Intermediate REXX

Rich Smrcina
22 July 2026

Overview

- References
- This class continues from ‘Introduction’
- We will cover additional REXX topics
  - External functions
  - DIAG/DIAGRC
  - Interpret
  - Program stack
  - Value conversion
  - Date management
  - Interfacing with
    - System Commands
    - XEDIT
    - CMS Pipelines

VELOCITY SOFTWARE

Reference Materials

REXX/VM User's Guide

- Phenomenal resource for learning REXX

REXX/VM Reference

- The paper version of HELP REXX MENU

CMS Pipelines User's Guide and Reference

IBM Training Digital Learning Platform

- https://learn.ibm.com - Free course for learning the basics of REXX

REXX Language Association

- https://rexxla.org

Open Object REXX

- http://www.oorexx.org

Regina REXX

- https://regina-rexx.sourceforge.io

VELOCITY SOFTWARE

External Functions

- An *internal* function is contained within the REXX program
- An *external* function is in a separate REXX program file
- Both are called exactly the same way

```
/* MYNAME EXEC */
Return 'Capt. Obvious'
```

```
/* EXAMP25 EXEC */
Say 'The time is' Time()
Say 'My name is' Myname()
Exit
```

```
examp25
The time is 14:03:31
My name is Capt. Obvious
```

DIAG/DIAGRC

- The DIAG variants can be used to issue some CP Diagnose functions from REXX
  - Diag 8 to issue a CP command

```
Parse Value Diag(8,'CP QUERY USERID') with uid . sysid '15'x
```

Parses the CP QUERY USERID response into variables uid and sysid; ‘15’x is always included on the end of each returned line

Interpret

- Execute dynamically built REXX instructions

```
Interpret expression
```

- Expression is interpreted as REXX code and then executed
- If you need to execute commands based on the contents of a variable, use Interpret

Interpret - Examples

```
/* TEST1 EXEC */
Interpret 'c = 4*3'
say c
Exit
```

```
test1
12
```

```
/* TEST2 EXEC */
b = 8
c = 'This is number'
Interpret 'Say c b'
Exit
```

```
test2
This is number 8
```

Program Stack

- `Push` - puts an item on to the stack (LIFO)
- `Queue` - puts an item on the bottom of the stack
- `Pull` – retrieves the next item from the top of the stack
- `Queued()` - returns the number of items on the stack

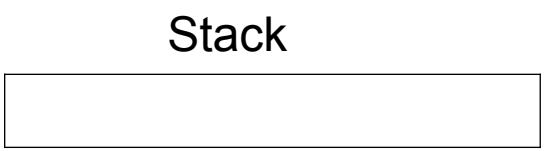
Data is always ‘Pull’-ed from the top of the stack

Program Stack

- A data structure that is used to temporarily store data
- Can be used by multiple programs in the same virtual machine
- A number of languages implement a similar structure
  - C++, Java, Python, PHP
- A piece of data is ‘queue’-d or ‘push’-ed onto the stack
- When data is on the stack, it can be ‘pull’-ed from the stack
- A stack can contain many different data items
- Depending upon how you use the stack it is FIFO or LIFO
- Push puts data onto the top of the stack
  - If you always use Push, the stack is LIFO
- Queue puts data onto the bottom of the stack
  - If you always use Queue, the stack is FIFO

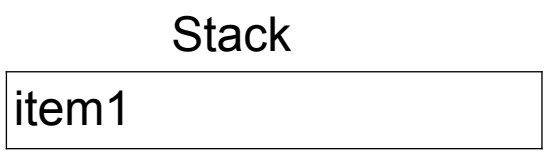
Program Stack

Push ‘item1’



Program Stack

Push ‘item1’



Program Stack

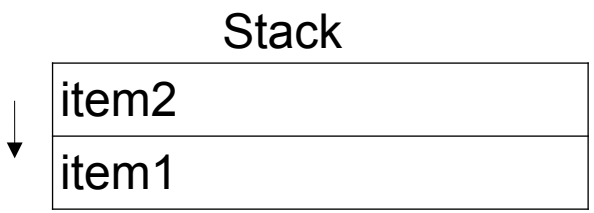
Push ‘item2’

Stack

item1

Program Stack

Push ‘item2’



Program Stack

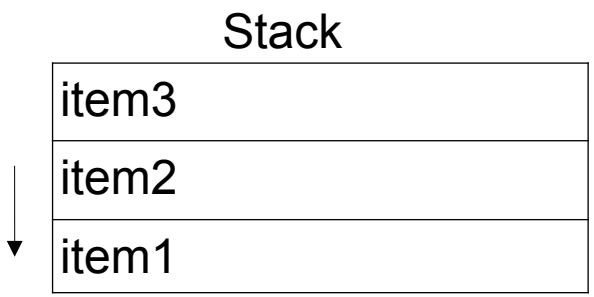
Push ‘item3’

Stack

item2
item1

Program Stack

Push ‘item3’



Program Stack

Stack

item3
item2
item1

```
/* TEST1 EXEC */
Push 'item1'
Push 'item2'
Push 'item3'
Say Queued()
```

```
test1
3
Ready; T=0.01/0.01 09:47:28
Unknown CP/CMS command
Unknown CP/CMS command
Unknown CP/CMS command
```

When your program ends, CMS will think anything on the stack is a command, and try to execute it

Program Stack

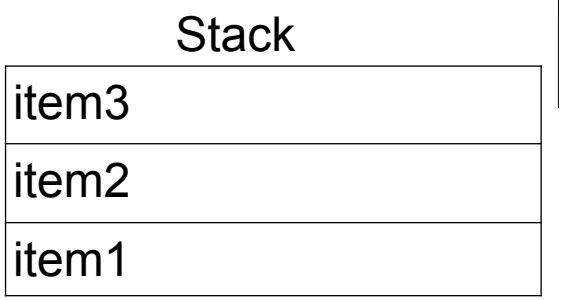
Use DROPBUF to delete the program stack

```
/* TEST1 EXEC */
Push 'item1'
Push 'item2'
Push 'item3'
Say Queued()
'DROPBUF'
```

```
test1
3
Ready; T=0.01/0.01 09:47:28
```

Program Stack

Pull fromstack



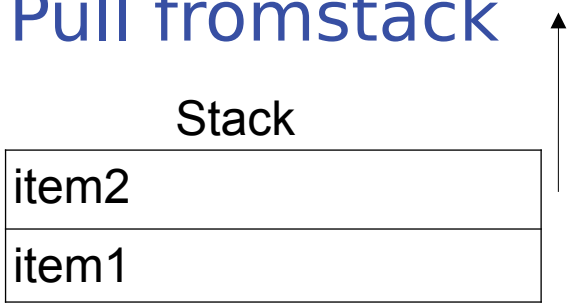

```
/* TEST1 EXEC */
Push 'item1'
Push 'item2'
Push 'item3'
Say Queued()
Pull fromstack
Say fromstack
```

```
test1
3
ITEM3
```

Writing a REXX program

Program Stack

Pull fromstack




```
/* TEST1 EXEC */
Push 'item1'
Push 'item2'
Push 'item3'
Say Queued()
Do Queued()
  Pull fromstack
  Say fromstack
End
```

```
test1
3
ITEM3
ITEM2
ITEM1
```

VELOCITY SOFTWARE

Quiz!

In the program stack example on the previous slide, why are the values from the stack displayed in upper case?

What is the function that tells us how many items are on the program stack?

Quiz!

In the program stack example on the previous slide, why are the values from the stack displayed in upper case?

The Pull instruction translates to upper case. Use Parse Pull to respect case.

What is the function that tells us how many items are on the program stack?

Queued()

Value conversion

Useful for changing values

- Between bases (2-10-16)
- Display hex values

Helpful when working with data in varying formats

Value conversion

Binary

b2x	To hex

Character

c2x	To hex
c2d	To decimal

Decimal

d2c	To character
d2x	To hex

Hex

x2b	To binary
x2c	To character
x2d	To decimal

Value conversion c2d - Examples

```
/* TEST1 EXEC */
c = '6e'x
Say c2d(c)
```

```
test1
110
```

```
/* TEST2 EXEC */
c = 'fffd'x
say c2d(c)
```

```
test2
65533
```

```
/* TEST3 EXEC */
c = 'fffd'x
say c2d(c,2)
```

```
test3
-3
```

```
/* TEST4 EXEC */
c = 'fffd'x
say x2b(c2x(c))
```

```
test4
1111111111111101
```

Value conversion – Other examples

```
/* TEST1 EXEC */
c = 20
x = d2x(c)
Say c '=' x
```

```
test1
20 = 14
```

```
/* TEST2 EXEC */
c = 20
x = d2x(c)
b = x2b(x)
Say c '=' x '=' b
```

```
test2
20 = 14 = 00010100
```

From running code...

```
rflgs = x2b(c2x(substr(record,1,1)))
rtype = c2d(substr(record,2,1))
```

Date management

- REXX provides a very powerful date function
- It can return a date in many different formats
- It can also convert the date from one format to another
- HELP REXX DATE provides a lot of information

Date management

Date Format options

B	Base date (number of complete days since 01 Jan 0001)
C	Day number in the current century
D	Day number in the current year
J	Julian date in the format yyddd
E	European date dd/mm/yy
M	Full name of the current month
N	Date in format dd mon yyyy
O	Ordered Date (suitable for sorting) yy/mm/dd
S	Standard Date (suitable for sorting) yyyymmdd
U	USA format (mm/dd/yy)
W	Returns weekday name

Date management

Various date formats

```
/* TEST1 EXEC */
say 'Normal...' Date()
say 'Standard.' Date('S')
say 'Normal...' Date('N')
say 'European.' Date('E')
say 'Ordered..' Date('O')
say 'Julian...' Date('J')
say 'Base.....' Date('B')
```

```
test1
Normal... 15 Jul 2026
Standard. 20260715
Normal... 15 Jul 2026
European. 15/07/26
Ordered.. 26/07/15
Julian... 26196
Base..... 739811
Ready; T=0.01/0.01 10:15:49
```

Date management

Converting between formats

`date(output-format, input-date, input-format, output-sep, input-sep)`

```
00001 /* TEST1 EXEC */
00002 d1 = '2024-02-26'  /* Standard with '-' sep */
00003 d2 = '13*12*07'    /* European with '*' sep */
00004 d3 = '21 Aug 2007' /* Normal */
00005 say date('S', d1, 'S', '/', '-')
00006 say date('S', d2, 'E', '-', '*')
00007 say date('B', d3, 'N')
```

```
test1
2024/02/26
2007-12-13
732908
Ready; T=0.01/0.01 10:37:05
```

VELOCITY SOFTWARE

Date management

Base date give us a very powerful tool to determine how many days have elapsed

```
/* TEST1 EXEC */
d1 = date('B', '21 Aug 2007', 'N')
d2 = date('B', '21 Jul 2026', 'N')
say d2-d1
```

```
test1
6909
Ready; T=0.01/0.01 10:45:22
```

Interfacing with system commands

- System commands are enclosed in quotes
  - Unless part of the command is a variable
- Messages are displayed on the console

```
/* TEST1 EXEC */
'CP QUERY RICH'
say rc
```

```
test1
HCPCQV003E Invalid option - RICH
3
```

Interfacing with system commands

- For CP commands, messages can be suppressed
- Use SET EMSG OFF to turn off error message displays
- The current setting of EMSG is displayed with ‘CP Q SET’

```
/* TEST1 EXEC */
'SET EMSG OFF'
'CP QUERY RICH'
say rc
```

```
test1
3
```

The problem with this method is that EMSG is off after the EXEC ends

Interfacing with system commands

- DIAGRC provides a return code and a condition code from a diagnose function
- Dependent on the diagnose function
- No need to SET EMSG

```
/* */
rc = DIAGRC(8, 'CP QUERY RICH')
say rc
Parse Var rc retcode .
say retcode
```

```
test1
        3 0     HCPCQV003E Invalid option - RICH

3
```

Interfacing with system commands

- CMS Pipelines can also issue commands
- PIPE will ‘eat’ the error message and still pass on rc

```
/* TEST1 EXEC */
'PIPE CP QUERY RICH | hole'
say rc
```

```
test1
3
```

Interfacing with VMLINK

- VMLINK can be easily used in a REXX program
- Accessing a minidisk or directory
- Get the filemode that VMLINK assigns
- After done, release the filemode

Interfacing with VMLINK

```
/* TEST1 EXEC */

Address CMS 'VMLINK RICH 195 ( STEM RESP. .FM'
'PIPE STEM resp. | console'
```

```
test1
DMSVML2060I RICH 195 linked as 0121 file mode R
RKSDEV195 .FM R
Ready; T=0.01/0.01 13:17:00
```

Interfacing with VMLINK

```
/* TEST1 EXEC */
Address CMS 'VMLINK RICH 195 (NOTYPE STEM RESP. .FM'
Parse Var resp.1 '.FM' fmode .
say 'Filemode='fmode
'PIPE CMS RELEASE' fmode
```

```
test1
Filemode=R
Ready; T=0.01/0.01 13:30:37
```

resp.1=

```
RKSDEV195 .FM R
```

Interfacing with CMS Pipelines

- Read a file into the pipeline
- Write it to a stem (array)
- Produce a count of the records
- Pipelines enforces the number of elements in a stem is in the .0 element

```
/* TEST1 EXEC */
'PIPE < ZOSTRCE 12041338 Q',
  '| stem smfrecs.'
say smfrecs.0
```

```
test1
2567
Ready; T=0.01/0.02 14:50:56
```

Interfacing with CMS Pipelines

- Refining the pipe slightly

```
/* TEST1 EXEC */
'PIPE < ZOSTRCE 12041338 Q',
  '| spec 2.1 c2d 1',
  '| pick w1 == /110/',
  '| count lines',
  '| console'
```

```
test1
1939
Ready; T=0.01/0.02 14:50:56
```

Interfacing with XEDIT

- Implemented in an XEDIT macro
- A file of REXX code and XEDIT commands
- Commands are passed to the XEDIT environment
- Values can be obtained (extracted) from the XEDIT environment
  - Over 100 items can be extracted
- Panel displays can be created
  - Not unlike ISPF or CICS

Interfacing with XEDIT

- Everytime XEDIT is entered a profile is executed
- The default is PROFILE XEDIT
- Your XEDIT profile can be used to tailor your environment

```
/* */
'CURLIN ON 4'
'SCALE ON 3'
'V 1 72'
'NULLS ON'
'CASE M I'
'NUM ON'
```

Interfacing with XEDIT

- An XEDIT profile can also be used to do useful tasks with REXX
- To use an XEDIT profile reference it in the XEDIT command

```
/* TEST2 EXEC */
Address CMS 'XEDIT TEST1 EXEC A (PROFILE CHKTEST1'
```

For reference, here is TEST1 EXEC

```
00001 /* TEST1 EXEC */
00002 'PIPE < ZOSTRCE 12041338 Q',
00003   '| spec 2.1 c2d 1',
00004   '| pick w1 == /110/',
00005   '| count lines',
00006   '| console'
```

Interfacing with XEDIT

- The best way to think about this is to imagine being in an XEDIT session
- Issuing XEDIT commands, checking responses

```
00001 /* CHKTEST1 XEDIT */
00002 '/PICK'
00003 say rc
00004 'QQUIT'
```

Interfacing with XEDIT

- When we execute our TEST2 it will invoke the XEDIT profile
- Attempt to locate the text
- Display the return code
- Quit the XEDIT session

```
00001 /* CHKTEST1 XEDIT */
00002 '/PICK'
00003 say rc
00004 'QQUIT'
```

```
test2
2
DMSXDC546E Target not found
Ready; T=0.01/0.01 13:39:10
```

Interfacing with XEDIT

- Change the XEDIT profile
  - Do not display error messages
  - Display my own message if rc = 2

```
00001 /* CHKTEST1 XEDIT */
00002 'SET MSGMODE OFF'
00003 '/PICK'
00004 If rc = 2 Then
00005   Say 'PICK not found'
00006 'QQUIT'
```

```
test2
PICK not found
Ready; T=0.01/0.01 13:46:01
```

Interfacing with XEDIT

- Next step
  - Ignore case
  - Display the line number on which the target is found
    - Use EXTRACT to get the current line number

```
00001 /* CHKTEST1 XEDIT */
00002 'SET MSGMODE OFF'
00003 'SET CASE MIXED IGNORE'
00004 '/PICK'
00005 If rc = 2 Then
00006   Do
00007     Say 'PICK not found'
00008     'QQUIT'
00009     Exit
00010   End
00011 'EXTRACT /LINE'
00012 say 'PICK was found on line' line.1
00013 'QQUIT'
```

```
test2
PICK was found on line 4
Ready; T=0.01/0.01 13:58:18
```

Quiz!

What command can be used in an XEDIT macro to get information about the current XEDIT session?

Which element of a REXX stem is used to contain the number of elements in that stem?

Quiz!

What command can be used in an XEDIT macro to get information about the current XEDIT session?

EXTRACT

Which element of a REXX stem is used to contain the number of elements in that stem?

0

Questions?

Homework time!

Assignment 1

Write a REXX function to

- Get the EMSG setting from ‘CP Q SET’
- Reverse the setting
  - If it is ON, turn it OFF
  - If it is OFF, turn it ON
- Display the new value of EMSG

Hint: DIAG8, PIPE and Parse will be very useful

Assignment 2

Write a REXX program to

- Put a list all of the EXECs on your A disk into a stem
  - Including the last referenced date
  - Hint: the ISODATE option of LISTFILE
    - LISTFILE * EXEC A (ISODATE
- Walk through the stem, calculate how old each exec is in days
- Write the filename and age back to the console

Hint: PIPE, a Do loop, and the date function will be useful

Send an email to homework@velocitysoftware.com when done

Send your homework program to the HOMEWORK user

‘SENDFILE fileid HOMEWORK’

VELOCITY SOFTWARE

Introduction to REXX

Thank you for attending!

Training, Education and Information

https://velocitysoftware.com/educate/

Presentations and Handouts

https://velocitysoftware.com/pdfs/