*25th International Rexx Language Symposium, Memphis TN, 2014-05-04*

# Things to Do with Rexx When You're on Z

René Vincent Jansen

# Previously

- There was EXEC2 on VM
- There was CLIST on TSO
- But the console typewriter ran out of &&&&ampersands

# 1979, REX is born

- Yes, it is 35 years old today
- A second X was bought for one million $
- Introduced publicly in 1981, Houston, TX
- IBM Product 1982 due to customer demand

# It is a high-level language

- Scripts
- Applications
- Scientific and Commercial Programming
- Without resorting to an assembler language

THE RexX LANGUAGE

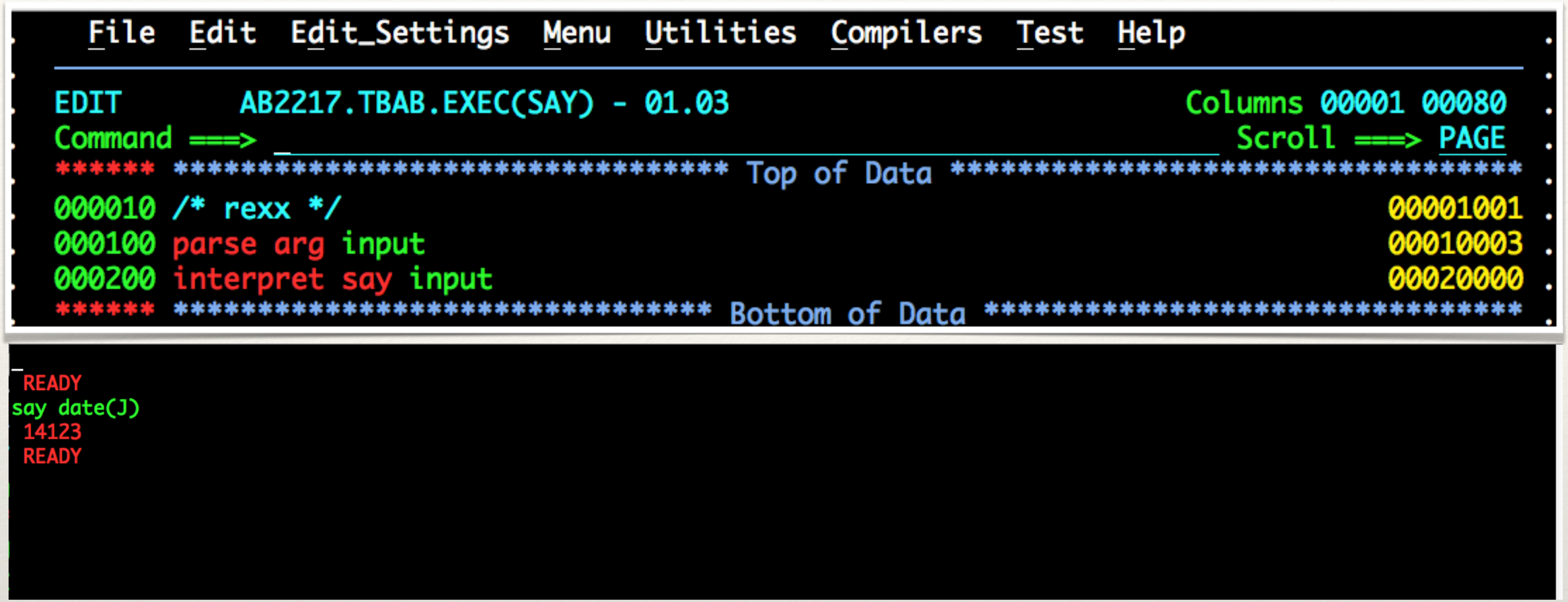
**The story of how a single language answers the question, "Can a young girl with no previous programming experience find happiness handling both commercial and scientific applications, without resorting to an assembler language?"**

Let's face it. The cost of programming just keeps going up. So for some time to come, how well you do your job depends on how programmers like Susie Meyer do theirs.

That's the reason for PL/I, the high-level language for both scientific and commercial applications.

With PL/I, programmers don't have to learn other high-level languages. They can concentrate more on the job, less on the language.

So think about PL/I. Not just in terms of training, but in terms of the total impact it can have on your operation.



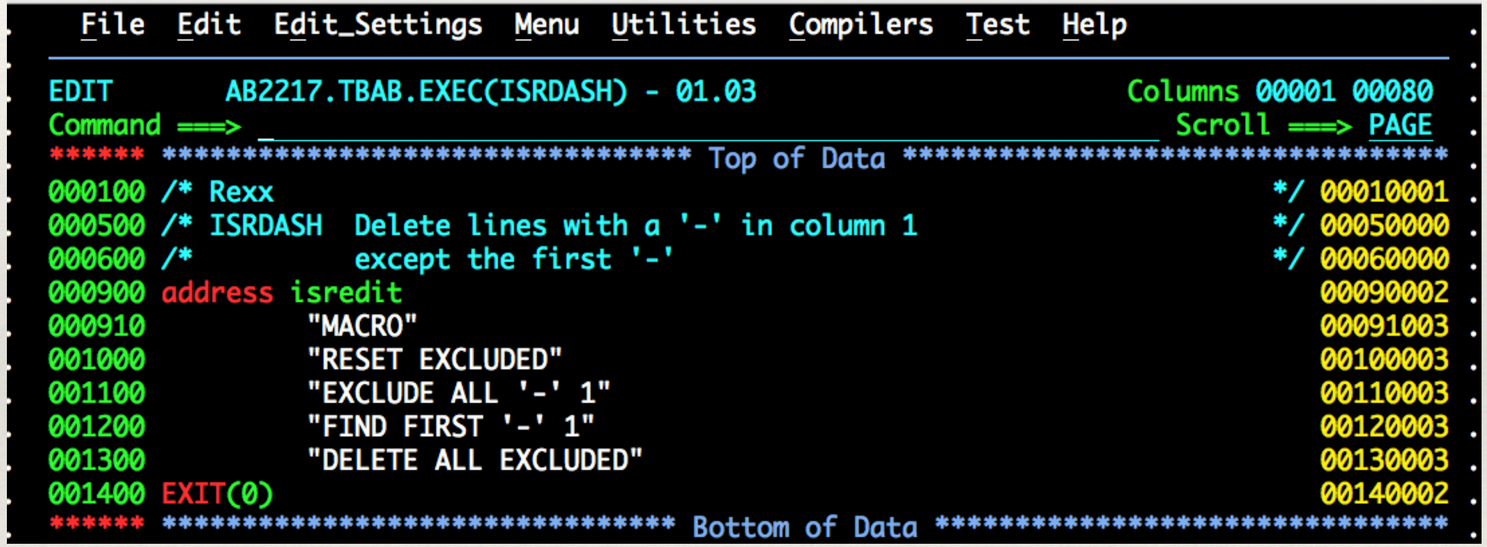

*The most useful command ever: Rexx' read-evaluate-print*

# Command line scripting

In this case, the Julian date

# ISPF Edit Macro Language

- ISREDIT enables quick writing of edit macros




- An example isredit macro in Rexx
- Note the “address isredit” to set the environment

# Write an ISPF application in Rexx

- ISPF shares its variable pool with Rexx: A Rexx ISPF application has “nothing to declare”
- Define your panels using GML or just the old fashioned way
- Implement the logic in Rexx

# DB2

- Can execute SQL and make complete applications
- Stored procedures
- DB2 command procedures
- Formatting traces

# Supported DB2 statements

- CALL
- CLOSE
- CONNECT
- DECLARE CURSOR
- DESCRIBE *prepared statement or table*
- DESCRIBE CURSOR
- DESCRIBE INPUT
- DESCRIBE PROCEDURE
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH
- OPEN
- PREPARE
- RELEASE *connection*
- SET CONNECTION
- SET CURRENT PACKAGE PATH
- SET CURRENT PACKAGESET
- SET *host-variable* = CURRENT DATE
- SET *host-variable* = CURRENT DEGREE
- SET *host-variable* = CURRENT MEMBER
- SET *host-variable* = CURRENT PACKAGESET
- SET *host-variable* = CURRENT PATH
- SET *host-variable* = CURRENT SERVER
- SET *host-variable* = CURRENT SQLID
- SET *host-variable* = CURRENT TIME
- SET *host-variable* = CURRENT TIMESTAMP
- SET *host-variable* = CURRENT TIMEZONE

```
123  SIGNAL ON ERROR
124
125  SELECT_STMT = 'SELECT EMPPROJACT.PROJNO, PROJNAME, COUNT(*),            ',
126              '   SUM( (DAYS(EMENDATE) - DAYS(EMSTDATE)) * EMPTIME *    ',
127              '       DECIMAL(( SALARY / ? ),8,2) )                      ',
128              'FROM CORPDATA/EMPPROJACT, CORPDATA/PROJECT, CORPDATA/EMPLOYEE',
129              'WHERE EMPPROJACT.PROJNO = PROJECT.PROJNO  AND             ',
130              '     EMPPROJACT.EMPNO = EMPLOYEE.EMPNO  AND               ',
131              '     PRENDATE > ?                                         ',
132              'GROUP BY EMPPROJACT.PROJNO, PROJNAME                      ',
133              'ORDER BY 1                                                '
134  EXECSQL,
135       'PREPARE S3 FROM :SELECT_STMT'
136  11EXECSQL,
137       'DECLARE C2 CURSOR FOR S3'
138  EXECSQL,
139       'OPEN C2 USING :WORK_DAYS, :RAISE_DATE'
140
141  /* Handle the FETCH errors and warnings inline */
142  SIGNAL OFF ERROR
143
144  /* Fetch all of the rows */
145  DO UNTIL (SQLCODE <> 0)
146   12EXECSQL,
147         'FETCH C2 INTO :RPT2.PROJNO, :RPT2.PROJNAME,   ',
148         '           :RPT2.EMPCOUNT, :RPT2.TOTAL_COST '
149
150   /* Process any errors that may have occurred. Continue so that   */
151   /* we close the cursor for any warnings.                         */
152   IF SQLCODE < 0 THEN
153     SIGNAL ERROR
154
155   /* Stop the loop when we hit the EOF. Don't try to print out the */
156   /* fetched values.                                               */
157   IF SQLCODE = 100 THEN
158     LEAVE
159
160   /* Print out the fetched row */
161   SAY RPT2.PROJNO '  ' RPT2.PROJNAME ' ',
162       RPT2.EMPCOUNT '       ' RPT2.TOTAL_COST
163  END;
164
165  EXECSQL,
166       'CLOSE C2'
167
```

# Format DB2 Traces

- Start a monitor trace
- Dest OPX
- format an IFCA block

```
000100 /* rexx */                                                          00010004
000102 rc = callifi('-stop  trace(mon)                    ')               00010207
000103 rc = callifi('-start trace(mon) class(1)   dest(opx)')              00010307
000104 do 10                                                              00010407
000105   call ifiread                                                     00010507
000106 end                                                                00010607
000107 exit                                                               00010707
000110 callifi: procedure expose ifca                                     00011007
000200 parse upper arg cmd                                                00020004
000400 if left(cmd,2) = ""'" & right(cmd,2) = "'"" then                   00040004
000500 cmd = substr(cmd,2,length(cmd)-2)                                  00050004
000600 else                                                               00060004
000700 if left(cmd,2) = """'" & right(cmd,2) = "'""" then                 00070004
000800 cmd = substr(cmd,3,length(cmd)-4)                                  00080004
000900 command = substr("COMMAND",1,18," ")                               00090004
001000                                                                    00100004
001400 ifca = substr('00'x,1,180,'00'x)                                   00140004
001500 ifca = overlay(d2c(length(ifca),2),ifca,1+0)                       00150004
001600 ifca = overlay("IFCA",ifca,4+1)                                    00160004
001610                                                                    00161004
```

# Format DB2 traces (continued)

use linkpgm to call the db2 attachment facility

```
001700 rtrnareasize = 262144 /*1048572*/                                00170004
001800 rtrnarea = d2c(rtrnareasize+4,4)left(' ',rtrnareasize,' ')       00180004
001900 output = d2c(length(cmd)+4,2)!!'0000'x!!cmd                      00190004
002000 buffer = substr(" ",1,16," ")                                    00200004
002400 address linkpgm "dsnwli2 command ifca rtrnarea output"           00240004
002500 wrc = rc                                                         00250004
002600 rtrn= c2d(substr(ifca,12+1,4))                                   00260004
002700 reas= d2x(c2d(substr(ifca,16+1,4)))                              00270004
002800 totlen = c2d(substr(ifca,20+1,4))                                00280004
002900 say 'returncode ='   rtrn                                        00290004
003000 say 'reasoncode ='   reas                                        00300004
003100 say 'output     ='   output                                      00310004
003200 say strip(rtrnarea)                                              00320004
003300 return 0                                                         00330007
003310 ifiread: procedure expose ifca                                   00331007
003500  totlen = c2d(substr(ifca,20+1,4))                               00350007
003600  reads = substr('READS',1,8)                                     00360007
003700  rtrnarea = '00001004'x!!copies(' ',4096)                        00370007
004200  ifcidarea = '0006000000e1'x                                     00420007
004300  address linkpgm "dsnwli2 reads ifca rtrnarea ifcidarea"         00430007
004400  retcd = c2d(substr(ifca,13,4))                                  00440007
004500  reascd = d2x(c2d(substr(ifca,17,4)))                            00450007
004600 say 'returncode ='   retcd                                       00460007
004700 say 'reasoncode ='   reascd                                      00470007
004900 say strip(rtrnarea)                                              00490005
005000 return 0                                                         00500007
****** ******************************** Bottom of Data ********************************
```

# DB2 - El Cheapo Lock Monitor

- This actually worked
- Solved a nasty timeout
- It was the first day on the job

```
EDIT       AB2217.TBAB.EXEC(LOCKMON) - 01.23                    Columns 00001 00080
Command ===> _____________________________________________________ Scroll ===> PAGE
****** ******************************** Top of Data ********************************
000010 /* rexx */                                                          00001013
000100 say 'lockmon running' date()                                        00010000
000200 parse arg input                                                     00020000
000400 data = start                                                        00040000
000700 parse pull data                                                     00070003
000800 parse var data subsystem database interval                          00080003
001300 do i = 0 to 1000                                                    00130005
001310   say "++++++++++++++++++++++++++++++++++++++++"                    00131023
001400   say date()":"time('l')                                            00140022
001401   queue "-dis db("database") locks limit(99999)"                    00140121
001402   queue "END"                                                       00140220
001403   queue ""                                                          00140320
001404   x = outtrap(outlines.)                                            00140420
001410   address TSO "dsn system("subsystem")"                             00141017
001420   x = outtrap("off")                                                00142020
001430   do i = 1 to outlines.0                                            00143020
001440     say outlines.i                                                  00144020
001450   end                                                               00145020
001510   call sleep 10                                                     00151007
001520   say                                                               00152023
001600 end                                                                 00160004
****** ******************************* Bottom of Data ********************************
```

# CICS

- Make complete applications in Rexx

```
"CICS XCTL PROGRAM('PGMA') COMMAREA(COMA)"
```

# Systems Programming Language

```
/* REXX */
ascb = C2D(Storage(224,4))
assb = C2D(Storage(D2X(ascb+336),4))
jsab = C2D(Storage(D2X(assb+168),4))
jbnm = Storage(D2X(jsab+28),8)
jbid = Storage(D2X(jsab+20),8)
usid = Storage(D2X(jsab+44),8)
Say 'JOBNAME='jbnm' JOBID='jbid' USERID='usid
```

# System Rexx

- Since z/OS 1.09
- Automate all console commands
- See the 2010 symposium materials

# NetView Rexx

- This is here for a long time already
- Take care of monitoring and network automation

# The Rexx Compiler

- Delivers performance benefits
- Provides CEXEC modules for the Rexx environment
- Provides native z/OS load modules to be linked with other programs

# JCL Replacement

- Fred Brooks called JCL the worst language ever designed and has stated he is sorry it happened on his watch
- True JCL opponents could rewrite most of the jobs in Rexx; this is very seldom seen
- ADDRESS LINKMVS is your main tool here

# Calling DFSORT

- As an example, call a sort from a Rexx exec without using JCL

- "FREE FI(SYSOUT SORTIN SORTOUT SYSIN)"
- "ALLOC FI(SYSOUT)  DA(*)"
- "ALLOC FI(SORTIN)  DA('Y897797.INS1') REUSE"
- "ALLOC FI(SORTOUT)  DA('Y897797.OUTS1') REUSE"
- "ALLOC FI(SYSIN)  DA('Y897797.SORT.STMTS') SHR REUSE"
- ADDRESS LINKMVS ICEMAN
- Here are the DFSORT control statements that might appear in the Y897797.SORT.STMTS data set:

- SORT FIELDS=(5,4,CH,A)
- INCLUDE COND=(21,3,SS,EQ,C'L92,J82,M72')

- the DFSort Manual calls this a “Rexx CLIST”

# Calling ICETOOL

- "FREE FI(TOOLMSG DFSMSG VLR LENDIST TOOLIN)"
- "ALLOC FI(TOOLMSG)  DA(*)"
- "ALLOC FI(DFSMSG)  DUMMY"
- "ALLOC FI(VLR)  DA('Y897797.VARIN') REUSE"
- "ALLOC FI(LENDIST)  DA(*)"
- "ALLOC FI(TOOLIN)  DA('Y897797.TOOLIN.STMTS') SHR REUSE"
- ADDRESS LINKMVS ICETOOL
- Here are the ICETOOL statements that might appear in the Y897797.TOOLIN.STMTS data set:

- OCCURS FROM(VLR) LIST(LENDIST) -
- TITLE('LENGTH DISTRIBUTION REPORT') BLANK -
- HEADER('LENGTH')  HEADER('NUMBER OF RECORDS') -
- ON(VLEN)  ON(VALCNT)

# Scripting your apps

- In order to make your application scriptable, you define Rexx function packages that execute code in your application
- This interface is highly standardized and exhaustively documented
- Its is usual to define these in Assembler but C can also be used

# Calling Rexx from COBOL

```
       procedure division.
        000-do-main-logic.
          display "PROGRAM COBPRG   - Beginning".
          display "Return code before call is " RETURN-CODE.
      *
      *   Pass the procedure parm HELLO to IRXJCL.
      *   Pass 3 to REXX procedure 'HELLO'.
      *   Set the size of the argument.
      *
          move "HELLO  3" to ARG-CHAR.
          move 8 to arg-size.
      *   Call "IRXJCL" in order to execute the REXX procedure
          move "IRXJCL" to PGM-NAME.
          CALL PGM-NAME     USING ARGUMENT.
      *   Display the return code.
          display "Return code after  call is " RETURN-CODE.
          display "PROGRAM COBPRG   - Normal end".
           stop run.
```

# Calling Rexx from PL/1

```
FETCH IRXEXEC;
        CALL  IRXEXEC(EXECBLK_PTR,
                      ARGTABLE_PTR,
                      flags,
                      INSTBLK_PTR,
                      reserved_parm5,
                      EVALBLK_PTR,
                      reserved_workarea_ptr,
                      reserved_userfield_ptr,
                      reserved_envblock_ptr,
                      REXX_return_code_ptr);
        /* Handle the return code. */
        RETURN_CODE = PLIRETV;
        PUT SKIP EDIT ('     RETURN CODE: ' , RETURN_CODE) (A, F(4));
        PUT SKIP EDIT ('REXX RETURN CODE: ' , REXX_RETURN_CODE) (A, F(4));
        PUT SKIP EDIT ('REXX RESULT IS: ' ||
              SUBSTR(EVALBLK_EVDATA,1,EVALBLK_EVLEN)) (A);
        PUT SKIP EDIT ('End of PLIPROG') (A);
        RETURN;
        END PLIPROG;
```

This is an impression of the main call; there is some DCL overhead needed

# ZOC and its Rexx interface

- Zap-o-com is a 3270 (+5250+Unix) emulator and as such is on-topic for this talk
- It has a well maintained Rexx interface that enables use of ooRexx and Regina
- It is highly recommended (I have no stake in it, it is from a German company
- All emulator actions can be scripted

# Using Unix System Services with Rexx

- Ever tried to make an exec sleep for 10 seconds?
- This is how it is done the easy way:
- ADDRESS SYSCALL
- "sleep" 10

# Of course, z/OS is UNIX

…and has been a looong time …

- Young persons: read right to left

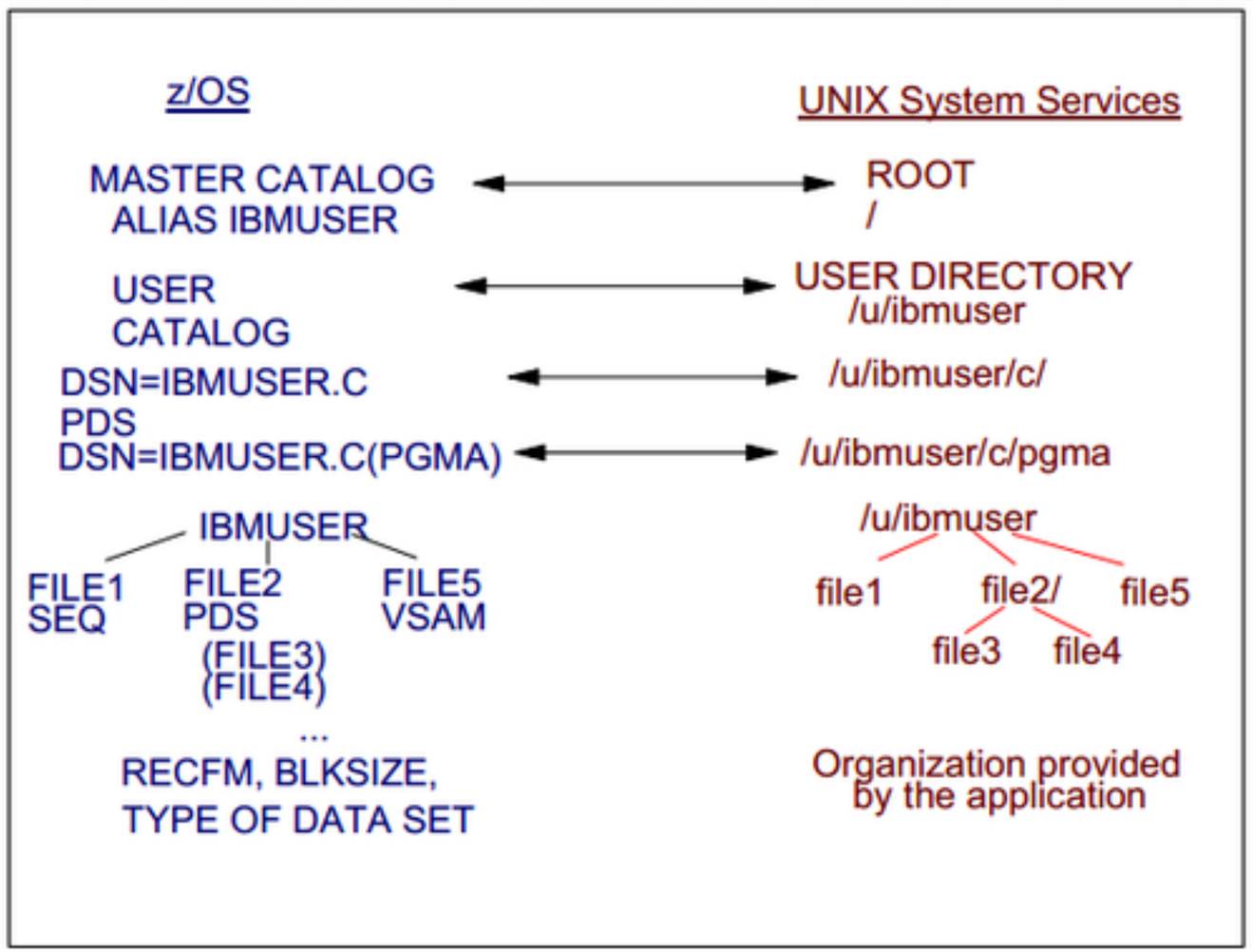



*Figure 2-12 Comparison of MVS data set and file system files*

# Wait, does it have Stream-IO?

- Of course, it has them, the UNIX way
- It is ironic that this environment has these calls, years after they did not made the source-freeze when Rexx went to Endicott
- Let us look a bit deeper into this very modern way to write Rexx on z/OS

*Classic Rexx into the 21st Century*

# z/OS Unix adds three environments to ADDRESS

SYSCALL - for, erm, System Calls

SH - The Unix Shell

TSO - The (very non-optional) Time Sharing Option

# File System Considerations

- A Rexx program that is invoked from a z/OS shell or from a program must be a text file or a compiled Rexx program that resides in the z/OS Unix file system
- It must be readable and writeable
- CEXEC output can be executed in the z/OS shell environment. The catalogued procedure REXXOEC can be used to compile and OCOPY the program to the Unix filesystem in one go


# SYSCALL

# SYSCALL can be run from TSO/E or Unix Shell

- start program with `syscall('on')`
  - ensures that `ADDRESS syscall` is enabled
  - ensures that the address space is a process (this is called 'dubbing')
  - initializes the Rexx variables in the initial variable pool
  - sets the process signal mask to block all blockable signals
  - clears the *argc* and *argv* variables


SH

# The SH environment

For a REXX program with syscall commands that will be run from a z/OS shell or from a program, SH is the initial host environment. The SYSCALL environment is automatically initialized as well, so you do not need to begin the REXX program with a **syscalls('ON')** call.

Syscall commands within the REXX program (for example, **chmod**) are interpreted as z/OS shell commands, not as syscall commands.

# Using external functions and subroutines

- The search path for subroutines and external functions is similar to that for a Rexx program that is used from a z/OS shell or a program
- The PATH variable is used to locate programs that are called by only using the file name
- For executable programs, LPA, link list and STEPLIB are searched
- If the name contains special or **lowercase** characters, quotes must be used
  - *ans='myfunc'(p1,p2)*
  - Otherwise, the name is folded to uppercase
- Only interpreted Rexx programs are found in the z/OS Unix filesystem, other languages and compiled Rexx is not found in the filesystem (but is found in STEPLIB or LPA, link list)


# TSO

# The TSO environment

A REXX program can run TSO/E commands, but you cannot use TSO commands to affect your REXX environment, or have REXX statements or other host command environments affect your TSO process.

Commands that are addressed to TSO will be run in a TMP running in a separate address space and process from your REXX program.

The TSO process is started when the first TSO command is run, and persists until your REXX program terminates or you run the TSO LOGOFF command.

# ADDRESS TSO

- The TSO command environment can be used from a z/OS Unix Rexx environment, and is initialized with:
  - `address tso [command]`
- where [command] may be any TSO command, clist, exec that can run in a TSO batch tmp
- the started program can be observed with **ps** as process **bpxwrtso**

# TSO Input

- Most TSO programs use TGET for input and will fail
- For commands that are able to read input, first data is what is on the stack, and then any data that is in your Rexx exec's standard input stream
- The standard input stream may also be queued as part of the input stream

For example, if you have a file redirected as input and you run a TSO command before processing that file, some or all of the file may be queued to the TSO command. If input is the terminal, queued input may be queued to the TSO command.

This characteristic can be used to interact with some TSO commands.

# TSO Output

- The standard output stream of the Rexx exec will be used
- The outtrap() function can be used to store output in a variable

# TSO Examples

To run the TSO/E TIME command:

```
address tso ’time’
```

To trap command output and print it:

```
call outtrap out.
   address tso ’listc’
   do i=1 to out.0
say out.i end
```

To run a REXX exec in TSO/E:

```
address tso
   "alloc fi(sysexec) da(’schoen.rexx’) shr"
   "myexec"
```

# More examples

# Variable Scope

When the REXX program is initialized and the SYSCALL environment is established, the predefined variables are set up. If you call an internal subroutine that uses the PROCEDURE instruction to protect existing variables by making them unknown to that subroutine (or function), the predefined variables also become unknown. If some of the predefined variables are needed, you can either list them on the PROCEDURE EXPOSE instruction or issue another **syscalls('ON')** to reestablish the predefined variables. The predefined variables are automatically set up for external functions and subroutines. For example:

```
subroutine: procedure
junk = syscalls('ON')
parse arg dir
'readdir (dir) dir. stem.'
```

# Running Rexx in z/OS Unix from a C program

```
#pragma strings(readonly)
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
typedef int EXTF();
#pragma linkage(EXTF,OS)
int main(int argc, char **argv) {
extern char **environ;
EXTF *irxjcl;
EXTF *bpxwrbld;
/* access environ variables     */
/* pointer to IRXJCL routine    */
/* pointer to BPXWRBLD routine  */
/* addr of REXX environment     */
/* temps                        */
/* return code                  */
/* ptr to env length pointers   */
/* ptr to env lengths           */
/* OE MVS env work area         */
/* name of exec up to 8 chars   */
char *penvb;
int  i,j;
long rcinit;
int  **environlp;
int  *environl;
char rxwork[16000];
char *execname="execname";
char *execparm="exec parameter string"; /* parm to exec              */
struct s_rxparm {
   short len;
   char name[8];
   char space;
   char text[253];
   } *rxparm;
/* parm to IRXJCL               */
/* halfword length of parm      */
/* area to hold exec name       */
/* one space                    */
/* big area for exec parm       */
```

```
/* if stdin or stdout are not open you might want to open file     */
/* descriptors 0 and 1 here                                        */
/* if no environ, probably tso or batch - make one                 */
if (environ==NULL) {
environ=(char **)malloc(8);
environ[0]="PATH=.";
environ[1]=NULL;
};
/* create one              */
/* set PATH to cwd         */
/* env terminator          */
/* need to build the environment in the same format as expected by */
/* the exec() callable service.  See
/* Assembler Callable Services for UNIX System Services.
/* the environ array must always end with a NULL element
*/
*/

*/

for (i=0;environ[i]!=NULL;i++);
environlp=(int **)malloc(i*4+4);
environl=(int *)malloc(i*4+4);
for (j=0;j<i;j++) {
   environlp[j]=&environl[j];
   environl[j]=strlen(environ[j])+1;
   };
/* count vars
/* get array for len ptrs  */
/* get words for len vals  */
/* point to len            */
/* set len word            */
/* null entry at end       */
                           */
environlp[j]=NULL;
environl[j]=0;
```

```
/* load routines
 irxjcl=(EXTF *)fetch("IRXJCL
");
```

Chapter 2. z/OS UNIX REXX programming services **17**

```
*/

bpxwrbld=(EXTF *)fetch("BPXWRBLD ");
/* build the REXX environment                                      */
rcinit=bpxwrbld(rxwork,
argc,argv,

               i,environlp,environ,
            &penvb);
if (rcinit!=0) {
   printf("environment create failed rc=%d\n",rcinit);
   return 255;
   };
/* if you need to add subcommands or functions to the environment, */
/* or create a new environment inheriting the current one, this is */
/* the place to do it.  The user field in the environment is used  */
   /* by the z/OS UNIX REXX support and must be preserved.
   /* run exec
   rxparm=(struct s_rxparm *)malloc(strlen(execname)+
                                    strlen(execparm)+
                                    sizeof(struct s_rxparm));
   memset(rxparm->name,' ',sizeof(rxparm->name));
   memcpy(rxparm->name,execname,strlen(execname));
   rxparm->space=' ';
   memcpy(rxparm->text,execparm,i=strlen(execparm));
   rxparm->len=sizeof(rxparm->name)+sizeof(rxparm->space)+i;
   return irxjcl(rxparm);
}

*/
*/
```

For example, to define your external functions for an application support package

# chmod

assume that *pathname* was assigned a value earlier in the exec. This example changes the mode of the file to read-write-execute for the owner, and read-execute for all others:

"chmod (pathname) 755"

# Rexx I/O Functions

- lineout() and charout()
- linein() and charin()
- stream()
- streams can be opened implicitly and explicitly

# Example Rexx I/O Functions

This example opens a stream for the file mydata.txt:

```
file=stream('mydata.txt','c','open write')
```

This example opens a stream for the file mydata.txt, but replaces the file if it exists:

```
file=stream('mydata.txt','c','open write replace')
```

To read the next 256 characters:

```
say charin(file,,256)
```

To set the read location to the sixth 80-byte record:

```
call charin file,5*80+1,0
```

# Submit a Job to JES2

```
do i=1 by 1 while lines(fn)>0
      fn.i=linein(fn)
   end
   fn.0=i-1
   say submit('fn.')
```

# BPXWDYN, interface to dynamic allocation

This example allocates SYS1.MACLIB to SYSLIB and directs messages to z/OS UNIX standard error (sdterr):

```
call bpxwdyn "alloc fi(syslib) da(sys1.maclib) shr msg(2)”
```

This example requests that the name of the data set allocated to ddname SYSLIB be returned in the REXX variable *dsnvar*.

```
call bpxwdyn "info fi(syslib) inrtdsn(dsnvar)"
```

This example frees SYSLIB and traps messages in stem S99MSG.:

```
call bpxwdyn "free fi(syslib)"
```

This example concatenates SYS1.SBPXEXEC to SYSPROC:

```
if bpxwdyn("alloc fi(tmp) da(sys1.sbpxexec) shr msg(2)")=0 then
       call bpxwdyn "concat ddlist(sysproc,tmp) msg(2)"
```

# Rexx for the 21st Century

- Pervasive on z/OS
- Support for USS
- Reuse these interfaces for ooRexx on z/OS