

# An introduction to using REXX from High Level Assembler

Session 7691

Barry Lichtenstein  
IBM Poughkeepsie  
Barry.Lichtenstein@us.ibm.com



SHARE in Boston – August 2010  
Copyright International Business Machines Corporation 2010

**SHARE** in Boston

# Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- IBM\*
- z/OS\*
- OS/390\*
- Language Environment\*
- S/360
- MVS
- z/Architecture

\* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

\* All other products may be trademarks or registered trademarks of their respective companies.

## Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

# Why High Level Assembler?

- Really *any* High Level language...
  - Register parms also in parm lists
  - Special (short) alternate entry point names for Fortran
- but...

## Why High Level Assembler? ...

- If you're a REXX programmer
  - There is a lot you can do in Assembler that you cannot in REXX
    - Deal with registers, SVCs
    - Add functions and function packages
    - Preload execs
    - Replace or extend some REXX native functionality such as I/O

## Why High Level Assembler? ...

- If you're an Assembler programmer
  - Most HLLs have run-times
    - REXX can be a powerful and easily extensible addition
    - Could be useful even just for prototyping

## Why High Level Assembler? ...

- A bunch of assembler macros (many to be covered later) in SYS1.MACLIB:
  - IRXARGTB – Argument Table
  - IRXCMPTB – Compiler Programming Table
  - IRXDSIB – Data Set Information Block
  - IRXEFPL – External Function Parameter List
  - IRXENVB – Environment Block
  - IRXEVALB – Evaluation Block
  - IRXEXECB – Exec Block
  - IRXEXTE – Vector of External Entry Points
  - IRXFPDIR – Function Package Directory
  - IRXINSTB – In-Storage Block
  - IRXMODNT – Module Name Table
  - IRXPACKT – Function Package Table
  - IRXPARMB – Parameter Table
  - IRXSHVB – Shared Variable Request Block
  - IRXSUBCT – Subcommand Table
  - IRXWORKB – Work Block Extension

## Why High Level Assembler? ...

- z/OS (and z/VM) only
  - Only z/OS described here!
  - Not in ooRexx<sup>®</sup> etc.
    - ooRexx has C extensible APIs
      - *Some similar capabilities*
        - *Like building external native libraries (usually DLLs)*

# What can you do?

- REXX to Assembler
  - Easy, just call as a “host” program!
    - Like *Address LINKMVS* ...
  - A little harder...
    - Write assembler as a REXX function or subroutine
      - *Return data, not just a return code*
  - A little harder still...
    - Use REXX programming services
      - *For example to share variables*



# What can you do? ...

- Assembler to REXX
  - Not too hard, CALL like any other program...
    - REXXC (REXX compiler) can create program modules
      - *Need optional product “IBM Compiler and Library for REXX”*
      - *Not just base element “Alternate Library for REXX” (no compiler)*
    - IRXJCL – invoke REXX exec from batch or program
      - *Single MVS style parameter string*
  - Harder, call as a REXX function or subroutine
    - IRXEXEC – invoke REXX exec from batch or program
      - *Pass multiple arguments*
      - *Preload execs*
      - *Return data, not just a return code*
        - *A “command” can only return a signed fullword number*

## What you can do? ...

- Services (like IRXEXEC, IRXEXCOM)...
- Parameter lists
  - Standard OS linkage
    - R1 points to a list of pointers to parameters
    - Last parameter is identified by the Hob
      - *On most calls, some parameters are optional*
    - standard R13, R14, R15
- Structures (“Blocks”)

# What you can do? ...

- Return Codes
  - R15, also return code parameter
  - *Not* returned to the REXX program!
    - REXX variables are (RC, RESULT)
  - **IRX0040I Error running exec\_name, line nn: Incorrect call to routine**

The language processor encountered an **incorrectly used call to a built-in or external routine**.

*You may have passed invalid data (arguments) to the routine. This is the most common possible cause and is dependent on the actual routine.*

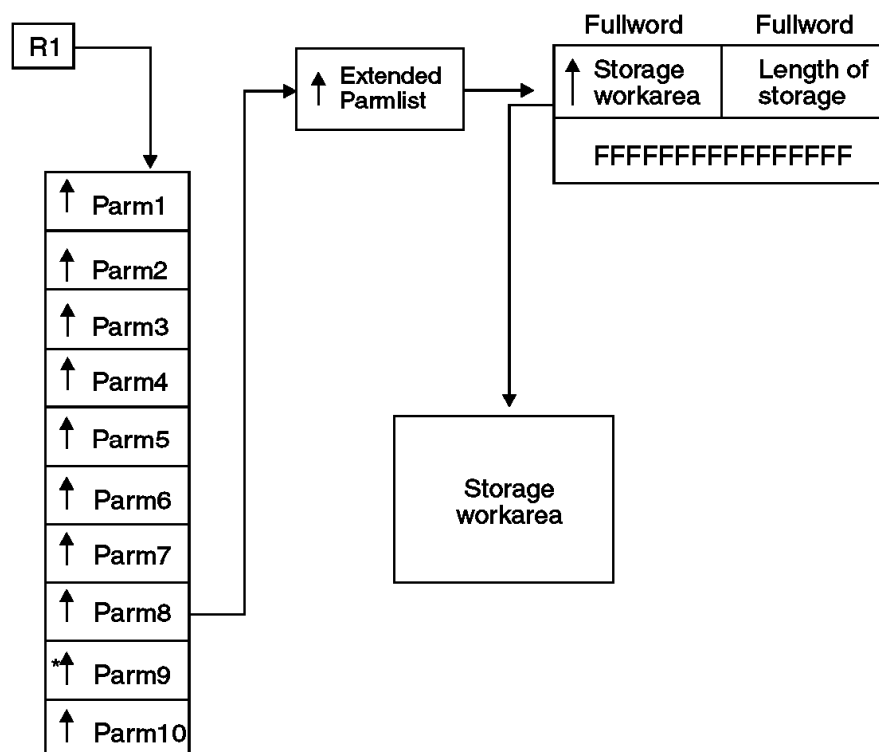
**If a routine returns a non-zero return code, the language processor issues this message and passes back its return code of 20040.**

# Initialization & Termination

- IRXINIT (IRXTERM) - Initialize (Terminate) a language processor environment.
- IRXINIT R1 parm list (of addresses of)...
  1. Function 8 characters
  2. Parameters module 8 characters  
and/or
  3. In-storage parameter list address
  4. User field address
  5. Reserved address, parameter must be 0
  6. Environment block address, output
    - Also in R0
  7. Reason code fullword, output
  8. Extended parameter list address, optional
    - Storage workarea; by default system obtained
    - Generally 3 pages (12K) of storage is needed for the storage workarea for normal exec processing, for each level of exec nesting.
  9. Return code fullword, output, optional
  10. TSO/E ECT address of address of, optional
    - Only for initializing TSO/E integrated environment

# Initialization ...

- IRXINIT...



\* high order bit on

## Initialization ...

- Precedence for initializing environment (parameters)
  - Each type can exist but have (some) null parameters
    - blanks or zeroes depending on type
  - 1. In-storage parameter list
  - 2. Parameters module
  - 3. Previous environment
  - 4. IRXPARMS default parameters module

## Initialization ...

- Provided parameter module tables
  - IRXPARMS – non-TSO/E
  - IRXTSPRM – TSO/E
  - IRXISPRM – ISPF

# Initialization ...

- IRXINIT... Function
  - INITENVB - initialize an environment
  - FINDENVB - find the current environment
  - CHECKENVB - validate a given address is an environment
    - R0 must point to an existing environment block (optional for other calls)



# Initialization ...

- Initialization normally not required
  - MVS, TSO/E, ISPF, z/OS UNIX automatically initialize for you
- Will initialize based on previous environment
  - Environments are chained
  - This allows you to create your own environment with select updates
    - Cannot be “integrated into TSO/E”
      - *Cannot use TSO/E commands, service routines such as IKJPARS and DAIR, or ISPF services or CLISTs*

# Initialization Parameters

- The format of the in-storage list is identical to the format of the parameters module.
  1. ID 8 characters
  2. Version 4 characters, "0200"
  3. Language 3 characters, "ENU"
  4. Reserved 1 byte
  5. MODNAMET address of Module Name Table
  6. SUBCOMTB address of Subcommand Table
  7. PACKTB address of Function Package Table
  8. PARSETOK 8 bytes, Parse Source token
  9. FLAGS fullword, environment flags
  10. MASKS fullword, FLAGS mask bits
  11. SUBPOOL fullword, Storage Allocation Subpool Number
  12. ADDRSPN fullword, Address Space Name
  13. End of Block doubleword of X'FF'

## Initialization Parameters ...

- MODNAMET (IRXMODNT) -- module name table
  - The DDs for reading and writing data
    - *SYSTSIN / SYSTSPRT*
  - The DD from which to load REXX execs
    - *SYSEXEC*
  - Replaceable routines
    - *Replace I/O (Say, EXECIO, etc), Stack, USERID()*
  - Several exit routines
    - *EXECINIT/EXECTERM – before/after language processing of exec*

# Initialization Parameters ...

- SUBCOMTB (IRXSUBCT) – subcommand table
  - “host” command environments
    - “address” subcommand names
      - *the environment to which the language processor passes commands for execution*
  - An “address” name
  - A corresponding processing routine

# Termination

- Pass environment pointer
- Same task
- LIFO
  
- Closes all data sets opened under that environment
- Deletes any data stacks (NEWSTACK)

# Updating the Subcommand Table

- IRXSUBCM
  - ADD
    - Add an entry to the subcommand table (ignoring duplicates)
  - DELETE
    - Delete the last occurrence from the table
  - UPDATE
    - Update the values for the last occurrence of an entry (Routine, Token)
  - QUERY
    - Query the values of the last occurrence of an entry

# Structures

- Environment Block (IRXENVB, ENVBLOCK)
  - Address in R0 when external function or subroutine gets control
  - Required for all services (still optional, current will be found if not provided)
    - Unless it's reentrant
  - Contains...
    - Parameter Block (IRXPARMB, PARMBLOCK)
    - Vector of External Entry Points (IRXETE)
      - *REXX routines*
      - *System / User replaceable routines*
      - *You might like IRXSAY, IRXLOAD, etc.*
  - You can initialize more than one and run (REXX) in any particular one
    - by passing that environment block address

# Structures ...

- Example SUBCOM Table in UNIX

1	name=MVS	routine=IRXSTAM	token=	.
2	name=LINK	routine=IRXSTAM	token=	.
3	name=ATTACH	routine=IRXSTAM	token=	.
4	name=CPICOMM	routine=IRXAPPC	token=	.
5	name=LU62	routine=IRXAPPC	token=	.
6	name=LINKMVS	routine=IRXSTAMP	token=	.
7	name=LINKPGM	routine=IRXSTAMP	token=	.
8	name=ATTCHMVS	routine=IRXSTAMP	token=	.
9	name=ATTCHPGM	routine=IRXSTAMP	token=	.
10	name=APPCMVS	routine=IRXAPPC	token=	.
11	name=SYSCALL	routine=BPXWREXX	token=	.
12	name=MVS	routine=IRXSTAM	token=	.
13	name=LINK	routine=IRXSTAM	token=	.
14	name=ATTACH	routine=IRXSTAM	token=	.
15	name=CPICOMM	routine=IRXAPPC	token=	.
16	name=LU62	routine=IRXAPPC	token=	.
17	name=LINKMVS	routine=IRXSTAMP	token=	.
18	name=LINKPGM	routine=IRXSTAMP	token=	.
19	name=ATTCHMVS	routine=IRXSTAMP	token=	.
20	name=ATTCHPGM	routine=IRXSTAMP	token=	.
21	name=APPCMVS	routine=IRXAPPC	token=	.
22	name=SYSCALL	routine=BPXWREXX	token=	.
23	name=SYSCALL	routine=BPXWREXX	token=	.
24	name=SH	routine=BPXWRKSH	token=	.
25	name=TSO	routine=BPXWRADT	token=	.



# Structures ...

- Subcommand Table Block (IRXSUBCT)
  - Previous output from small assembler program called as function from REXX program
    - REXX passed ENVBLOCK address in R0 when external function or subroutine gets control
    - Parameter block contains SUBCOMTB address
    - Assembler subroutine passes SUBCOMTB back to REXX program
      - *REXX factoid: The only difference between functions and subroutines is that functions must return data, while subroutines may return data*

# Structures ...

- Subcommand Table Block (IRXSUBCT) ...

- Table header

- ADDRESS fullword address of first entry (row) in table
- TOTAL fullword # of entries in table (used & unused)
- USED fullword # of used entries
- LENGTH fullword length of each entry (always 32)
- INITIAL fullword address of name of host command environment (only if not passed on IRXEXEC)
- reserved doubleword
- End of Table doubleword of X'FF'

- Array of entries (rows)

- NAME 8 characters
- ROUTINE 8 characters
- TOKEN 16 characters, passed to ROUTINE when called
- ...

## Structures ...

- External Function Parameter List (IRXEFPL)
  - REXX passes EFPL address in R1 when external function or subroutine gets control
  - 5<sup>th</sup> word points to the Argument Table
    - Parsed arguments
  - 6<sup>th</sup> word points to the Evaluation Block
    - For returning data
    - Preset size

# Passing and Returning Arguments

- Argument Table (IRXARGTB)
  - Argument lists can be passed on IRXEXEC call
  - Same arguments/format received by any function/subroutine
- An array of fullword pairs
  - Argument address
  - Argument length
- Terminated with a doubleword of X'FF'.

# Passing and Returning Arguments ...

- Evaluation Block (IRXEVALB, EVALBLOCK)
  - When REXX calls a function / subroutine
    - It is allocated for you with a fixed size
      - *TSO/E provides 250 bytes for your returned data*
  - If you have coded (assembler) function / subroutine
    - You must create a larger block if necessary (using IRXRLT)
- Same format used by IRXEXEC
  - For returning from a REXX function / subroutine

# Example 1



## Return Subcommand Table block in the Evaluation Block

```
SUBR1      CSECT
. . .
*
      LR      r3,r0          Save ENVblock
      USING  ENVBLOCK,r3
      LR      r4,r1          Save EFPL
      USING  EFPL,r4
*
      L       r5,EFPLEVAL    ptr to addr of...
      L       r5,0(r5)      .. evalblock
      USING  EVALBLOCK,r5
*
      L       r6,ENVBLOCK_PARMBLOCK
      USING  PARMBLOCK,r6
      L       r6,PARMBLOCK_SUBCOMTB
      USING  SUBCOMTB_HEADER,r6
*
      MVC     EVALBLOCK_EVLEN,=F'24'
      MVC     EVALBLOCK_EVDATA(4),EVALBLOCK_EVSIZE
      MVC     EVALBLOCK_EVDATA+4(20),SUBCOMTB_HEADER
*
. . .
RET      LHI    r15,0
. . .
      BR      r14          RETURN TO CALLER
```

```
...
**
      YREGS
**
efpl0    IRXENVB
efpl1    IRXEFPL
evalb    IRXEVALB
parmb    IRXPARMB
subcomt  IRXSUBCT
**
      END
```

## Loading the exec

- IRXEXEC runs the exec which is ...
  - Preloaded with IRXLOAD or user replaceable routine
  - In-Storage Control Block (IRXINSTB, INSTBLK)
    - header
    - array of REXX record/length pairs
- -- or --
- Loaded by building an Exec Block (IRXEXECB, EXECBLK)
  - Member
  - DDNAME (default is SYSEXEC from module name table)
  - DSNptr
    - *for Parse Source*
  - Initial SUBCOM environment
  - Extended execname
    - *Not used by IRXLOAD; could be a UNIX pathname*

# Sharing Variables

- IRXEXCOM – REXX exec communication
  - 4<sup>th</sup> parameter points to ...
  - SHVBLOCK (IRXSHVB) – shared variable request block
    - SHVBLOCKS can be chained
    - assembler coded function / subroutine can get and set REXX variables



# Sharing Variables ...

- SHVBLOCK (IRXSHVB) – shared variable request block
  - SHVNEXT fullword chain pointer (0 if last block)
  - SHVUSER fullword user value  
except for "Next"
  - SHVCODE byte function code
  - SHVRET byte return code
  - reserved halfword, set to zero
- SHVBUFL fullword length of "Fetch" value buffer
- SHVNAMA fullword address of variable name
- SHVNAML fullword length of variable name (250 max)
- SHVVALA fullword address of value buffer
- SHVVALL fullword length of value  
set for "Fetch"

## Sharing Variables ...

- IRXEXCOM – REXX exec communication ...
  - SHVRET – Return Code Flags

SHVCLEAN	X'00'	Execution was OK
SHVNEWV	X'01'	Variable did not exist
SHVLVAR	X'02'	Last variable transferred (for "N")
SHVTRUNC	X'04'	Truncation occurred during "Fetch"
SHVBADN	X'08'	Invalid variable name
SHVBADV	X'10'	Value too long
SHVBADF	X'80'	Invalid function code (SHVCODE)

## Sharing Variables ...

- IRXEXCOM – REXX exec communication ...
  - Return Codes
    - -1 Insufficient storage
    - -2 Entry conditions not valid  
(like REXX exec not currently running)
    - 0 SUCCESS
    - 28 No environment found
    - 32 Invalid parameter list
    - nn Composite OR of SHVRETs  
(except SHVNEWV and SHVLVAR)

## Sharing Variables ...

- IRXEXCOM – REXX exec communication ...
  - Function code convention:
    - Direct interface (Uppercase):
      - *WYSIWYG*
      - *If b='Barry' then A.b is A.B*
    - Symbolic interface (Lowercase):
      - *Just like REXX does it*
      - *If b='Barry' then A.b is A.Barry*

## Sharing Variables ...

- IRXEXCOM – REXX exec communication ...
  - Function codes:
    - S/s – Set/Store (create)
    - F/f – Fetch
    - D/d – Drop
  
    - N – Fetch Next (exposed variables in generation)
    - P – fetch Private information (Arg, Source, Version)

# Example 2



## Returning variables from assembler to REXX

```
...
LR    r3,r0          Save ENVblock
USING ENVBLOCK,r3
***
shvr2 LA    r6,shvb2
USING SHVBLOCK,r6
MVC   shvr2.SHVNEXT,=F'0'
MVC   shvr2.SHVUSER,=F'0'
MVI   shvr2.SHVCODE,SHVSTORE
MVC   shvr2.SHVNAMA,=A(var2)
MVC   shvr2.SHVNAML,=A(evar2-var2)
MVC   shvr2.SHVVALA,=A(vvar2)
MVC   shvr2.SHVVALL,=A(evvar2-vvar2)
***
shvr1 LR    r0,r6
LA    r6,shvb1
USING SHVBLOCK,r6
ST    r0,shvr1.SHVNEXT
MVC   shvr1.SHVUSER,=F'0'
MVI   shvr1.SHVCODE,SHVSTORE
MVC   shvr1.SHVNAMA,=A(var1)
MVC   shvr1.SHVNAML,=A(evar1-var1)
MVC   shvr1.SHVVALA,=A(vvar1)
MVC   shvr1.SHVVALL,=A(evvar1-vvar1)
***

LA    r5,=CL8'IRXEXCOM'
ST    r5,parm1
*
LHI   r5,0
ST    r5,parm2
ST    r5,parm3
*
ST    r6,parm4
*
OI    parm4,X'80'
**
LR    r0,r3  restore ENVBLOCK 4 call!
LA    r1,plist
*
LINK  EP=IRXEXCOM
ST    r15,myret
...
BR    r14    RETURN TO CALLER
```

# Example 2 ...



## Returning variables from assembler to REXX ...

```
**
var1      DC      C'BARRY.Assembler'
evar1     EQU     *
*
vvar1     DC      C'doth he'
          DC      C' rexx codeth'
evvar1    EQU     *
*
var2      DC      C'RC'
evar2     EQU     *
*
vvar2     DC      C'1958'
evvar2    EQU     *
```

```
**
          YREGS
**
envb0     IRXENVB
evalb     IRXEVALB
parmb     IRXPARB
shrvar    IRXSHVB
**
MYAREA    DSECT
*
myret     DS      F
mysize    DS      F
**
plist     DS      0D
parm1     DS      F
parm2     DS      F
parm3     DS      F
parm4     DS      F
**
shvb1     ORG    *+SHVBLEN
shvb2     ORG    *+SHVBLEN
*
MYAREASZ  EQU    *-MYAREA
```

# Miscellany

- Using z/OS UNIX System Services
  - Environment created automatically when REXX program (***/\*REXX\**** “magic number) is *exec*’d.
    - BPXWRXEV parameters module
      - *Source in SYS1.SAMPLIB(BPXWRX01)*
    - Inherits default MVS REXX environment
    - I/O etc. overridden in MODNAMET table
    - Subcommand environments added in SUBCOMTB
      - *as we saw from example 1 earlier ...*
    - There is also a function package ..



## Miscellany ...

- Using z/OS UNIX System Services ...
  - BPXWRBLD
    - Create your own z/OS UNIX REXX environment
    - Sample C program in [z/OS Using REXX and z/OS UNIX System Services](#)

## Miscellany ...

- Using z/OS UNIX System Services ...
  - Other services available for assembler programmers
    - BPXWDYN – dynamic allocation (SVC 99) text string interface
    - bpxwunix() – run z/OS UNIX shell (/bin/sh)
      - *Run a shell script and/or other UNIX commands*

## Miscellany ...

- [z/OS TSO/E REXX Reference – SA22-7790](#)
- [z/OS Using REXX and z/OS UNIX System Services – SA22-7806](#)