

# Introduction to REXX Programming on z/VM

Brian Hugenbruch, CISSP  
z/VM Development, IBM: Endicott, NY, US  
[bwhugen@us.ibm.com](mailto:bwhugen@us.ibm.com)



# Agenda

- Rexx Overview and Related Topics
- Creating and Executing Rexx Programs
- Rexx Language
  - ▶ Basic Syntax
  - ▶ Strings, Operators, Expressions
  - ▶ Tracing, Parsing
  - ▶ Issuing Commands and use of Pipelines
  - ▶ Control Constructs
  - ▶ Subroutines & Functions
- Lab Exercises

## Rexx Overview

- **RE**structured **eX**tended **eX**ecutor
  
- Rexx is a procedural, general purpose language
  - ▶ Intuitive - easy to use and read
  - ▶ Many uses, ranging from:
    - Personal tools and utilities
      - For example, frequently used command sequences
    - Complex applications and licensed programs
  - ▶ Available on many IBM and non-IBM platforms
  
- Rexx is designed to be *interpreted*
  - ▶ Each program statement translated and executed as the program runs
  - ▶ Programs can also be *compiled* to improve
    - Performance
    - Security
    - Change control

## Rexx Overview (cont.)

- Few restrictions on program format
  - ▶ Indentation
  - ▶ 1 or more clauses on a line
  - ▶ */\** comments can be anywhere and any length *\*/*
  - ▶ *Implied* semicolon delimiters at end of lines
  - ▶ Comma ( , ) as a continuation character
  
- Nothing to Declare !
  - ▶ Implicit declarations take place during execution

## Rexx Platforms

- IBM Platforms
  - VM
  - TSO/E (z/OS)
  - VSE
  - AIX
  - OS/2
  
- Object Rexx
  - ▶ Object-Oriented Rexx supporting many utilities for a UNIX-type environment, including Linux for System z
  
- Regina Rexx
  - ▶ Rexx interpreter ported to most UNIX platforms, including Linux
  
- NetRexx
  - ▶ Blend of Rexx and Java; compiles into Java classes
  
- Language concepts are the same on all platforms
  - ▶ Minor differences such as file names and structure
  - ▶ Operating system-specific tools that support Rexx

*(See references page for website information)*

## Creating Rexx Programs: z/VM

- Create a file with filetype of EXEC using XEDIT, the CMS editor

**XEDIT myrexx exec a**

- Rexx programs begin with a comment line:

**/\* beginning of program \*/            /\* Rexx \*/**

- Can be run uncompiled and interpreted, or compiled with the Rexx compiler

## Executing Rexx Programs: z/VM

- Search order
  - ▶ Same for both compiled and interpreted execs
  - ▶ Loaded and started through CMS EXEC handler
  - ▶ Normal CMS Command search order:  
EXECs, synonyms, MODULEs...
  
- Invocation
  - ▶ Invoke as a CMS command or EXEC:  
**myexec** -or- **exec myexec**
  
  - Implied exec (IMPEX) settings control whether exec files are treated as commands
    - SET IMPEX ON|OFF (default is ON)
    - QUERY IMPEX

## Creating and Executing REXX Programs: TSO/E

- REXX exec can be a sequential data set or a PDS member
- TSO/E EXEC command to invoke a REXX program or a CLIST
- Three ways to use the EXEC command:
  - Explicit execution:  
**EXEC dataset(member) 'parameters' operands**
  - Implicit execution:  
**membername parameters**
  - Extended implicit execution:  
**%membername parameters**
- Search includes:
  - //SYSEXEC DD** concatenation  
then
  - //SYSPROC DD** concatenation for **membername** on the command line



## Lab Exercises: What to Expect...

1. Run an existing Rexx program to create temporary disk space
2. Write a program to accept an input argument, prompt for data, and display results
3. Trace and Debug existing Rexx programs
4. Write a program to obtain z/VM CP level information  
(*issues commands and Diagnose 8*)
5. Write a program using a subroutine to issue CMS commands and Pipes to query accessed disks

## Helpful Hints for Exercises

- List Files on A-disk: **FILELIST \* \* A** or... **LISTFILE \* \* A**
- XEDIT a file
  - from command line: **Xedit <filename> <filetype> <filemode>**
  - from prefix area on FILELIST Screen, PF11 or :

```
x PROFILE EXEC A1 V 75 74 1 09/17/07 15:48:18
```

- XEDIT Prefix area commands:
  - a** add (insert) a single line to the file
  - d** delete a line (**d5** deletes 5 lines)
  - m** move a line (**f** following or **p** preceding)
  - c** copy a line (**f** following or **p** preceding)
  - mm...mm** block move, **dd...dd** block delete, **cc...cc** block copy
- Leaving XEDIT:
  - Save changes: **FILE**
  - Quit (restore file without changes): **QQUIT**

## Helpful Hints for Exercises (cont.)

- Screen execution modes
  - ▶ **CP Read**
    - CP is waiting for a command
  - ▶ **VM Read**
    - CMS is waiting for a command
  - ▶ **Running**
    - System is ready for commands or is working on some
  - ▶ **More ...**
    - More information than can fit on the screen is waiting to be displayed)
      - Clear screen manually or let CP clear after x seconds determined by TERM command setting
  - ▶ **Holding**
    - Waiting for you to clear screen manually
  - ▶ **Not Accepted**
    - Too many commands in buffer; wait for executing command to complete)

## Logging on to the z/VM Lab System

- 3270 Session
- Userids
- Password

## Exercise 1: Create Temp Disk Space

1. Logon to your VM lab userid
2. Issue command **QUERY DISK** to see which disks are accessed
3. Run existing exec **GETTEMP** *mode* (*mode* is input parameter) to:
  - create a temporary disk at filemode *mode*
  - copy existing EXEC programs from a-disk to new temp disk

*mode* can be a letter from **b - z** representing an unused disk mode
4. Issue **QUERY DISK** again – notice new disk at *mode*
5. Issue command **FILELIST \* \* *mode***
6. Run **GETTEMP** again with mode **a**
7. Issue **QUERY DISK** again – notice new disk at mode **a**
8. **LOGOFF**

## Exercise 1: Create Temp Disk Space

```

/* Get Temporary disk space */

/* File mode of temporary disk is input argument */

parse upper arg fmode rest
If (fmode = '') | (rest = '') then
  Do
    say ''
    say 'ERROR:  Input parm is FILEMODE.'
    say ''
    exit 4
  End

'CP DETACH 555'          /* Get rid of old disk */
'CP DEFINE T3390 555 2' /* Define 2 cylinders of temp space */

queue 1                 /* Answer YES to FORMAT prompt */
queue TMP555            /* Disk label is TMP555 */
'FORMAT 555 'fmode      /* Format the disk for CMS files */

If (fmode = 'A') Then   /* If input mode is "A" move A disk to B */
  Do
    Parse Value Diag(8,'QUERY 'UserId() With thisuser .
    'access VMSYSU:'thisuser'. b/a'
    frommode = 'b'
  End
Else frommode = 'a'

'COPYFILE * exec ' frommode '= =' fmode /* COPY existing EXEC files
                                         to new temp disk */
exit 0

```

## Exercise 1: Create Temp Disk Space - Answer

query disk

LABEL	VDEV	M	STAT	CYL	TYPE	BLKSZ	FILES	BLKS USED-(%)	BLKS LEFT	BLK TOTAL
-	DIR	A	R/W	-	-	4096	44	-	-	-
MNT190	190	S	R/O	115	3390	4096	694	14562-70	6138	20700
MNT19E	19E	Y/S	R/O	355	3390	4096	1875	49995-78	13905	63900

gettemp z

```
HCPDTV040E Device 0555 does not exist
DASD 0555 DEFINED
DMSFOR603R FORMAT will erase all files on disk Z(555). Do you wish to continue?
Enter 1 (YES) or 0 (NO).
DMSFOR605R Enter disk label:
DMSFOR733I Formatting disk Z
DMSFOR732I 2 cylinders formatted on Z(555)
```

query disk

LABEL	VDEV	M	STAT	CYL	TYPE	BLKSZ	FILES	BLKS USED-(%)	BLKS LEFT	BLK TOTAL
-	DIR	A	R/W	-	-	4096	44	-	-	-
MNT190	190	S	R/O	115	3390	4096	694	14562-70	6138	20700
MNT19E	19E	Y/S	R/O	355	3390	4096	1875	49995-78	13905	63900
TMP555	555	Z	R/W	2	3390	4096	19	60-17	300	360

## Exercise 1: Create Temp Disk Space – Answer..

```
gettemp a
```

```
DASD 0555 DETACHED
```

```
DASD 0555 DEFINED
```

```
DMSFOR603R FORMAT will erase all files on disk A(555). Do you wish to continue?
```

```
Enter 1 (YES) or 0 (NO).
```

```
DMSFOR605R Enter disk label:
```

```
DMSFOR733I Formatting disk A
```

```
DMSFOR732I 2 cylinders formatted on A(555)
```

```
B (VMSYSU:PIPUSR00.) R/O
```

```
query disk
```

LABEL	VDEV	M	STAT	CYL	TYPE	BLKSZ	FILES	BLKS USED-(%)	BLKS LEFT	BLK TOTAL
TMP555	555	A	R/W	2	3390	4096	19	60-17	300	360
-	DIR	B/A	R/O	-	-	4096	44	-	-	-
MNT190	190	S	R/O	115	3390	4096	694	14562-70	6138	20700
MNT19E	19E	Y/S	R/O	355	3390	4096	1875	49995-78	13905	63900



## Rexx Language Syntax

- Case **Insensitivity**

**C**incinnati is the same as **c**incinnati

- ▶ specific support for upper and lower case is provided
- ▶ cases in quoted strings are respected

- **All** Rexx programs must begin with a comment

```
/* This is a comment */
```

- Long lines are common

- ▶ Continuation with commas

```
say 'This text is continued ',  
    'on the next line'
```

- ▶ May wrap as a long single line (*but don't do this*)

```
say 'This text is continued  
    on the next line'
```

## Rexx Strings

- Literal strings: Groups of characters inside single or double quotation marks

`"Try a game of blackjack", 'and beat the odds!'`

- Two " or ' indicates a " or ' in the string

`'Guess the dealer''s top card'`

`"The dealer""s card is an Ace"`

- Hexadecimal strings: Hex digits (0-9,a-f,A-F) grouped in pairs:

`'123 45'x` is the same as `'01 23 45'x`

- Binary strings: Binary digits (0 or 1) grouped in quads:

`'10000 10101010'b` is the same as `'0001 0000 1010 1010'b`

## Operators & Expressions

- String Expressions

(blank) "Ohio" "River" --> "Ohio River"  
|| 'Ben' || 'gals' --> 'Bengals'

(abuttal) **abc** = 'Ruther'  
**abc**'ford' --> 'Rutherford'

- Arithmetic Expressions

+ - \* / % (int division) // (remainder)  
\*\* (power) Prefix - Prefix+

## Input and Output

### ■ **say [expression]**

- ▶ writes output to the user's terminal

```
say 'Five Euros equals ' ,  
    5 * 1.35 'USD'
```

### ■ **pull**

- ▶ prompts for input from the user

```
pull rate  
say 'Five Euros equals' 5 * rate 'USD'
```

### ■ **parse arg**

- ▶ collects arguments passed to a Rexx Program

- Invoke program: **EXAMP input1 dataX moreData**

```
parse arg A1 A2 A3
```

```
say A1 A2 A3
```

- Result:

```
input1 dataX moreData
```

## Operators & Expressions

### ■ Comparative Express

▶ Normal = \= <> >< > < >= <=

- comparison is case sensitive
- leading/trailing blanks removed before compare
- shorter strings padded with blanks on right

▶ Strict == \== >> << >>= \<< <<= \>>

- comparison is case sensitive
- if 2 strings = except one is shorter, the shorter string is less than the longer string

### ■ Logical Expressions

& | &&

\ (preceding expression)

Note: the "not" sign and backslash "\ " are synonymous

## Numbers

- A Rexx character string that includes 1 or more decimal digits with an optional decimal point
  - ▶ May have leading and trailing blanks
  - ▶ Optional sign + or -
  - ▶ An "E" specifies exponential notation
    - Be careful with device addresses such as 1E00 (use quotes)
  
- Precision in calculations may be controlled by the NUMERIC DIGITS instruction
  - ▶ Default is 9 digits
  
- Examples (could also be enclosed in quotes):

**12**      **-17.9**      **+ 7.9E5**

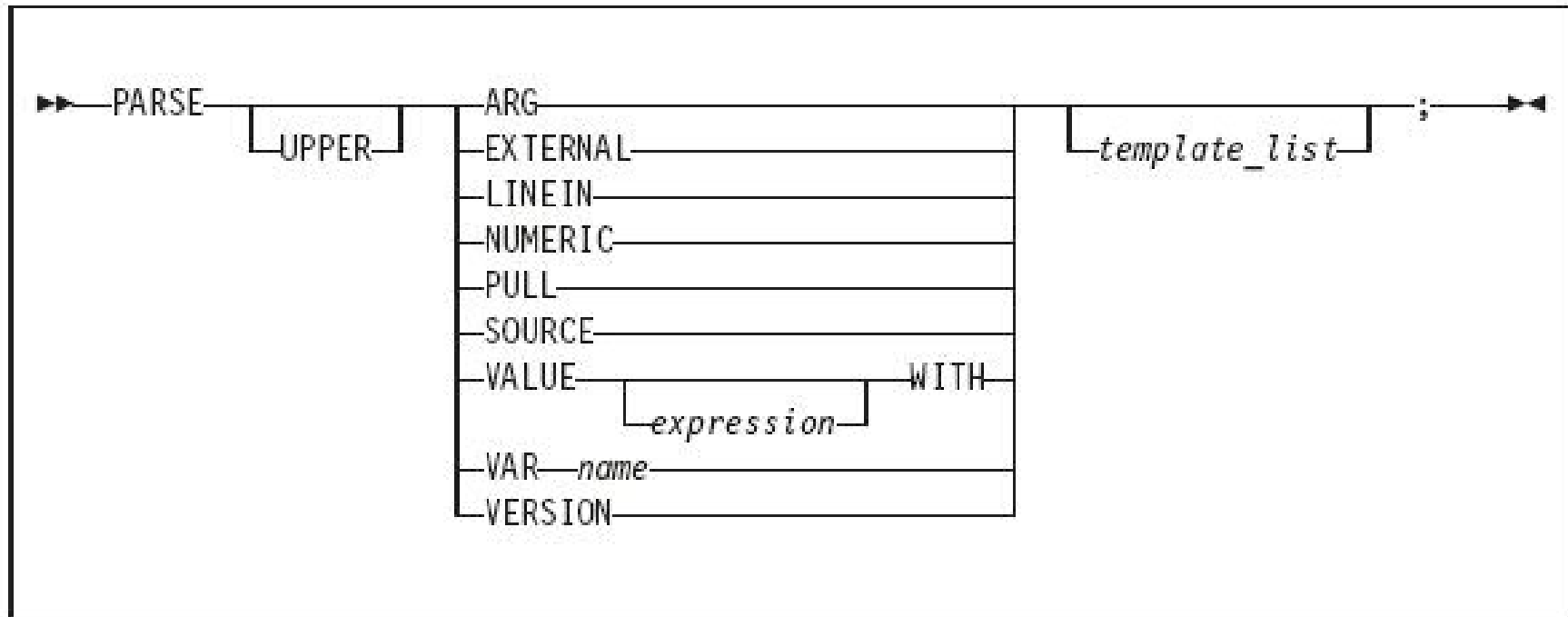
## Variables

- Data known by a unique name whose value may change
- Variable names
  - ▶ **NOT** case sensitive
  - ▶ **Cannot** begin with a digit 0-9
- Defined by assignment (give it a value)

**population = 184627**

- Variables with no assigned value will have the uppercase variable name as its initial value
- Special variables: **rc, result, sigl**
  - ▶ may be set automatically during program execution

## Parsing Strings



- Parse Arg – takes data passed into exec or internal routine
  - ▶ (see example on “Input and Output” chart)
- Parse Var – parses variable into other variable(s)



## Parsing Strings...

- Assigns data to variables using parsing rules

```
str1 = `April 13-16, 2014`
```

```
parse var str1 w1 w2 w3
```

- w1 = April
- w2 = 13-16,
- w3 = 2014

```
parse upper var str1 w1 . w2
```

- w1 = APRIL
- w2 = 2014

```
parse var str1 w1 w2
```

- w1 = April
- w2 = 13-16, 2014

## Parsing Strings...

- Default token delimiter is a blank
  - ▶ May be changed on Parse statement

```
str1 = 'April*13-16,*2014'
```

```
parse var str1 w1 '*' w2 '*' w3
```

- w1 = April
- w2 = 13-16,
- w3 = 2014

## Exercise 2: Say, Pull, & Passing Parameters

- Assume a card deck with suits of Hearts, Diamonds, Clubs, and Spades
  
- Write a Rexx program to:
  - ▶ **pass in** 1 of the 4 suits as an argument
  - ▶ **prompt** for a number from 2-10
  - ▶ **display** the number and the suit in the format:  

```
'Your card is a 10 of Hearts'
```
  
- **Run** the program with different suits and numbers

## Exercise 2: Say, Pull, & Passing Parameters - Answer

```
/* */  
parse arg suit  
say 'Enter a number from 2-10:'  
pull num  
say 'Your card is a 'num' of ' suit
```

## Tracing

- **Trace All** - clauses before execution
- **Trace Commands** - commands before execution. If the command has an error, then also displays the return code
- **Trace Error** - any command resulting in an error after execution and the return code
- **Trace Failure/Normal** – default setting, any command with a negative return code after execution, and the return code
- **Trace Intermediates** – Trace All, plus intermediate results during evaluation of expressions and substituted names
- **Trace Labels** - only labels passed during execution
- **Trace Off** - traces nothing and resets options
- **Trace Results** – Trace All, plus results of an evaluated expression and values assigned during PULL, ARG, and PARSE instructions
- **Trace Scan** – Trace All, but without the clauses being processed

## Tracing (cont.)

- output identifier tags:
  - \*-\* source of a single clause
  - >>> result of expression
  - >.> value assigned to placehldr
  - +++ error messages
  
- prefixes if TRACE Intermediates in effect:
  - >C> data is compound variable
  - >F> data is result of func call
  - >L> data is a literal
  - >O> data is result of operation on 2 terms
  - >P> data is result of prefix op
  - >V> data is contents of variable

## Tracing (cont.)

- Prefix Options **!** and **?** modify tracing and execution

**?** controls interactive debugging

### **TRACE ?Results**

**!** inhibits host command execution

**TRACE !C** causes command to be traced but not processed

- CMS command **SET EXEC TRAC ON** allows you to switch tracing on without modifying the program
- **TS** and **TE** immed commands turn tracing on/off asynchronously

## Tracing - Example

- Program

```
/* Trace Sample Program */  
Trace Intermediates  
number = 1/7  
say number
```

- Output

```
3  *-* number = 1/7  
>L>  "1"  
>L>  "7"  
>O>  "0.142857143"  
4  *-* say number  
>V>  "0.142857143"  
0.142857143
```



## Exercise 3: Tracing and Debugging

The following Rexx Programs are on your VM A-disk:

- ▶ REXXEX3A.EXEC
- ▶ REXXEX3B.EXEC

There is something wrong with each program

- ▶ Using the TRACE instruction, debug each problem
- ▶ Fix the code so that it functions properly

## Exercise 3: Tracing and Debugging – Answer A

### Trace Intermediate output:

```
6 *-* string1 = "Rexx" 'Lab'  
    >L>    "Rexx"  
    >L>    "Lab"  
    >O>    "Rexx Lab"  
7 *-* say string1  
    >L>    "STRING11"
```

**STRING11**

```
9 +++ string2 = "Exerc"|"ise'say string2
```

**Error 6 running REXXTR3A EXEC, line 9: Unmatched "/\*" or quote**

## Exercise 3: Tracing and Debugging – Answer A

### Corrected Rexx Program:

Trace I

```
string1 = "Rexx" 'Lab'  
say string1          /* Was: say string1          */  
  
string2 = "Exerc"||"ise" /* Was: string2 = "Exerc"||"ise' */  
say string2
```

### Result:

```
6 *-* string1 = "Rexx" 'Lab'  
  >L>  "Rexx"  
  >L>  "Lab"  
  >O>  "Rexx Lab"  
7 *-* say string1  
  >V>  "Rexx Lab"  
  
Rexx Lab  
9 *-* string2 = "Exerc"||"ise"  
  >L>  "Exerc"  
  >L>  "ise"  
  >O>  "Exercise"  
10 *-* say string2  
  >V>  "Exercise"
```

**Exercise**

## Exercise 3: Tracing and Debugging – Answer B

### Trace Intermediate output:

```
7 *-* Nums = "25 35 71"  
  >L>  "25 35 71"  
9 *-* parse arg w1 . w2 w3  
  >>>  ""  
  >.>  ""  
  >>>  ""  
  >>>  ""  
11 *-* $average = (w1 + w2 + w3) // 3  
  >V>  ""  
  >V>  ""  
11 +++ $average = (w1 + w2 + w3) // 3  
DMSREX476E Error 41 running REXXTR3B EXEC, line 11: Bad arithmetic conversion
```

## Exercise 3: Tracing and Debugging – Answer B

### Corrected Rexx Program:

```
Trace I
```

```
Nums = "25 35 71"
```

```
parse var Nums w1 w2 w3          /* Was: parse arg w1 . w2 w3 */
```

```
$average = (w1 + w2 + w3) / 3     /* Was: (w1 + w2 + w3) // 3 */  
say "The average value of these numbers is" $average "."
```

## Exercise 3: Tracing and Debugging – Answer B

### Result:

```
7 *-* Nums = "25 35 71"
  >L>  "25 35 71"
9 *-* parse var Nums w1 w2 w3
  >>>  "25"
  >>>  "35"
  >>>  "71"
11 *-* $average = (w1 + w2 + w3) / 3
  >V>  "25"
  >V>  "35"
  >O>  "60"
  >V>  "71"
  >O>  "131"
  >L>  "3"
  >O>  "43.6666667"
12 *-* say "The average value of these numbers is" $average "."
  >L>  "The average value of these numbers is"
  >V>  "43.6666667"
  >O>  "The average value of these numbers is 43.6666667"
  >L>  "."
  >O>  "The average value of these numbers is 43.6666667 ."
```

The average value of these numbers is 43.6666667 .

## Symbols and Stems

- Constant symbol starts with a digit (0-9) or period:

**77 .123 12E5**

- Simple symbol does not start with a digit and does not contain periods:

**ABC ?3**

- Compound symbol contains at least one period, and at least 2 other characters

- ▶ **Stem** (up to 1st period), followed by **tail**

**ABC.3 Array.i Total.\$name x.y.z**

## Symbols and Stems...

```
/* Stems as arrays */  
do i=1 to 50 by 1  
  array.i = i+5  
end  
say array.25          /* Output: "30" */  
say array.51         /* Output: "ARRAY.51" */
```

```
/* Stems as records */  
If attendee.payment == "LATE" then  
do  
  say attendee.$fullname  
  say attendee.$email  
  say attendee.$company.telephone  
end
```



## Issuing Commands from Rexx

- CP and CMS commands can be issued as a quoted string:
  - ▶ `'CP QUERY CPLEVEL'`
  - ▶ `'STATE PROFILE EXEC'`
  
- Use DIAG function to issue CP commands with Diagnose x'08'
  - ▶ `DIAG(8, 'QUERY CPLEVEL')`
    - ▶ Can be an expression as part of a longer statement
      - ▶ PARSE command output or parts of command output into variables
  
- Environment is selected by default on entry to a Rexx program
  - ▶ **ADDRESS** instruction can change the active environment
  - ▶ **ADDRESS( )** built-in function used to get name of the currently selected environment

## Issuing Commands – z/VM Example

```
Address CMS          /* send cmds to CMS */  
'STATE PROFILE EXEC'
```

```
If RC=0 Then        /* file found */  
'COPY PROFILE EXEC A TEMP = ='
```

```
/* Save command output in variable */  
Parse Value diag(8, 'QUERY CPLEVEL') With queryout  
say queryout
```

```
z/VM Version 6 Release 2.0, service level 1101 (64-bit)  
Generated at 05/09/12 19:47:52 EDT  
IPL at 06/03/12 16:29:17 EDT
```

## Issuing Commands – TSO

```
"CONSOLE ACTIVATE"
```

```
...
```

```
ADDRESS CONSOLE /* change environment to CONSOLE for all commands */
```

```
"mvs_cmd"
```

```
...
```

```
"mvs_cmd"
```

```
ADDRESS TSO tso_cmd /* change environment to TSO for one command */
```

```
...
```

```
"mvs_cmd"
```

```
ADDRESS TSO /* change environment to TSO for all commands */
```

```
"tso_cmd"
```

```
...
```

```
"CONSOLE DEACTIVATE"
```

## Using Pipelines with Rexx

- PIPE is a command that accepts *stage* commands as operands
  - ▶ Stages separated by a character called a *stage separator*
    - Default char is vertical bar | (**x'4F'**)
  
- Allows you to combine programs so the output of one serves as input to the next
  - ▶ Like pipes used for plumbing: data flows through programs like water through pipes!
  
- User-written stages are Rexx programs
  - ▶ Reads in data, works on it, places it back into pipe

## Using Pipelines with Rexx - Examples

- Invoking from CMS command line:

```
pipe < profile exec | count lines | console
```

- Invoking from an Exec:

```
/* Count number of lines in exec */  
'PIPE < profile exec | count lines| console'
```

```
/* or ... on multiple lines */  
'PIPE < profile exec',  
    '| count lines',  
    '| console'
```

## Using Pipelines with Rexx - Examples

- Invoking commands and parsing output into a stem:

```
'pipe',  
'CMS LISTFILE * EXEC A', /* issue cmd */  
  '| SPECS 1 1 ,          /* parse first word */  
  '| STEM response.'     /* save in stem */  
  
do i = 1 to response.0  
  say response.i          /* display file names */  
end
```

## Control Constructs – DO...END

DO ... END can be used to create a code block

```
if wins > losses then
  do
    say 'Congratulations!'
    say 'You have won!'
  end
else say 'Sorry, you have lost'
```

## Control Constructs - Selection

```
if wins > losses then say 'you have won'  
    else say 'you have lost'
```

```
select  
    when wins > losses then say 'winner'  
    when losses > wins then say 'loser'  
    otherwise say 'even'  
end
```

```
select  
    when wins > losses then say 'winner'  
    when losses > wins then say 'loser'  
    otherwise NOP  
end
```



## Control Constructs – DO Loops

```
do forever
  say 'You will get tired of this'
end
```

```
do 3
  say "Roll, Roll, Roll the dice"
end
```

```
do i=1 to 50 by 1
  say i
end
```

## More DO Loops

```
i=30
do until i > 21      /* Evaluate after DO executes */
  i=i+5
end
say i                → 35
```

```
i=30
do while i < 21     /* Evaluate before DO executes */
  i=i+5
end
say i              → 30
```

## Iterate, Leave, and Exit

- **Iterate** causes a branch to end of control construct

```
do i=1 to 4
  if i=2 then iterate
  say i
end
```

1, 3, 4

- **Leave** exits the control construct and continues the REXX program

```
do i=1 to 4
  say i
  if i=3 then leave
end
say 'I'm free!'
```

1, 2, 3  
I'm free!

- **Exit** exits the REXX program unconditionally

```
i=1
do forever
  say i
  if i=3 then exit
  i=i+1
end
say 'I'm free!'
```

1, 2, 3

## Built-In Functions

`ABS (-1.674)` → `1.674`

`/* absolute value */`

`C2D ('a')` → `129`

`D2X (129, 2)` → `'81'`

`/* char to decimal, dec to hex*/`

`DATATYPE ('10.5', 'W')` → `'0'`

`DATATYPE ('12 ')` → `'NUM'`

`/* determines if a string matches a provided type */`

`DATE ('U')` → `'05/24/12'`

`/* date function */`

`LENGTH ('abcdef')` → `6`

`/* length of the string */`

## Built-In Functions

`POS('day', 'Wednesday')` → `7`

`/* starting position of substr inside a string */`

`RIGHT('12', 4, '0')` → `'0012'`

`/* pad 12 out to 4 characters with 0's */`

`SUBSTR('abcdef', 2, 3)` → `'bcd'`

`/* obtain substring of 3 characters beginning at second character */`

`WORDS('are we done yet?')` → `4`

`/* return number of tokens inside a given string */`

`WORDPOS('the', 'now is the time')` → `3`

`/* return position of a given substring */`

`/* inside a string */`

## Subroutines & Procedures

- **CALL** instruction is used to invoke a routine
  - ▶ May be an internal routine, built-in function, or external routine
- May optionally return a result

### **RETURN expression**

- ▶ variable **result** contains the result of the expression
- Parameters may be passed to the called routine

### **CALL My\_Routine parm1**

...which is functionally equivalent to the clause:

### **NewData = My\_Routine(parm1)**

- Variables are global for subroutines, but not known to procedures unless passed in or EXPOSE option used

## Subroutine Example: Returning a Value

```
/* subroutine call example */
x = 5
y = 10
Call Calc x y          /* call subroutine Calc */
If result > 50 Then
    say "Perimeter is larger than 50"
Else
    say "Perimeter is smaller than 50"
exit

Calc:                  /* begin subroutine      */
Parse Arg len width   /* input args          */
return 2*len + 2*width /* calculate perimeter */
                      /* ...and return it   */
```

## Exercise 4: WHATCP EXEC

- Write REXX program WHATCP EXEC to show z/VM CP Level information
  - ▶ Issue CP command **QUERY CPLEVEL** to display CP level
  - ▶ Use **REXX DIAG** function to issue **QUERY CPLEVEL** command
    - **Parse command output** to display CP Version, Release, and Service level



## Exercise 5: MYDISKS EXEC

- Write a Rexx program to show which disks your userid has accessed
  - ▶ **Call a subroutine** that
    - ▶ Uses a PIPE to **issue** CMS command **QUERY DISK** and **save** response
    - ▶ **Determine** the number of disks accessed
    - ▶ **Return** the value to the main routine
  - ▶ **Display** the returned number of disks accessed
  - ▶ **Display** each of the disks that are accessed
  - ▶ **Issue** the CMS command **QUERY DISK** without using a PIPE
  - ▶ **Verify** that output from Steps 3 and 4 match

## Exercise 4: WHATCP – Answer

```
/* Display CP Level information for the z/VM system */
```

```
'CP QUERY CPLEVEL'
```

```
Parse value diag(8,'QUERY CPLEVEL') with ,  
          . . version . release . ',' . . servicelvl .
```

```
say 'z/VM Version = ' version
```

```
say 'z/VM Release = ' release
```

```
say 'Service Level = ' servicelvl
```

## Exercise 5: MYDISKS EXEC – Answer #1

```
/* Find Number of disks accessed and list them */
Call GetDisks
Say 'This user has' NumDisks 'disks accessed.'
Say '  '

Do i = 1 to Numdisks
  Say DiskList.i
End

Say '  '
ADDRESS CMS
'QUERY DISK'
Exit

/* Subroutine: Get list of disks and return number of disks accessed*/
GetDisks:
  'PIPE',
  'CMS QUERY DISK',
  '| Drop 1',
  '| STEM DiskList.'
  NumDisks = DiskList.0
Return NumDisks
```

## Exercise 5: MYDISKS EXEC – Answer #2

```
/* Find Number of disks accessed and list them */
Call GetDisks
Say 'This user has' NumDisks 'disks accessed.'
Say '  '

Do i = 1 to Numdisks
  Say DiskList.i
End

Say '  '
ADDRESS CMS
'QUERY DISK'
Exit

/*Subroutine: Get list of disks and return number of disks accessed*/
GetDisks:
  'PIPE',
  'CMS QUERY DISK',
  '| Drop 1',
  '| STEM DiskList.',
  '| count lines',
  '| var NumDisks'
Return NumDisks
```

## Sample Program: GETTMODE EXEC

- Rexx program GETTMODE locates the first unused file mode (A-Z) and creates a temporary disk at that file mode
  - ▶ Illustrates usage of many Rexx features covered in this workshop
    - ▶ Subroutine
    - ▶ Issuing commands
    - ▶ Building and parsing strings
    - ▶ Built-in functions
    - ▶ Stems
    - ▶ Pipelines
    - ▶ Displaying output

## Sample Program: GETTMODE EXEC

- Logic:
  - **Calls subroutine** that:
    - ▶ Uses a PIPE to **issue** CMS command **QUERY SEARCH** to obtain the used modes (file mode is 3<sup>rd</sup> word of response); **saves it in a stem**
    - ▶ **Builds a string** of used modes from the **output stem** of the PIPE
    - ▶ **Creates a string** of possible file modes (A-Z)
    - ▶ **Builds a stem** containing the possible file modes
    - ▶ **Marks** the used file modes "unavailable" in the list of possible modes
    - ▶ **Locates** the first available mode and **returns** it to the main program
  - **If a file mode is returned:**
    - ▶ **Issues commands** to define and format a temporary disk at the returned mode

## Sample Program: GETTMODE EXEC (1 of 3)

```
/* Get temporary disk space and access it at an available file mode */
'CP DETACH 555'          /* Get rid of old disk */

/* Call subroutine Findmode to locate the first available file mode. */
/* Once found, define a temporary disk and format and access it at */
/* the returned file mode. */

Call Findmode

If rtnmode <> 0 Then
  Say 'Temp disk will be accessed at mode' rtnmode
Else
  Do
    Say 'No Filemodes available for temp disk'
    Exit 8
  End

'CP DEFINE T3390 555 2' /* Define 2 cylinders of temp space */

queue 1                 /* Answer YES to FORMAT prompt */
queue TMP555           /* Disk label is TMP555 */
'FORMAT 555 'rtnmode    /* Format the disk for CMS files */

Exit rc
```

## Sample Program: GETTMODE EXEC (2 of 3)

```
/* Subroutine Findmode will locate the first available (A-Z) file mode.*/
/* and return it in variable rtnmode. If no file modes are available, */
/* rtnmode will be set to zero. */
Findmode:
  'PIPE',
  'CMS QUERY SEARCH',
  '| SPEC WORDS 3 1',
  '| STEM usedmode.'

/* Build string of accessed file modes */
acc_modes = ''
Do I = 1 TO usedmode.0
  acc_modes = acc_modes || SUBSTR(usedmode.I,1,1)
END

/* Build stem containing all possible file modes */
possible_modes = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Do i = 1 TO 26
  modelist.i = SUBSTR(possible_modes,i,1)
End

/* Remove all accessed file modes from possible file mode list */
mlength = LENGTH(acc_modes)
Do n = 1 TO mlength
  Do i = 1 TO 26
    If (SUBSTR(acc_modes,n,1) = modelist.i) Then
      Do
        modelist.i = ' '
      Leave
    End
  End
End
End
```



## Sample Program: GETTMODE EXEC (3 of 3)

```
/* Locate the first possible file mode that is "available" and      */  
/* return it                                                         */  
  
foundmd = 'NO'  
Do i = 1 TO 26  
  If modelist.i ^= ' ' Then  
    Do  
      rtnmode = modelist.i  
      foundmd = 'YES'  
      Leave  
    End  
  End  
End  
  
/* If no file modes available, return zero                          */  
  
If foundmd = 'NO' Then  
  rtnmode = 0  
  
Return
```

## Sample Program: GETTMODE EXEC – Pipelines Only

```
FINDMODE:  procedure

'Pipe',
'  literal A B C D E F G H I J K L M N O P Q R S T U V W X Y Z',
'| Split ',
'| Spec 1.1 13',
'| Append CMS Q disk *',
'| Nlocate 8.4 /VDEV/',
'| Spec 13.1',
'| Sort ',
'| Unique Single ',
'| Take 1',
'| Var freefm'
```

## For More Information...

- **Websites:**

- ▶ <http://www.ibm.com/software/awdtools/rexx/> Rexx webpage
- ▶ <http://www.ibm.com/software/awdtools/netrexx/> Netrexx
- ▶ <http://www-01.ibm.com/software/awdtools/rexx/opensource.html> Object Rexx
- ▶ <http://regina-rexx.sourceforge.net/> Regina Rexx

- **z/VM publications:**

- ▶ Rexx/VM Reference - SC24-6113
- ▶ Rexx/VM User's Guide - SC24-6114
- ▶ website for library downloads: <http://www.vm.ibm.com/library/>

- **z/OS publications:**

- ▶ TSO/E Rexx User's Guide - SC28-1974
- ▶ TSO/E Rexx Reference - SC28-1975
- ▶ website for library downloads: [http://publibz.boulder.ibm.com/cgi-bin/bookmgr\\_OS390/Shelves/IKJOSE10?filter=rexx](http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/IKJOSE10?filter=rexx)

- **Rexx Compiler**

- ▶ Products ordered separately from z/VM:
  - REXX/370 Compiler, 5695-013
  - REXX/370 Library, 5695-014

- **Other books:**

- ▶ [The Rexx Language](#) ISBN 0-13-780651-5
- ▶ [The Netrexx Language](#) ISBN 0-13-806332-X

- **List servers:**

- ▶ <http://listserv.uark.edu/scripts/wa.exe?A0=ibmvm>



John Franciscovich  
z/VM Design and Development  
francisj@us.ibm.com  
+1 607.429.3574

[Brian W. Hugenbruch](#), CISSP  
z/VM Security Design and Development  
[bwhugen at us dot ibm dot com](#)  
+1 607.429.3660  
[@Bwhugen](#)