

 #SHAREorg



An Introduction to Using REXX with Language Environment

Barry.Lichtenstein@us.ibm.com

August 2012
Session # 11538



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.

System z Social Media

- System z official Twitter handle:
 - [@ibm_system_z](#)
- Top Facebook pages related to System z:
 - [Systemz Mainframe](#)
 - [IBM System z on Campus](#)
 - [IBM Mainframe Professionals](#)
 - [Millennial Mainframer](#)
- Top LinkedIn Groups related to System z:
 - [Mainframe Experts Network](#)
 - [Mainframe](#)
 - [IBM Mainframe](#)
 - [System z Advocates](#)
 - [Cloud Mainframe Computing](#)
- YouTube
 - [IBM System z](#)



- Leading Blogs related to System z:
 - [Evangelizing Mainframe \(Destination z blog\)](#)
 - [Mainframe Performance Topics](#)
 - [Common Sense](#)
 - [Enterprise Class Innovation: System z perspectives](#)
 - [Mainframe](#)
 - [MainframeZone](#)
 - [Smarter Computing Blog](#)
 - [Millennial Mainframer](#)

Agenda

- Why Language Environment
- What can you do?
- Decisions, decisions
- Initialization (& Termination)
- Structures
- Passing and Returning Arguments
- Sharing Variables
- Miscellany

Why Language Environment?

- Really *any* language which produces program modules...
 - Register parms also in parm lists
 - Special (short) alternate entry point names for Fortran
- but...

Why Language Environment? ...

- If you're a REXX programmer
 - There is a lot you can do in an HLL that you cannot in REXX language
 - Deal with registers, SVCs
 - Add functions and function packages
 - Preload execs
 - Replace or extend some REXX native functionality such as I/O
 - With LE applications you can always bind in LE-conforming or LE-enabled High Level assembler!

Why Language Environment? ...

- If you're a C/C++ or COBOL or PL/I or assembler programmer
 - HLLs have run-times
 - REXX functions can be a powerful and easily extensible addition
 - *Could be useful even just for prototyping*

Why Language Environment? ...

- A bunch of assembler macros (many to be covered later) in 'SYS1.MACLIB':

- **IRXARGTB** – Argument Table
- **IRXCMPBTB** – 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

- All primarily mappings

Why Language Environment? ...

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

What can you do? REXX to Language Environment

- Starting in REXX
 - Calling Language Environment
- Starting in LE
 - Calling REXX
- Further Nesting ?
 - LE -> REXX -> LE ???

What can you do? ... REXX to Language Environment

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

What can you do? ... Language Environment 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; In storage 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
 - Language Environment HLLs support OS linkage
 - C use linkage(...,OS)
 - ...
- 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.

Decisions, decisions

- Tradeoffs
 - Time, complexity, isolation
- Need for REXX services ?
- Infrequently called ?

and / or
- Heavy-weight ?

Decisions, decisions ...

- Using Language Environment requires run-time initialization
 - Normally happens upon first program call from host
 - COBOL main program, C main, etc.
- LE Linkage Conventions
 - LE-conforming programs require LE and can use all services
 - LE-enabled applications follow similar OS-linkage conventions but cannot use all services

Decisions, decisions ... REXX to Language Environment

- Host program call using LE application
 - Like TSO or SH, so...
 - Host subcommand environment handles the call
 - Each call to Language Environment requires full LE initialization
 - Most isolated
 - No access to REXX services
 - Potentially slowest!

Decisions, decisions ... REXX to Language Environment

- REXX function or subroutine using LE application
 - C main(), COBOL main program, etc. so...
 - Still requires full LE initialization
 - Less isolated
 - Access to REXX services
 - Potentially faster...

Example 1 – HLLPIPIM2

```
/* REXX */  
Say "Before: LEREXX = <"LEREXX">"  
Call TIME 'Reset'  
Do pp=1 To cnt  
  Call HLLPIPIM2 "hi there","you all"  
End pp  
Say TIME('Elapsed')  
Say "After: LEREXX" LEREXX
```

As a REXX function, calling COBOL main program IGZPIPIM2:
Before: LEREXX = <LEREXX>
REXX RC IS 000000000, IRXEXCOM RC IS 000000000, SHVRET
IS 000000001.
REXX RC IS 000000000, IRXEXCOM RC IS 000000000, SHVRET
IS 000000000.
. . .
0.165410
After: LEREXX and COBOL all together!

Decisions, decisions ... REXX to Language Environment

- REXX function or subroutine using HLL sub-function
 - C function, COBOL subprogram, etc. so...
 - Something must still initialize the library!
 - Unless you're only interested in C...
 - METAL C
 - *Subset of C library functions*
 - System Programming C (SPC)
 - *Limited function C library support*

Decisions, decisions ... REXX to Language Environment

- System Programming C – SPC
 - Regular C compiles
 - No C++
 - No XPLINK, LP64, DLL (that all needs LE!)
 - Freestanding
 - @@XSTRT/@@XSTRL/@@XSTRX
 - *used by UNIX support of REXX syscalls*
 - persistent
 - @@XHOTC/@@XHOTL

Decisions, decisions ... REXX to Language Environment

- Preinitialization Services (PIPI)
 - Full LE support
 - Can call into main programs or subprograms
 - Requires some setup which has to be done in assembler
 - Or Metal C ...

Decisions, decisions ... REXX to Language Environment

- Preinitialization Services (PIPI) ...
 - With PIPI the environment is “resumed”
 - Careful of “Stop Semantics” which terminate enclave
 - *C exit(), COBOL STOP RUN, etc.*
 - So true subroutine can have static data
 - maintain a counter etc.
 - The subroutines must be known a priori in table
 - Loaded or provided on the INIT call
 - Added by an ADD_ENTRY call

Example 2 – ASMPIPI / ASMPIPC

```
/* REXX */  
  
Call ASMPIPI  
  
Say 'PIPIADDR (after) = 'C2X(PIPIADDR)'.'  
Say 'PIPITOKN (after) = 'C2X(PIPITOKN)'.'  
Say 'As a REXX function, calling an HLL sub' ,  
    'using PIPI CALL_SUB:'  
Call TIME 'Reset'  
Do pp=1 To cnt  
    Call ASMPIPC PIPIADDR, PIPITOKN, ,  
        "ASMPIPC ", ,  
        "As a REXX function via PIPI! call#" ,  
        pp "of" cnt  
End pp  
Say TIME('Elapsed')
```


Example 2 – ASMPIPI / ASMPIPC ...

```
PIPIADDR (after)  =8000B000.  
PIPITOKN (after) =210E0DB8.
```

As a REXX function, calling an HLL sub using PIPI CALL_SUB:

```
HLLPIPI COBOL subprogram, first call, using LE PIPI Call sub interface  
HLLPIPI COBOL subprogram returning.
```

```
HLLPIPI COBOL subprogram called 00000002 times.
```

```
. . .
```

```
HLLPIPI COBOL subprogram called 00000010 times.
```

```
HLLPIPI COBOL subprogram returning.
```

```
ELAPSED: 0.006767
```

Decisions, decisions ... REXX to Language Environment

- Simple timing comparison
 - REXX calling REXX subroutine implemented in LE
 - Simple HLL program written in C, writing a line of output
 - Assembler subroutine for PIPI INIT_SUB
 - Assembler subroutine for PIPI CALL_SUB to HLL subroutine
- versus*
- HLL application as subroutine
- versus*
- HLL application as host command

Decisions, decisions ... REXX to Language Environment

- Simple timing comparison ...
 - Called 1000 times
 - About a 3 to 1 ratio of time between PIPI vs. directly called subroutine!
 - HLL application about 1000 X worse !
 - Caveats
 - Ignored time spent for PIPI INIT_SUB
 - Measurements simply with REXX elapsed timer
 - Host command was in UNIX so spawn using /bin/sh

Decisions, decisions ... REXX to Language Environment

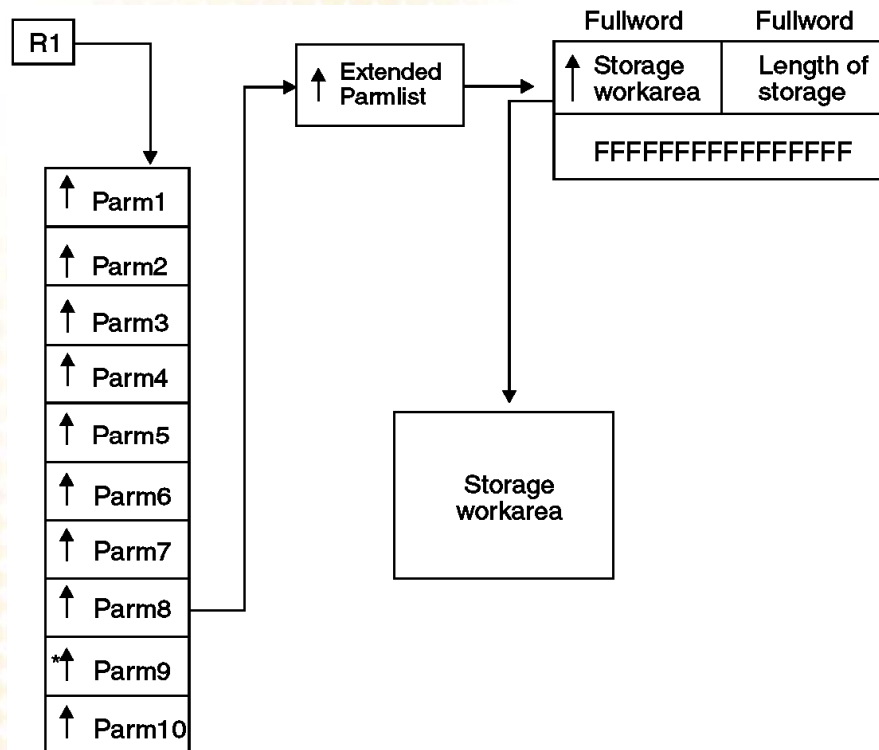
- Call directly as REXX external function or subroutine
 - Access to REXX control blocks needed to call REXX services
 - Access arguments
 - Create shared variables
 - etc.
 - Use PLIST(OS)
 - LE run-time option or C/C++ compiler option
 - Must be able to get R1 (`__osplist` macro in C/C++), the EFPL pointer
 - *Arguments*
 - *Eval Block*

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” unless a TSO/E address space
 - *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)

Dynamic Update of 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 (IRXPAMB, PARMBLOCK)
 - *Initialization parameters*
 - 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 header 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
 - 5th word points to the Argument Table
 - Parsed arguments
 - 6th 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 HLL/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 3 – SUBR1

Return Subcommand Table block in the Evaluation Block

<pre> 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_PARBLOCK 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 </pre>	<pre> ... ** YREGS ** efpl0 IRXENVB efpl1 IRXEFPL evalb IRXEVALB parmb IRXPARB subcomt IRXSUBCT ** END </pre>
---	---

Example 4a – ASMPIPC (see Example 2)

```
* Get REXX arguments
L    R1,efplarg
USING argtable_entry,R1

*

LHI  R3,0           1st arg index
L    R2,argtable_argstring_ptr(R3) 1st arg ptr
L    R2,0(R2)       1st arg (we know len is 4)
ST   R2,PPRTNPTR   Save the addr of CEEPIPI routine

*

AHI  R3,argtlen     2nd arg index
L    R2,argtable_argstring_ptr(R3) 2nd arg ptr
L    R2,0(R2)       2nd arg (we know len is 4)
ST   R2,TOKEN      Save the TOKEN

*

AHI  R3,argtlen     3rd arg index -- for callee
AR   R1,R3         . we'll
ST   R1,PARMLIS@   ...pass
LA   R1,PARMLIS@   .... only
ST   R1,PARMPTR    ..... strings
DROP R1

. . .
```


Example 4a – ASMPIPC ... (see Example 2)

* call the subroutine which was loaded by LE PIPI INIT call

```
CSUB      EQU      *
          L        R15,PPRTNPTR      Get address of CEEPIPI routine
          CALL     (15),(CALLSUB,PTBINDEX,TOKEN,PARMPTR,          X
          SUBRETC,SUBRSNC,SUBFBC)   Invoke CEEPIPI routine
          LTR      R2,R15            Is R15 = zero?
          BZ       DONE             Yes (success).. go to next section
```

Language Environment to REXX

- REXX into a module (compiled)

or

- IRXJCL, IRXEXEC

- Inherits environment

or

- Initializes environment / runs / Terminates environment

Example 5

Language Environment to REXX ...

- IRXJCL – from [z/OS TSO/E REXX Reference](#)

```
JCLXMP1 : Procedure options (Main);
/* Function: Call a REXX exec from a PL/I program using IRXJCL      */
DCL IRXJCL EXTERNAL OPTIONS(RETCODE, ASSEMBLER);
DCL 1 PARM_STRUCT,          /* Parm to be passed to IRXJCL      */
      5 PARM_LNG BIN FIXED (15), /* Length of the parameter          */
      5 PARM_STR CHAR (30);    /* String passed to IRXJCL          */
DCL PLIRETV BUILTIN;        /* Defines the return code built-in */
PARM_LNG = LENGTH(PARM_STR); /* Set the length of string         */
/*                                                                    */
PARM_STR = 'JCLXMP2 This is an arg to exec'; /* Set string value
                                           In this case, call the exec named
                                           JCLXMP2 and pass argument:
                                           'This is an arg to exec'          */
FETCH IRXJCL;              /* Load the address of entry point */
CALL IRXJCL (PARM_STRUCT); /* Call IRXJCL to execute the REXX
                           exec and pass the argument          */
PUT SKIP EDIT ('Return code from IRXJCL was:', PLIRETV) (a, f(4));
                           /* Print out the return code from
                           exec JCLXMP2.                          */
/* End of program */
END ;
```

Example 5 ... Language Environment to REXX ...

- JCLXMP2 is in SYSEXEC

```
Say 'LE has called JCLXMP2'  
/* Note we didn't need the REXX "magic number" since we  
pointed the interpreter right at it! */
```

```
Say 'ARGS:' ARG(1)  
Say 'now exiting with RC=22'
```

```
Exit 22
```

```
LE has called JCLXMP2  
ARGS: This is an arg to exec  
now exiting with RC=22
```

```
Return code from IRXJCL was: 22
```

Language Environment to REXX ...

- 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*
- UNIX users take note!
 - Executable external functions or subroutines that are written in a language other than interpreted REXX and located in the z/OS UNIX file system are not supported.

Example 6 Language Environment to REXX ...

- IRXEXEC home-grown C using DSECT utility headers

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <errno.h>

#include <irxevalb.h>
#include <irxargtb.h>
#include <irxinstb.h>

typedef struct irxexec_flags_s {
    unsigned int
        command : 1,
        function : 1,
        subroutine : 1 ,
        extended_retcodes : 1 ,
        _reserved : 4;
    char _reseved_flags[3];
} IRXEXEC_FLAGS;
```

Example 6 ... Language Environment to REXX ...

- badly prototyped!

```
typedef int (IRXEXEC)(void** execblk, void** argtable, IRXEXEC_FLAGS flags,  
                    void** instor, void** cppl, void** evalb,  
                    void** work, void** user,  
                    void** envb, void* retcd );  
  
#pragma linkage(IRXEXEC, OS)  
  
int aa = 0;  
void *envbp0=0, *null=NULL, *userfld1=0;  
  
IRXEXEC_FLAGS execflags = {0, 0, 1, 1, 0};
```

Example 6 ... Language Environment to REXX ...

```
#define EVDATA_SIZE 240
struct eval1 {
    struct evalblock EVB1;
    char restofevdata[EVDATA_SIZE];
} EVAL1 = { 0, ((EVDATA_SIZE/8)+2), 0}; /* 250?, has to be 240 or 248 ? */
unsigned char *EVAL1data_ptr = &EVAL1.EVB1.evalblock_evdata;

#define ARG1 "hello"
#define ARG2 " REXX "
#define ARG3 "from LE"
#define ARGEND "\xff\xff\xff\xff\xff\xff\xff\xff"

struct argtable1 {
    struct argtable_entry args1[3];
    struct argstring args1end;
} ARGTABLE1 =
{ARG1, sizeof(ARG1)-1,
 ARG2, sizeof(ARG2)-1,
 ARG3, sizeof(ARG3)-1,
 ARGEND};
```


Example 6 ... Language Environment to REXX ...

```
char *REXX1[] = {  
    "Say; Say COPIES('= ',72)"  
    , "Parse Arg arg1, arg2, arg3"  
    , "Say \"Args to REXX are:\" arg1\", \" arg2\", \" arg3\".\""  
    , "Parse Source src"  
    , "Say \"REXX source is '\"src\"'\""  
    , "Say COPIES('= ',72); Say"  
    , "Exit 72"  
};
```

```
#define REXXIN1_CNT (sizeof(REXX1)/sizeof(char*))  
struct instblk_entry REXXIN1[REXXIN1_CNT];
```

```
struct instblk_header INSTBLK1 =  
{ "IRXINSTB", sizeof(INSTBLK1),  
  0, 0, /* address of INSTOR exec */  
  sizeof(struct instblk_entry)*REXXIN1_CNT,  
  "insideMe", "-ddNone-", "SH      ",  
  0, 8, "NoDSname", 0};
```

Example 6 ... Language Environment to REXX ...

```
IRXEXEC* execfunc;

int retcode, RC, rr;

struct argtable1      *ARGTABLE1_ptr = &ARGTABLE1;
struct eval1          *EVAL1_ptr     = &EVAL1;
struct instblk_header *INSTBLK1_ptr  = &INSTBLK1;

INSTBLK1._instblk_address = &REXXIN1[0];

for (rr=0; rr<REXXIN1_CNT; rr++) {
    REXXIN1[rr].instblk_stmt_ = REXX1[rr];
    REXXIN1[rr].instblk_stmtlen = strlen(REXX1[rr]);
}

execfunc = (IRXEXEC*) fetch("IRXEXEC");
if (execfunc==NULL) { perror("fetch IRXEXEC failed:"); exit(1); }
```

Example 6 ... Language Environment to REXX ...

```
ARGTABLE1.args1[0].argtable_argstring_ptr = argv[1];  
ARGTABLE1.args1[0].argtable_argstring_length = strlen(argv[1]);  
  
RC=execfunc(&null,  
            (void**) &ARGTABLE1_ptr,  
            execflags,  
            (void*)&INSTBLK1_ptr,  
            &null,  
            (void*)&EVAL1_ptr,  
            &null,  
            &userfld1,  
            &envbp0,  
            &retcode);  
  
printf("retcode is: %d\n", retcode);  
printf("EVAL_data is: %.8s\n", EVAL1data_ptr);  
  
if (RC==0) RC=strtoul((char *)EVAL1data_ptr,NULL,0);  
  
exit(RC);
```

Example 6 ... Language Environment to REXX ...

```
EXECXMP1 “ here’s some arguments: the PID is 66193”
```

```
=====
Args to REXX are: here's some arguments: the PID is 66193, REXX , from LE.
REXX source is 'TSO SUBROUTINE inSideMe -ddNone- NODSname ? SH MVS ?'
=====
```

```
retcode is: 0
EVAL_data is: 72
```

Sharing Variables

- IRXEXCOM – REXX exec communication
 - 4th parameter points to ...
 - SHVBLOCK (IRXSHVB) – shared variable request block
 - SHVBLOCKS can be chained
 - HLL/ 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 7a – HLLPIPIM2 COBOL (see Example 1)

CBL LIB,QUOTE

*

IDENTIFICATION DIVISION.
PROGRAM-ID. COBPIPIM2.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY IRXSHVB.

*

77 REXXRTN PIC X(8) VALUE "IRXEXCOM".
77 REXXRC PIC S9(9) BINARY.
77 REXXRTNRC PIC S9(9) BINARY.
77 SHVRETB PIC S9(9) BINARY.

*

77 NULADDR PIC 9(9) USAGE BINARY VALUE 0.

*

Example 7a – HLLPIPIM2 COBOL ... (see Example 1)

```
01 VarName1 VALUE "006LEREXX".  
02 VarName1L PIC 999.  
02 VarName1V PIC X  
    Occurs 1 To 250 Times Depending on VarName1L.  
*  
01 VarValue1 VALUE "023and COBOL a11 together!".  
02 VarValue1L PIC 999.  
02 VarValue1V PIC X  
    Occurs 1 To 250 Times Depending on VarValue1L.
```

Example 7a – HLLPIPIM2 COBOL ... (see Example 1)

PROCEDURE DIVISION.

 PARA-REXXVAR.

 Initialize SHVBLOCK Replacing AlphaNumeric By X'00'.

* STORE (direct not symbolic) the variable

 Move SHVSTORE TO SHVCODE OF SHVBLOCK.

* The variable name

 Set SHVNAMA OF SHVBLOCK To Address of VARNAME1V(1).

 Move VARNAME1L TO SHVNAML OF SHVBLOCK.

* The variable value

 Set SHVVALA OF SHVBLOCK To Address of VARVALUE1V(1).

 Move VARVALUE1L TO SHVVALL OF SHVBLOCK.

Example 7a – HLLPIIM2 COBOL ... (see Example 1)

- * call the REXX service as a subprogram to set the
- * shared variable

```
CALL REXXRTN USING REXXRTN ,  
    OMITTED , OMITTED ,  
    SHVBLOCK ,  
    NULADDR , REXXRC  
Returning REXXRTNRC.
```

*

```
MOVE SHVRET To SHVRETB.  
DISPLAY "REXX RC IS " REXXRC ","  
    " IRXEXCOM RC IS " REXXRTNRC ","  
    " SHVRET IS " SHVRETB "." .
```

*

```
GOBACK.
```

Example 7a – IRXSHVB COBOL ... (see Example 1)

```
*****  
*      SHARED VARIABLE REQUEST BLOCK  
*****  
01  SHVBLOCK.  
*      Chain pointer to next SHVBLOCK  
02  SHVNEXT          USAGE POINTER.  
*      Used during "FETCH NEXT"  
*      Contains length of buffer  
*      pointed to by SHVNAMA  
02  SHVUSER          PIC S9(9) BINARY.  
02  SHVCODES.  
*      Function code - indicates type  
03  SHVCODE          PIC A.  
*      Return codes  
03  SHVRET          PIC X.
```

Example 7a – IRXSHVB COBOL ... (see Example 1)

```
*          Reserved (should be 0)
02          PIC X(2).
*          Length of fetch value buffer
02  SHVBUFL          PIC S9(9) BINARY.
*
*          Address of variable name
02  SHVNAMA          USAGE POINTER.
*          Length of variable name
*          (Set on fetch)
02  SHVNAML          PIC S9(9) BINARY .
*          Address of value buffer
02  SHVVALA          USAGE POINTER.
*          Length of value buffer
02  SHVVALL          PIC S9(9) BINARY .
*
*          Length of SHVBLOCK
77  SHVBLEN          PIC 9(9) BINARY VALUE IS 32.
```


Example 7a – IRXSHVB COBOL ... (see Example 1)

```
*****  
*   SHARED VARIABLE REQUEST BLOCK - function codes  
*****  
*   Set variable from given value  
77   SHVSTORE          PIC A VALUE IS 'S'.  
*   Copy value of variable to Buffer  
77   SHVFETCH         PIC A VALUE IS 'F'.  
*   Drop variable  
77   SHVDROPV        PIC A VALUE IS 'D'.  
*   Symbolic name Set variable  
77   SHVSYSET        PIC A VALUE IS 's'.  
*   Symbolic name Fetch variable  
77   SHVSYFET        PIC A VALUE IS 'f'.  
*   Symbolic name DROP variable  
77   SHVSYDRO        PIC A VALUE IS 'd'.  
*   Fetch next variable  
77   SHVNEXTV        PIC A VALUE IS 'N'.  
*   Fetch private information  
77   SHVPRIV         PIC A VALUE IS 'P'.
```

Example 7a – IRXSHVB COBOL ... (see Example 1)

```
*****  
*   SHARED VARIABLE REQUEST BLOCK - return codes (SHVRET)  
*****  
*   Execution was OK  
77   SHVCLEAN          PIC X VALUE IS X'00'.  
* variable did not exist  
77   SHVNEWV          PIC X VALUE IS X'01'.  
* Last variable transferred ("N")  
77   SHVLVAR          PIC X VALUE IS X'02'.  
* Truncation occurred for "Fetch"  
77   SHVTRUNC        PIC X VALUE IS X'04'.  
* Invalid variable name  
77   SHVBADN          PIC X VALUE IS X'08'.  
* Invalid value specified  
77   SHVBADV          PIC X VALUE IS X'10'.  
* Invalid function code (SHVCODE)  
77   SHVBADF          PIC X VALUE IS X'80'.
```

Example 4b – ASMPIPI (see Example 2)

```

ASMPIPI CSECT
ASMPIPI AMODE 31
ASMPIPI RMODE ANY
STM R14,R12,12(R13)
LR R12,R15
USING ASMPIPI,R12
ST R13,SAVE+4
LA R15,SAVE
ST R15,8(R13)
LR R13,R15

```

```

Save caller's registers
Get base address
Identify base register
Back-chain the save area
Get addr of this routine's save area
Forward-chain in caller's save area
R13 -> save area of this routine

```

*

```

LR R3,R0
USING ENVBLOCK,R3

```

Save usable ENVblock

*

* Load LE CEEPIPI service routine into main storage.

```

LOAD EP=CEEPIPI
ST R0,PPRTNPTR

```

```

Load CEEPIPI routine dynamically
Save the addr of CEEPIPI routine

```

Example 4b – ASMPIPI ... (see Example 2)

```
* =====  
* Initialize an LE Preinitialization subroutine environment.  
* =====  
INIT_ENV EQU *  
    LA    R5,PPTBL           Get address of Preinitialization Tabl  
    ST    R5,@CEXPTBL       Ceexptbl-addr->Preinitialization Tabl  
    L     R15,PPRTNPTR       Get address of CEEPIPI routine  
*                               Invoke CEEPIPI routine  
    CALL (15), (INITSUB,@CEXPTBL,@SRVRTNS,RUNTMOPT,TOKEN)  
*                               Check return code:  
    LTR   R2,R15             Is R15 = zero?  
    BZ    SETVAR              Yes (success).. go to next section  
*                               No (failure).. issue message, abend  
    WTO   'ASMPIPI : INIT_SUB failure RC is not 8.',ROUTCDE=11  
    ABEND (R2),DUMP          Abend with bad RC and dump memory
```

Example 4b – ASMPIPI ... (see Example 2)

```

=====
* Set the REXX variables PIPIADDR and PIPITOKN for access
* to the REXX driver, for use on subsequent calls into CEEPIPI
* =====
SETVAR   EQU   *
          L     R0,TOKEN
          ST    R0,TOKENCPY
* *****
* First save the PIPI information needed later,
* in order to return in REXX vars
* *****
          MVC   vvar1(4),PPRTNPTR
          MVC   vvar2(4),TOKENCPY
* *****
* Create a shared variable request block for last variable
* *****
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(avar2-var2)
          MVC   shvr2.SHVVALA,=A(vvar2)
          MVC   shvr2.SHVVALL,=A(avar2-vvar2)

```

Example 4b – ASMPIPI ... (see Example 2)

```
* ****  
* Chain & Create a shared variable request block for next variable  
* ****  
  
LR    R0,R6  
LA    R6,shvb1  
shvr1 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)
```

Example 4b – ASMPIPI ... (see Example 2)

```

* ****
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
LA    R1,plist
*
* ****
LINK  EP=IRXEXCOM
ST    R15,myret
*

```

restore ENVBLOCK for the call!

Example 4b – ASMPIPI ... (see Example 2)

```
* =====  
* Standard exit code.  
* =====  
DONE      EQU      *  
*          LA       R15,0           Passed return code for system  
          L        R15,myret  
          L        R13,SAVE+4      Get address of caller's save area  
          L        R14,12(R13)     Reload caller's register 14  
          LM       R0,R12,20(R13)  Reload caller's registers 0-12  
          BR       R14            Branch back to caller  
  
* =====  
* CONSTANTS and SAVE AREA.  
* =====  
SAVE      DC       18F'0'  
PPRTNPTR  DS       A             Save the address of CEEPIPI routine  
TOKENCPY  DS       F             Save a copy of the TOKEN
```


Example 4b – ASMPIPI ... (see Example 2)

```

* =====
* Parameters passed to a (INIT_SUB) call.
* =====
INITSUB  DC      F'3'                Function code to initialize for subr
@CEXPTBL DC      A(PPTBL)           Address of Preinitialization Table
@SRVRTNS DC      A(0)              Addr of service-rtns vector, 0 = none
* -----
RUNTMOPT DC      CL255' '          Fixed length string of runtime optns
      ORG RUNTMOPT
      DC      C'RPTOPTS(ON) '
      DC      C'MSGFILE(SYSPRINT) '
      ORG RUNTMOPT+L'RUNTMOPT
TOKEN    DS      F                  Unique value returned (output)
* =====
* Parameters passed to a (CALL_SUB) call.
* =====
CALLSUB  DC      F'4'                Function code to call subroutine
PTBINDEX DC      F'0'                The row number of Preinitialization
*                                     Table entry
PARMPTR  DC      A(0)              Pointer to @PARMLIST or zero if none
SUBRETC  DS      F                  Subroutine return code (output)
SUBRSNC  DS      F                  Subroutine reason code (output)
SUBFBC   DS      3F                Subroutine feedback token (output)

```

Example 4b – ASMPIPI ... (see Example 2)

```
* =====  
* Preinitialization Table.  
* =====  
PPTBL      CEEXPIT      ,      Preinitialization Table with index  
           CEEXPITY HLLPIPI2,0      0 = dynamically loaded routine  
*  
           CEEXPITS      ,      End of PreInit table  
*  
* =====  
* This should all be AUTODATA!  
* =====  
MYAREA     EQU      *  
*
```

Example 4b – ASMPIPI ... (see Example 2)

```

* =====
* REXX stuff
* =====
*
* *****
* Shared variable 1 name and value
* *****
var1      DC      C'PIPIADDR'
evvar1    EQU     *
* *****
vvar1     DC      CL4'addr'           will be overwritten
evvar1    EQU     *
*
* *****
* Shared variable 2 name and value
* *****
var2      DC      C'PIPITOKN'
evvar2    EQU     *
* *****
vvar2     DC      CL4'tokn'          will be overwritten
evvar2    EQU     *

```

Example 4b – ASMPIPI ... (see Example 2)

```
* ****  
* Parameters for IRXEXCOM call  
* ****  
myret      DS      F  
mysize     DS      F  
**  
plist      DS      OD  
parm1      DS      F  
parm2      DS      F  
parm3      DS      F  
parm4      DS      F
```

Example 4b – ASMPIPI ... (see Example 2)

```

* ****
* Shared variable block mappings
* ****
shvb1    ORG  *+SHVBLEN
shvb2    ORG  *+SHVBLEN
* ****
MYAREASZ EQU  *-MYAREA
* ****

        LTORG
* =====
* Control block macro mappings
* =====
envb0    IRXENVB
evalb    IRXEVALB
parmb    IRXPAMB
shrvar   IRXSHVB
* ****
* Map register names to variables
* ****
        YREGS
* ****

        END  ASMPIPI

```

Example 7b – HLLPIPIM2 C ... (see Example 1)

```
LEREXX before = <LEREXX>  
C main beginning 1 args: <1=BPXWREXC>,  
  <__osplist=217a949c>  
arg[000]=<hi there>  
arg[001]=<you all>  
...  
0.043548  
LEREXX after = <perfect together>
```

Example 7b – HLLPIPIM2 C... (see Example 1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <irxefp1.h>
#include <irxargtb.h>
#include <irxshvb.h>
#include <irxenvb.h>

typedef int (IRXEXCOM)(char func[8],
    void *zero2, void *zero3,
    struct shvblock shvbp, int envbp0,
    int retcode);

#pragma linkage(IRXEXCOM, OS)

int cnt = 0, envbp0=0;
```

Example 7b – HLLPIPIM2 C... (see Example 1)

```
main (int argc, char **argv)
{
    struct efpl * EFPLP = (struct irxefpl *) __osplist;
    struct argtable_entry * ARGTEP = EFPLP->efplarg;

    char varname1[] = "LEREXX";
    char varvalue1[] = "perfect together";

    IRXEXCOM* excomfunc;
    int retcode;
```


Example 7b – HLLPIPIM2 C ... (see Example 1)

```
struct shvblock SHVAR1 = { 0 };  
  
SHVAR1.shvcodes._shvcode = shvstore;  
  
SHVAR1.shvnama = &varname1;  
SHVAR1.shvnaml = sizeof(varname1)-1;  
  
SHVAR1.shvvala = &varvalue1;  
SHVAR1.shvvall = sizeof(varvalue1)-1;
```

Example 7b – HLLPIPIM2 C ... (see Example 1 & DSECT utility argtable_entry)

```
excomfunc = fetch("IRXEXCOM");
if (excomfunc==NULL) { perror("fetch IRXEXCOM failed:"); exit(1); }

fprintf (stderr,
        "C main beginning %d args: <1=%s>, <__osplist=%x>\n",
        argc, argv[0], __osplist);

cnt=0;
while (ARGTEP[cnt].argtable_argstring_ptr != (void *) -1)
{
    fprintf (stderr, "arg[%03d]=<%. *s>\n",
            cnt,
            ARGTEP[cnt].argtable_argstring_length,
            ARGTEP[cnt].argtable_argstring_ptr);
    cnt++;
}

excomfunc("IRXEXCOM", NULL, NULL, SHVAR1, envbp0, retcode);

return(0);
```

If you code in C/C++

- For C/C++ the DSECT conversion utility – EDCDSECT
- SYSADATA override required for multiple steps in one batch job

```
//      SET      REL=ZOS1D0
//      SET      INLIB=SYS1.MACLIB
//      SET      OUTLIB=BARRYL.BINDER.MACLIB
// *-----
//IRXARGTB EXEC PROC=EDCDSECT,
//          INFILE=DUMMY,
//          OUTFILE=&OUTLIB.(IRXARGTB),
//          DPARM='EQU(DEF),LOC(En_US.IBM-1047),PP',
//          LIBPRFX=&REL..CEE,
//          LNGPRFX=&REL..CBC
//ASSEMBLE.SYSADATA DD DSN=&&IRXARGTB
//ASSEMBLE.SYSLIB DD DSN=&INLIB.(IRXARGTB),DISP=SHR
//ASSEMBLE.SYSIN DD *
          CSECT
          IRXARGTB
          END
/*
```

If you code in C/C++ ...

- Some editing is required for some DSECT utility created headers...
 - Because REXX defines with alignments

ARGTABLE_ENTRY DSECT		REXX Argument Table Entry
DS 0D		Align on doubleword boundary
ARGTABLE_ARGSTRING_PTR	DS A	Address of the argument string
ARGTABLE_ARGSTRING_LENGTH	DS F	Length of the argument string
ARGTABLE_NEXT	DS 0D	Next ARGTABLE entry

- C/C++ doesn't have a comparable capability
 - It uses "natural alignments so requires a member of that size...
 - OK if extra last field, not OK for this array of arguments!

```
struct argtable_entry {  
void *argtable_argstring_ptr; /* Address of the argument string */  
int argtable_argstring_length; /* Length of the argument string */  
/*double argtable_next;*/ /* Next ARGTABLE entry */  
};
```

If you code in C/C++ ...

- Some editing is required for some DSECT utility created headers...
 - Because REXX defines with alignments

INSTBLK_ENTRY DSECT			REXX In-Storage Block Entry.
	DS	0F	Align on a fullword boundary
INSTBLK_STMT@	DS	A	Address of REXX statement
INSTBLK_STMTLEN	DS	F	Length of the REXX statement
INSTBLK_NEXT	DS	0CL8	Next INSTBLK_ENTRY

```
struct instblk_entry {  
    void          *instblk_stmt_; /* Address of REXX statement */  
    int           instblk_stmtlen; /* Length of the REXX statement */  
    /*unsigned char instblk_next[8]; ** Next INSTBLK_ENTRY */  
};
```

- Similar issues necessitated removing nesting structure from **instblk_header**
 - and #ifdef'd out "helpful" #defines that were because of inner structure

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 ..
 - *for most of the UNIX REXX functions such as `getpass()`*

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 HLL/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 ...

- New & Improved PD!
- IRX0900E REXX INITIALIZATION FAILED WITH RETURN CODE 20 AND REASON CODE 1.
 - OA07204 - NEW FUNCTION - MSGISPI025 TSO/E ROUTINE IRXINIT SEVERE ERROR RAS ENHANCEMENT
 - *Opened 2004, Closed 2010/07/22*
 - *PTFs available 2010/10/18 for z/OS V1.9 & later*

Miscellany ...

- [z/OS TSO/E REXX Reference – SA22-7790](#)
- [z/OS Using REXX and z/OS UNIX System Services – SA22-7806](#)
- [z/OS Language Environment Programming Guide – SA22-7561](#)
- [z/OS XL C/C++ User's Guide – SC09-4767](#)
- [z/OS XL C/C++ Programming Guide – SC09-4765](#)