An Article about

R E X X

on

VSE/ESA 2.1

September 11, 1995

Bernd Dowedeit

IBM Laboratory Boeblingen,Germany
REXX/VSE Development

Contents

The Procedure Language REXX on VSE/ESA 1
REXX/VSE is More Than Just Another Language 1
What makes REXX Easy to Learn and to Use ? 2
What makes REXX So Powerful ? 2
REXX Can Be Customized to Your Needs 2
Close Relationship between REXX and VSE Job Control 3
Loading and Calling VSE Batch Programs 4
VSE Command Environment 5
POWER Command Environment 6
Automate Your VSE Console 6
Where You Find More Details 7

The Procedure Language REXX on VSE/ESA

Bernd Dowedeit

REXX on VSE has first been generally available at September 1993 as an optional program product for VSE/ESA 1.3.2. The first REXX/VSE release implemented the SAA REXX language level 3.48 and had only basic VSE-specific support. Its main focus was to provide the pure REXX language together with a TSO/E REXX compatible execution environment. Hence, REXX/SAA and REXX/TSO applications can easily be ported to REXX/VSE.

Since April 1995, REXX/VSE is available for all VSE/ESA 2.1 customers and is integrated closely into the VSE/ESA 2.1 central functions. It now provides many powerful VSE unique functions that make it easy to work with VSE/ESA and helps to automate repetitive tasks.

The most recent enhancement is the introduction of the REXX Console Automation that has been made generally available via the PTF UN80995 on top of VSE/ESA 2.1. and will be integrated into VSE/ESA 2.1.1. It exploits the new VSE/ESA console support and makes it easy to write REXX console applications for automating the operation of your VSE/ESA console. In addition, an console application framework is available to demonstrate how you can use REXX console automation to your best advantage.

REXX/VSE is More Than Just Another Language

REXX is a powerful general purpose language that is easy to learn. It is for users ranging from beginners to computer professionals.

The main purpose of REXX is to act as intermediary between the user and the underlying processing system. REXX on VSE provides easy access to VSE/ESA resources and subsystems and, therefore, enables you to automate and make more productive the operation of your VSE/ESA 2.1 system. REXX/VSE consists of three parts:

The REXX Kernel is known as the SAA Procedure Language REXX and is common on all IBM processing systems.

The REXX/370 Library provides fast performance of compiled REXX programs. The runtime library is common on MVS, VM and VSE. Compilation of a REXX/VSE program must be done on VM or MVS.

VSE System Dependent Interface is extensively enhanced for VSE/ESA 2.1 and provides access to VSE/ESA resources and subsystems:

- New dimension of Job Control Language capabilities
- Console automation
- Console Application Framework
- Batch job Management capabilities
- Full integration of VSE batch services such as LIBR or IDCAMS.
- Easy-to-Use I/O functions for VSE library and SAM files.

What makes REXX Easy to Learn and to Use ?

- **Familiar Syntax and Easy Typing**. REXX's syntax is similar to natural language. Names for variables and constants can be up to 250 bytes long, so you can give meaningful data names. Free formatting and uppercase and lowercase in any combination enhances readability. REXX also provides language constructs such as SELECT, assorted DO loops and IF-THEN-ELSE branching.
- **Automatic Data Handling**. Everything is a characterstring, including numbers. Storage managing is performed automatically. Therefore, no data declaration is required.
- **Superior Array Handling**. REXX uses compound variables called stems for easy handling of data structures. A stem provides a superior form of arrays, because a stem index could be any character string.

What makes REXX So Powerful ?

- **A Rich Set of Built-In Functions**. Many of the built-in functions operate on words or strings. A string can be of any length. Other built-in functions do arithmetic or data conversion or provide system information.
- **Built-In Parsing with Many Options**. You can parse strings into words or into sections by matching tokens, numeric character positions, or string patterns.
- **Extensive Trace and Interactive Debug**. You can trace all instructions as they are processed and see all results or intermediate results that they produce. With interactive debug, processing pauses after each instruction until you press Enter to go on to the next instruction. Interactive trace can for example be displayed on the a CICS based or on the OS/2 user console that is provided by the VSE/ESA Interactive Interface.
- **Command Building**. You can build strings into commands that REXX sends to the underlying operating system or application environment. Such environments are called host command environments. The various VSE related host command environments make it possible to automate the VSE/ESA operation.
- **External Functions**. Many VSE unique external functions are imbedded into the REXX language and provide a simple access to the VSE processing system.

REXX Can Be Customized to Your Needs

There are REXX Programming and Customizing Services that make REXX flexible to **fit into different execution environments** and permit you to extend the capabilities of the REXX programming language by your own external functions and commands.

REXX/VSE Customizing Services The customizing services let you change the processing environment that defines how REXX accesses and uses system services. These are, for example, services for loading and freeing a REXX program, performing I/O, obtaining and freeing partition storage, and handling data stacks. These services are called replaceable routines. There are also various exit routines and a module called ARXPARMS that defines the environment characteristics, such as default STDIN/OUT data stream, which function packages can be used, which host command environments can be used or the search order for locating commands, external functions and subroutines.

REXX/VSE Programming Services The programming services let you interface with the REXX language processor. Programs written in Assembler, COBOL, PL1 or C can use application programming interfaces in order to call the services that can, for example, be used to call a REXX program or to access the REXX variable pool from an external function or command environment.**:**

Close Relationship between REXX and VSE Job Control

REXX works closely together with VSE Job Control and offers a new dimension of Job Control capabilities.

How to Run a REXX Program: The easiest way to invoke a REXX program is shown in the following example:

```
// EXEC REXX=REXXWAIT
```

It runs the REXX program **rexxwait** that resides in a VSE library as member type PROC.

The next example runs the REXX program **rexxdom** and passes a parameter list using the PARM statement and the size of the program area using the SIZE parameter. The REXX program parses the parameter list as one single argument.

```
// EXEC REXX=REXXDOM,SIZE=(ASMA90,50K),PARM='LINK'
```

In-Storage JCL Procedures: When the REXX program exits it can create an in-storage JCL procedure containing a sequence of JCL statements and have them executed by VSE JCL right after the REXX program has completed. The REXX program simply queues the JCL statements on the data stack and exits with return code zero. Normal VSE JCL procedure rules apply to the REXX created in-storage JCL procedure.

Job Control Command Environment: You need not to exit the REXX program in order to execute JCL commands. REXX provides the Job Control command environment (ADDRESS JCL) that let you issue JCL commands from a REXX program. One JCL statement is executed at a time. The return code RC is set to zero if the JCL statement is executed successfully otherwise RC contains a meaningful error code. All JCL informational messages and decision messages (normally written to SYSLOG) are routed back to the REXX program. They are written into the REXX stem defined by the OUTTRAP function. **The Job Control command environment permits combination of the full set of modern REXX language constructs with a wide range of frequently used JCL commands** That permits, for example, to loop through the execution of JCL commands.

Scan the MAP Command Output: There are JCL statements that produce output records normally written on SYSLOG, such as MAP. The following example executes the MAP command and scans the output lines for the starting address of a given partition id.

```
ARG pid    /* Input string such as BG or F5 */
posArea= 8; lenArea= LENGTH(pid); posAddr=36; lenAddr= 7
call OUTTRAP lines.   /* Define the output stem */
ADDRESS JCL "MAP"
do i=1 TO lines.0     /* Scan for given PID in the output stem */
   if STRIP(SUBSTR(lines.i,posArea,lenArea)) = pid then
    do
     addr = STRIP(SUBSTR(lines.i,posAddr,lenAddr))
     say "Partition" pid "starts at address" addr
     EXIT 0           /* Success, found the PID */
    end
end
EXIT 8                /* Nothing found */
```

Create a Phase Using a REXX Program: There are JCL statements that read data from SYSIPT, such as the SET SDL statement or the INCLUDE statement. REXX/VSE provides an equivalent to VSE data input stream SYSIPT. It is called REXXIPT. **REXXIPT** is a REXX function and defines a stem that contains the input data records. REXXIPT data stream is transparent for VSE JCL and for any other VSE batch program called by REXX (as you will read later in this article).

```
 ....
rc = Assemble_and_LinkEdit('ARXVERFY','PRD1.BASE':hp2)
if rc = 0 then
   ADDRESS LINK arxverfy                      /* Execute phase        */
 ....
EXIT
Assemble_and_LinkEdit:                        /* Create the phase     */
                                              /* Name of assembler pgm */
arg mod_name,libname                          /* ... and sublibrary   */
call REXXIPT rexxipt.                         /* Open input stream    */
EXECIO '* DISKR' libname"".'""mod_name""'.A',/* Read assembler program*/
                        '(STEM rexxipt. FINIS'/* ... from VSE library  */
ADDRESS JCL
'// OPTION CATAL'                             /* Open syslink         */
ADDRESS LINK 'ASSEMBLY'                       /* Assemble program     */
if rc = 0 then
  do
    '// LIBDEF PHASE,CATALOG='libname         /*Phase will be cataloged*/
    ADDRESS LINK                              /*... into VSE library  */
    'LNKEDT MSHP,AMODE=31,RMODE=24'           /* LINK-EDIT program    */
  end
return rc
```

Loading and Calling VSE Batch Programs

The REXX CALL instruction can call a program that resides as a phase in a VSE library. However, this program **must have implemented the REXX EFPL** (REXX External Function Parameter List). Existing VSE batch programs do not. Therefore, REXX/VSE provides the LINK command environment.

Call VSE Batch Programs: The LINK command environment allows to call VSE batch utilities such as LIBR, IDCAMS, DITTO, ASSEMBLER, LNKEDT or MSHP. You can also call your own VSE batch program.

To load and call a program specify the name of the program followed by the parameter list you want to pass to the program. The LINK command environment is the REXX equivalent to the JCL EXEC statement as you will see in the following example.

```
* A NORMAL VSE JOB CONTROL INVOCATION OF IDCAMS
// EXEC IDCAMS,PARM='MARGINS(1 80)'
LISTCAT CLUSTER
/*
```

The above JCL invocation of the VSAM LISTCAT command can be performed by a REXX program as follows:

```
/* A REXX invocation of IDCAMS                                */
CALL OUTTRAP output.                /* Open output...         */
CALL REXXIPT sysipt.                /* ...and input data stream */
sysipt.0 = 1
sysipt.1 = 'LISTCAT CLUSTER'        /* Specify IDCAMS command(s)*/
ADDRESS LINK 'IDCAMS MARGINS(1 80)'    /* Invoke IDCAMS          */
```

Full REXX Input/Output Routing for IDCAMS and LIBR: Look at the previous IDCAMS example (you can write similar REXX programs to execute LIBR commands).

The REXX function REXXIPT defines the REXX stem ***sysipt*** as input data stream for the program called by ADDESS LINK, in this case the VSAM utility IDCAMS. Then you assign the IDCAMS commands to the REXX stem and invoke IDCAMS. For a batch program, such as IDCAMS, it is totally transparent to read from the REXX stem ***sysipt*** rather then from the actual SYSIPT device.

If your REXX program wants to analyze the result of the LISTCAT command it uses the OUTTRAP function to specify the name of the REXX stem ***output*** that will contain all output records generated by the LISTCAT command.

Call COBOL, PL1 or C Programs as Subroutines: Sometimes, you want to call a COBOL, PL1 or a C program from a REXX program as a subroutine passing multiple parameters, and you expect this subroutine to have the input parameters modified on return to the REXX program. You can do this using the LINKPGM command environment.

For the LINKPGM environment, you can pass multiple parameters to the called program. Variable substitution is performed on the parameters you specified and the value of each parameter is passed as argument to the program. The called program can update the arguments it receives and return the updated values. These are automatically stored into the associated REXX variables.

Assume the example program **mypgm** modifies the first parameter to the string 'Hello' and the second parameter to'World'.

```
v1 = '     '; v2=v1             /* Initialize the variables */
say v1 v2                       /* Will display blanks      */
ADDRESS LINKPGM 'MYPGM v1 v2'
say v1 v2                       /* Will display the string "Hello World" */
```

VSE Command Environment

There are REXX/VSE commands that perform **data stack services** providing a versatile data stack handling. Data stacks can be shared by different REXX programs (in opposite to REXX variables that cannot). Examples for VSE commands are NEWSTACK that allows to create a stack of stacks or MAKEBUF that encapsulates stack segments within a specific stack. There are also commands that provides the number of stacks, buffers or queue elements.

The **EXECIO** command gives you access to VSE resources in an ease to use and very powerful way. EXECIO can read or write data on the program stack or into a REXX stem directly. You can use EXECIO for I/O task such as copying information to or from a file to add, delete or update information.

EXECIO can read from SYSLST, write from SYSIPT or READ/WRITE from/into a SAM file or a VSE library member. Following example reads an entire VSE library member into the program stack just with one command:

```
'EXECIO * DISKR PRD1.BASE.REXXTABL.Z (FINIS'
```

The next EXECIO example stores all stem variables of the stem ***data*** into a sequential file (SAM file). Assume there is a DLBL such as '// DLBL SYS023,'MY.OUTPUT.FILE' that defines the output file:

```
'EXECIO * DISKW SYS023 (FINIS STEM data. BLKSIZE 120 RECFORM FIXBLK RECSIZE 120'
```

POWER Command Environment

The POWER host command environment exploits the VSE/POWER spool-access service requests GET, PUT and CTL. It allows easy access to VSE/POWER resources from a REXX program.

ADDRESS POWER commands include:

- **GETQE** command, which retrieves a VSE/POWER LST, PUN or RDR entry into a REXX stem.
- **PUTQE** command, which places the contents of a REXX stem or library member into a VSE/POWER LST, PUN or RDR.
- **QUERYMSG** command, which returns job completion messages into a REXX stem.
- **VSE/POWER commands** such as 'D RDR,ALL' or 'S RDR,00C' and many more.

Delete all Listings that Contain the Word "confidential": The following example displays all LST queue entries, retrieves one after another and deletes the LST queue entry if it contains the word "confidential".

```
call outtrap lstq.
ADDRESS POWER
'D LST,ALL'                /* Query LST queue entries          */
do i = 2 to lstq.0         /* For each queue element....       */
  parse var lstq.i . . . jobname jobnum .
  'GETQE LST JOBNAME' jobname 'JOBNUM' jobnum 'STEM PRTLINES.'
  IF Confidential() THEN   /* Seach for confidential, if found */
     'L LST,'||jobname||','||jobnum    /* Delete Q elem    */
end
exit
Confidential:
do line = 1 to prtlines.0
  if wordpos('CONFIDENTIAL',translate(prtlines.line))¬=0 then return 1
end
return 0
```

New PUTQE/GETQE Options: The PUTQE and GETQE commands have many options. New options are the FORMAT parameter to specify printer control characters, the DESTUSER and DESTNODE parameters to specify a userid and pnet node for a remote printer, and the NOGENCM parameter to suppress generation of job completion messages.

Job Management Capabilities: The POWER command environment provides job management capabilities. The QUERYMSG command can be used to manage asynchronously running batch jobs. You can submit or release POWER jobs from a REXX program in order to have such a batch job generate a job completion message that the QUERYMSG command will later retrieve. In this case you just put the job into the RDR queue from a REXX stem or library member. But you can also submit the job and let REXX wait for the execution and completion of the job. You will get back job information such as the job's return code or if not completed the reason, e.g. class is busy, job has canceled. If you don't know which VSE/POWER class to run specify a list of possible classes. REXX will try and select the next available class.

Automate Your VSE Console

There is a separate article about the REXX Console Automation Capabilities included in VSE/ESA 2.1.

Where You Find More Details

For a detailed description of REXX/VSE refer to following manuals

- VSE/REXX Reference, SC33-6642
- VSE/REXX User's Guide, SC33-6641
- VSE/REXX Console Automation, SC33-6598