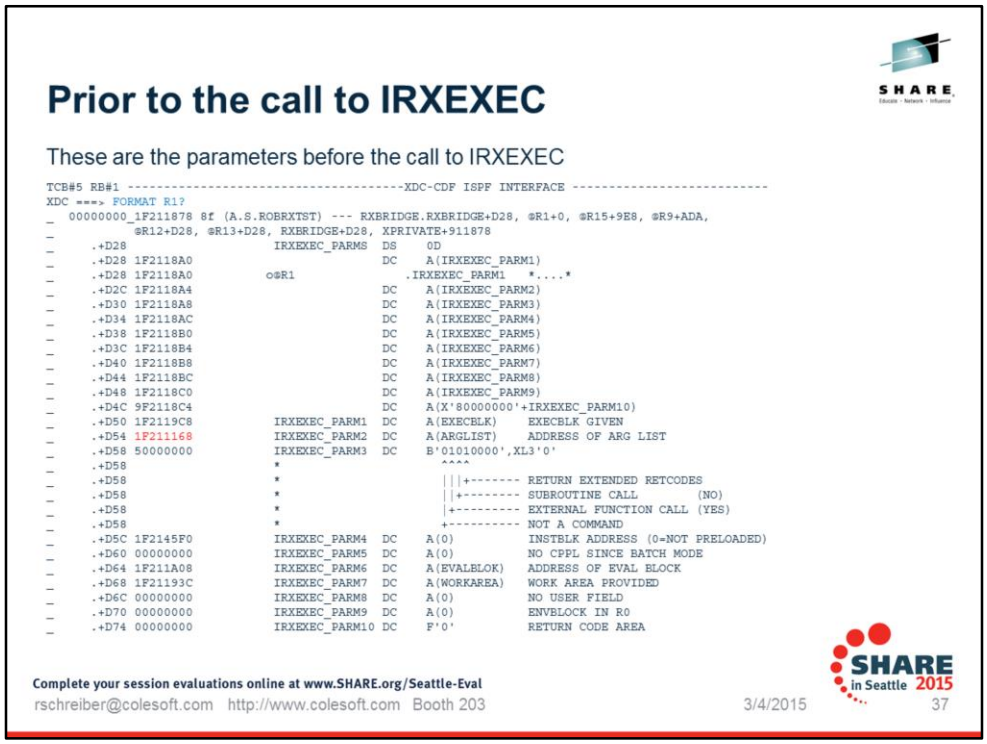
Inputs to the routine

- 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.

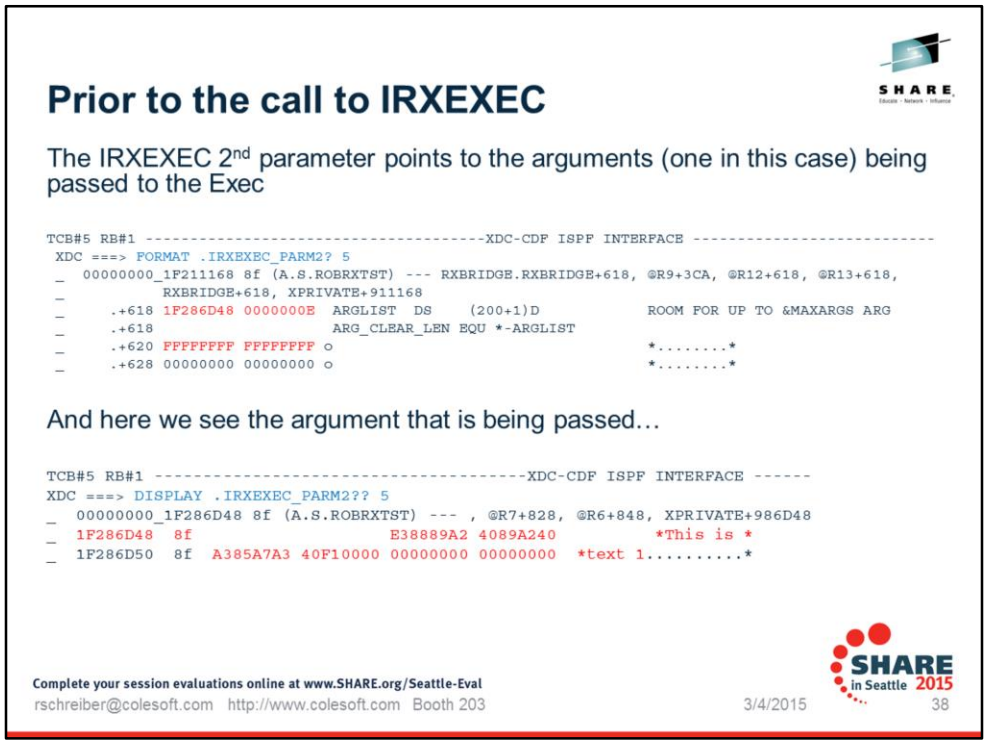


This is a mapped display of live storage as it existed while the program was running.

Here we can see that REXX has provided an Environment Block. This is required for all the other calls we make to REXX.



Parameter 2 is a list of arguments to the REXX Exec.
The list has ADDRESS and LENGTH and is terminated with a picket fence.



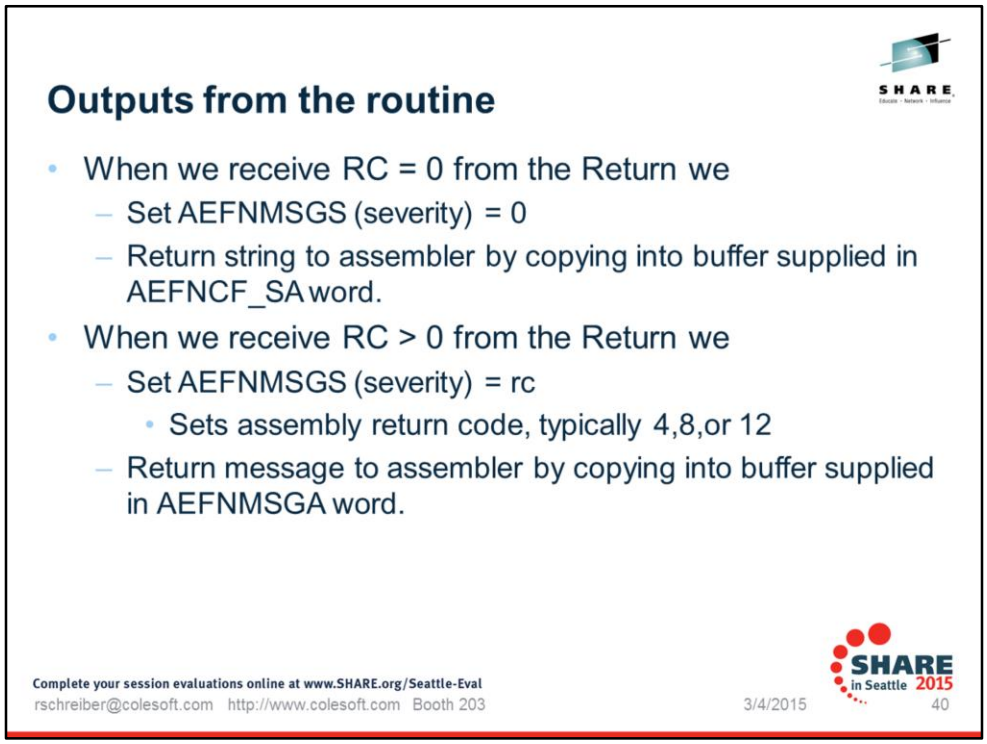
Here we can see that there is one parameter followed by the Picket Fence.
The first entry has the address and length of the RXarg1 being passed in to the Exec.

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



AEFNMSGS is the return code that is used by the assembler. Think of like an MNOTE n,'asd' statement.
Message may be up to 240 bytes since we use some of them as a leader.

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?