VM/VSE Technical Conference - Orlando - May 2000

Session E83

REXX/VSE and Automated Operations

Klaus-Dieter Wacker
VSE Development
IBM Germany

RETURN TO INDEX

Trademarks / Disclaimers

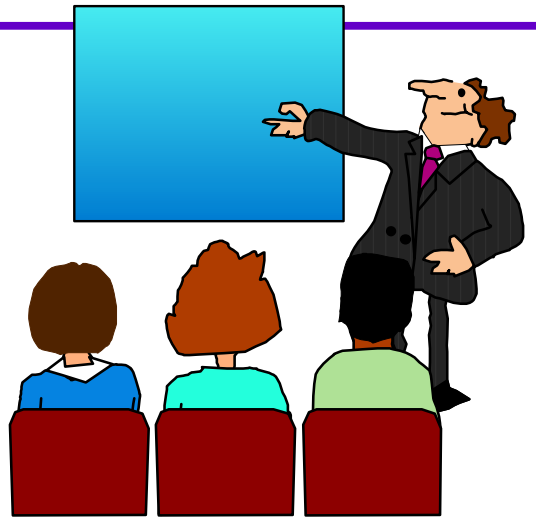
The following are trademarks of the International Business Machines Corporation:

OS/390
VM/ESA
VSE/ESA

The information contained in this document is distributed on an "as is" basis without any warranty either express or implied. The customer is responsible for use of this information and/ or implementation of any techniques mentioned. IBM has reviewed the information for accuracy, but there is no guarantee that a customer using the information or techniques will obtain the same or similar results in its own operational environment.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM’s licensed program may be used; any functionally equivalent program may be used instead.

Agenda7



- Introduction
- Input / Output to disk
- JCL command environment
- LINK command environment
- POWER command environment
- CONSOLE command environment
- Enhancements (incl. SOCKET support)

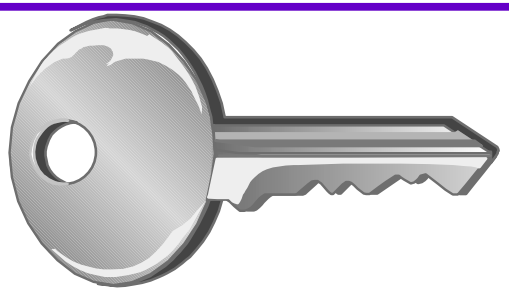
REXX/VSE Language

- General purpose programming language
  - SAA REXX language level 3.48
- VSE dependent functions and commands
- subset of TSO/E REXX compatible functions
- TSO/E REXX compatible programming and customizing services
- common REXX/370 Library on VSE, OS/390 and VM for compiled REXX programs

VSE/ESA Versions and REXX/VSE

VSE/ESA Version	Components	Compid	CLC
1.3 / 1.4	REXX interpreter	5686-058-00	DC7
1.3 / 1.4	REXX/VSE Interface	5686-058-01	DC7
1.3 / 1.4	REXX/370 Library	5686-058-02	DC7
2.2 / 2.3 / 2.4	REXX interpreter + VSE interface	5686-066-16	15I / 35I / 45I
2.2 / 2.3 / 2.4	REXX/370 Library	5686-066-12	15I / 35I / 45I

Customers use REXX/VSE to...

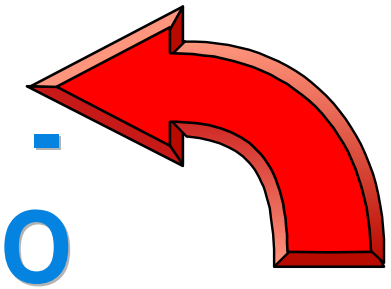


- clean-up POWER LST queue
- automate night shift processing
- automate startup (start partitions, open files, ...)
- automate shutdown (close files, terminate applications, ...)
- scan datasets (libraries, SAM-files, VSAM-files, POWER queues)
- scan hardcopy file
- monitor VSE/ESA system
- ...

VSE unique REXX-Functions

- I/O function for VSE library and sequential data sets
- Increase of JCL capabilities
  - NETVIEW, DITTO support REXX
- Invocation of VSE batch programs
- POWER batch job management and spool support
- REXX Console Automation

VSE Command Environment - Input/Output to disk: EXECIO



Example:

Create job *REXXJOB* and store it into library member *PRD2.EXEC.REXXJOB.JOB.*

- VSE library members
  - * fixed length records of 80 bytes
  - * 1 variable length record (dumps, phases)
- sequential files
- SYSIPT and SYSLST

```
job.0 = 6
job.1 = '* $$ JOB JNM=REXXJOB,CLASS=4'
job.2 = '// JOB REXXJOB'
job.3 = '// LIBDEF *,SEARCH=(PRD2.EXEC)'
job.4 = '// EXEC REXX=RXPGM1'
job.5 = '/&'
job.6 = '* $$ EOJ'
EXECIO job.0 'DISKW PRD2.EXEC.REXXJOB.JOB',
             'STEM job. FINIS'
```

Input/Output to disk: EXECIO

Example:

Insert a string after 100 bytes

Library member with logical record format "string":

- **Parameter BYTES**
- **Parameter STRTBYTE**

only since VSE/ESA 2.3.

```
'EXECIO 1 DISKRU SYSDUMP.BG.DBG00000.DUMP (STEM dump. ' ,
  'BYTES 100 STRTBYTE 101 OPEN'
dump.1 = ' REXX_CHANGE ' || dump.1
'EXECIO 1 DISKW SYSDUMP.BG.DBG00000.DUMP (STEM dump. ',
  'BYTES 113 FINIS'
```

Input/Output to disk: ASSGN

If the ASSGN function assigned STDOUT/STDIN to disk, the following instructions can access data from disk:

- SAY

- PULL

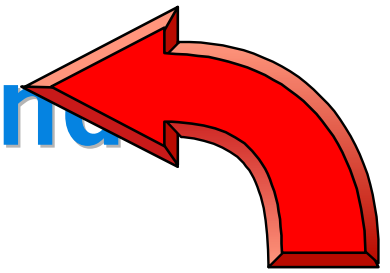
- TRACE

Example:

Read input from a SAM (sequential) file.

```
ADDRESS JCL "// DLBL NEWIPT,'some.file.id',,VSAM"
ADDRESS JCL "/*"
saved_value = ASSGN(STDIN,NEWIPT)
'EXECIO 0 DISKR NEWIPT (OPEN ',
   'BLKSIZE 80 RECSIZE 80 RECFORM FIXBLK'
pull dataline
say 'The line read from sequential file follows:'
say dataline
```

Job Control Command Environment

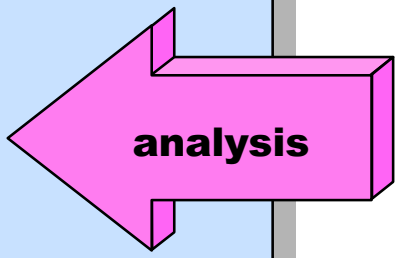


- **Host command environment:**
  - *ADDRESS JCL 'jcl_command'*
- Imbeds VSE JCL statements into REXX program
- One JCL statement at a time is executed
- One string - No continuation characters required
- Return codes
- Output data is routed back to REXX via *OUTTRAP*
  - command output
  - JCL error/informational messages
- Input data may be provided via *REXXIPT*

ADDRESS JCL Example: Analyse MAP command

This REXX program looks for the partition start address of the given partition id *pid*. It issues a JCL *MAP* command and captures the *MAP* output into stem *LINES*.

```
ARG pid
posArea = 8; lenArea = LENGTH(pid)
posAddr = 36; lenAddr = 7
CALL OUTTRAP lines.
ADDRESS JCL "MAP"
DO i=1 TO lines.0
  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
  END
END
```



ADDRESS JCL Example: Using SETPARM

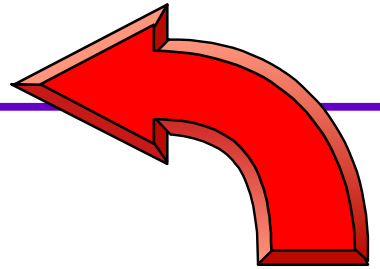
This REXX program invokes SETPARM to set a symbolic parameter.

```
ARG input
ADDRESS JCL "//SETPARM TODAY=" || input
```

If the new value should also be valid for the higher JCL level on return, it must be invoked as:

```
// EXEC REXX=SETP,PARM='MONDAY',TODAY
/*
* Parameter TODAY is now 'MONDAY'
```

ADDRESS LINK + ADDRESS LINKPGM



- Host command environments:
  - *ADDRESS LINK 'pgmname parmlist'*
  - *ADDRESS LINKPGM 'pgmname var1 var2 ...'*
- Loads and calls a non-REXX program
  - Loads *pgmname* from active PHASE chain into GETVIS or program area
  - Program defined in table *ARXEOJTB*
- *ADDRESS LINK* has same parameter list convention as *// EXEC PGM=pgmname,PARM='parmlist'*
- *ADDRESS LINKPGM* provides standard parameter list "Called by reference" which is used by PL/1, COBOL and C
  - Parameter list passes multiple REXX variables
  - Allows called program to update the REXX variables

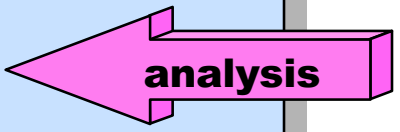
ADDRESS LINK Example: Invoking LIBR

This REXX program exploits the LINK environment to execute LIBR commands. They are supplied via REXXIPT function; their output records are trapped via OUTTRAP function.

```
ARG sublib
CALL OUTTRAP libr_output.
CALL REXXIPT libr_input.
libr_input.0 = 2
libr_input.1 = 'ACC S='||sublib
libr_input.2 = 'LD ARX*.PHASE'

ADDRESS LINK 'LIBR'

IF Word_Found('ARXINIT')
  THEN SAY 'REXX/VSE was installed into' sublib
EXIT
```

ADDRESS LINK Example... Invoking LIBR

```
Word_Found:
 ARG search_name
 DO line = 1 to libr_output.0
  IF WORDPOS(search_name,TRANSLATE(libr_output.line)) ¬= 0
   THEN RETURN 1
 END
 RETURN 0
```



LIBR output example

```
   DIRECTORY DISPLAY          SUBLIBRARY=PRD1.BASE          DATE: 96-03-20
                                                            TIME: 08:56

   ---------------------------------------------------------------------------
    M E M B E R      CREATION    LAST      BYTES     LIBR CONT SVA   A- R-
   NAME     TYPE     DATE       UPDATE    RECORDS    BLKS STOR ELIG MODE
   ---------------------------------------------------------------------------
   ARXIC    PHASE    95-11-28    -   -       22 B       1  YES YES   31 ANY
   ARXIDCAM PHASE    95-11-28    -   -     1048 B       2  YES YES   24  24
   ARXINIT  PHASE    95-11-28    -   -   306576 B     311  YES YES   31 ANY
      ...
```

ADDRESS LINK Example: Invoking IDCAMS

This REXX program exploits the LINK environment to execute IDCAMS commands. They are supplied via REXXIPT function; their output records are trapped via OUTTRAP function.

```
ARG infile catalog
ADDRESS JCL
"// DLBL PRINTFL,'" || infile || "',,VSAM,CAT=" || catalog
"/*"
CALL OUTTRAP idcams_output.
CALL REXXIPT idcams_input.
idcams_input.0 = 1
idcams_input.1 = 'PRINT INFILE (PRINTFL) DUMP'
ADDRESS LINK  IDCAMS MARGINS(1 80)'
/* file content now in stem idcams_output. */
EXIT RC
```

initializations

invocation

Example: Using DITTO

This REXX program exploits the DITTO-REXX-Interface to write and read a VSAM file.

More information in: DITTO/ESA User's Guide and Reference SH19-8221-01

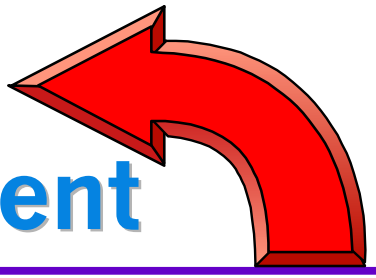
```
DO i=1 TO 10
  invsam.i = right(i,7,'0') || ' this is record' i
END
invsam.0 = 10
CALL 'DITSETUP'
ADDRESS DITTO
/* write the file */
'DITTO $XV DSNOUT=USCHI.TESTFILE.ESDS UCAT=VSAM.MASTER.CATALOG' ,
      'VARNAME=invsam. REUSE=YES'
/* read the file into stem outvsam. */
'DITTO $VX DSNIN=USCHI.TESTFILE.ESDS UCAT=VSAM.MASTER.CATALOG' ,
      'VARNAME=outvsam.'
```

Example: Assemble and LNKEDT Source Code

This REXX program exploits the JCL and LINK environment to assemble a *source program* and linkedit the phase.

```
ARG mod_name                                          /* Assembler program */
libname = PRD2.PROD                                   /* Sublibrary        */
CALL REXXIPT rexxipt.                                 /* Open input stream */
EXECIO '* DISKR' libname||'.'||mod_name||'.A (STEM rexxipt. FINIS'
ADDRESS JCL '// OPTION CATAL'                         /* Open SYSLINK      */
ADDRESS JCL ' PHASE' mod_name||',*'
ADDRESS LINK 'ASSEMBLY'                               /* Assemble program  */
IF rc = 0 THEN DO
 ADDRESS JCL '// LIBDEF PHASE,CATALOG='libname /*Catalog sublib   */
 ADDRESS LINK 'LNKEDT MSHP,AMODE=31,RMODE=24' /* LINK-EDIT program*/
END
IF rc = 0 THEN ADDRESS LINK mod_name                  /* Execute phase     */
EXIT rc
```

POWER Command Environment



- Host command environment:
  - *ADDRESS POWER 'power_command'*
- Available commands:
  - *GETQE* - gets entry from VSE/POWER queues
  - *PUTQE* - puts entry into VSE/POWER queues
  - VSE/POWER commands
  - *QUERYMSG* - retrieves job completion messages
- Return codes
- Output data is routed back to REXX via *OUTTRAP*
  - command output
  - job completion messages
  - error message(s):
    - VSE/POWER messages
    - VSE/POWER SAS Return and Feedback codes

POWER Command Environment: Extensions for GETQE

- support for Printer Control Characters (LST queue only)
  - *FORMAT, CTRLREC*
- retrieval of segmented output queue entries
  - *JOBSUF*
- retrieval of only part of a queue entry
  - *STARTREC, RECORDS, STARTPAG, PAGES*
- retrieval of the corresponding FCB-image phase
  - *FCB*
- conversion from ASA to MCC
  - *ASATOMCC*

POWER Command Environment: Extensions for PUTQE

- appending data to an output queue entry
  - *FIRST, NEXT, LAST*
- starting a new segment for an output queue entry
  - *JOBSUF, NEWSEGM*
- change of destination user and/or destination node
  - *DESTUSER, DESTNODE*
- further attributes for an output queue entry
  - *FORMAT, PRIORITY, COPIES, DISP, USERINFO, OPTB, FCB, FLASH_COUNT, FLASH_NAME*
- suppression of job completion messages (RDR queue only)
  - *GENCM / NOGENCM*
- creating a RDR queue entry with record length 128
  - *LONGREC*

ADDRESS POWER Example 1

This REXX program deletes all LST queue entries with TO-user=HUGO that contain the word CONFIDENTIAL in the first 10 lines.

```
CALL OUTTRAP lstq.

ADDRESS POWER
'PDISPLAY LST,TUSER=HUGO'      /* query LST queue entries */
'SETUID HUGO'
DO i = 2 to lstq.0
 PARSE VAR lstq.i . jobname jobnum . . cl .
 'GETQE LST JOBNAME' jobname 'JOBNUM' jobnum 'CLASS' cl ,
       'STARTREC 1 RECORDS 10 STEM prtlines.'
 IF Confidential() THEN
  'PDELETE LST,'||jobname||','||jobnum    /* delete entry */
END
EXIT
```

ADDRESS POWER Example 1 ...

```
/* CONFIDENTIAL: Check first 10 lines of the LST file for */
/*               the blank delimited word CONFIDENTIAL    */
/*               Return 1 if found, otherwise return 0    */

Confidential:
  DO line = 1 TO prtlines.0
    IF WORDPOS('CONFIDENTIAL',TRANSLATE(prtlines.line)) ¬= 0
    THEN RETURN 1
  END
  RETURN 0
```

ADDRESS POWER Example 2

This REXX program consolidates PUN queue entries with the same job name into one big entry, deleting all of the smaller ones.

```
ARG name .
SETUID 'HUGO'
fc = OUTTRAP(punq.,,concat)
ADDRESS POWER
'PDISPLAY PUN,'||name       /* query punch queue entries */
DO i = 3 to punq.0 - 1; append.i = 'NEXT'; END
j = punq.0; append.j = 'LAST'
append.2 = 'FIRST'
newnum = 'newnum'
DO i = 2 to punq.0
  PARSE VAR punq.i . jobname jobnum .
  'GETQE PUN JOBNAME' jobname 'JOBNUM' jobnum 'STEM oldpun.'
  'PUTQE PUN' append.i 'STEM oldpun. JOBNAME' name 'JOBNUM' newnum
  'PDELETE PUN,'||jobname||','||jobnum   /* delete entry */
END
```

ADDRESS POWER Example 3

This REXX program creates a LST queue entry with 2 segments in 4 steps.

```
ADDRESS POWER
'PUTQE LST JOBNAME MYPUTQE JOBNUM jobnum FIRST CLASS Q' ,
  'STEM text1.'
'PUTQE LST JOBNAME MYPUTQE JOBNUM' jobnum 'NEXT CLASS Q' ,
  'STEM text2. NEWSEGM segnum'
/* here starts the new segment */
'PUTQE LST JOBNAME MYPUTQE JOBNUM' jobnum 'NEXT CLASS Q' ,
  'STEM text3. JOBSUF' segnum+1
'PUTQE LST JOBNAME MYPUTQE JOBNUM' jobnum 'LAST CLASS Q' ,
  'STEM text4. JOBSUF' segnum+1
```

ADDRESS POWER Example 4

This REXX program submits a job and awaits its execution. The job runs a utility program (EXEC LIBR) whose output is retrieved and processed.

```
ARG name
class = 'YZ'
CALL OUTTRAP out.,,noconcat
ADDRESS POWER
'PUTQE RDR WAIT 999 MEMBER A.B.LIBR.JOB',
      'JOBNAME jobname JOBNUM jobnum CLASS class'

IF rc = 0 THEN DO
  'GETQE LST STEM job_output.',
        'JOBNAME' jobname 'JOBNUM' jobnum 'CLASS' class
  IF Check_Job_Output(name)
    THEN SAY 'Member' name 'found in Directory'
END
ELSE IF rc > 0
  THEN SAY 'EXEC LIBR return code is:' rc
  ELSE SAY 'EXEC LIBR job has not been completed'
```

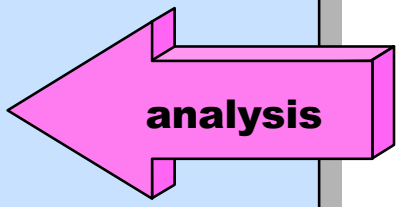
initializations

job submission

analysis

ADDRESS POWER Example 4 ...

```
/* Check_Job_Output: Check every line of the job output for the */
/*                   blank delimited word passed via argument.  */
/*                   Return 1 if found, otherwise return 0      */

Check_Job_Output:
       ARG looking_for

       DO line = 1 TO job_output.0
         IF WORDPOS(looking_for,TRANSLATE(job_output.line)) ¬= 0
           THEN RETURN 1
       END
       RETURN 0
```



ADDRESS POWER Example 5

This REXX program provides a simple job manager. VSE/POWER jobs are submitted and their asynchronous execution is controlled until completion.

```
ARG number_of_jobs
SETUID 'USER'
CALL OUTTRAP msg.,,noconcat
ADDRESS POWER
DO i=1 TO number_of_jobs          /* start jobs */
  'PUTQE RDR MEMBER A.B.JOB'i'.JCL'
END
```

initializations

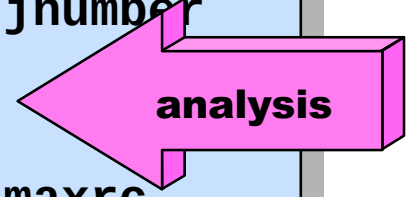
job submission

Job completion message:

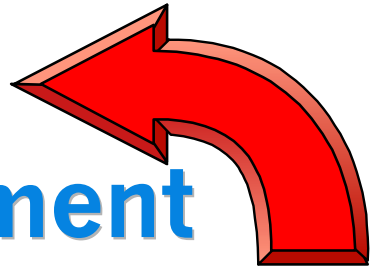
```
ARX0970I JOB REXXCAT 02546 EXECUTED NODE=LOCAL DATE=03/26/96

         TIME=10:23:19 MAXRC=0000 LASTRC=0000
```

ADDRESS POWER Example 5 ...

```
DO WHILE number_of_jobs ¬= 0
  CALL SLEEP 5                          /* wait for messages */
  'QUERYMSG KEEP'                      /* Query for messages */
  IF rc = 0 THEN
    DO i=1 TO msg.0          /* Look for completion message */
      IF WORD(msg.i,1) = 'ARX0970I' THEN DO
        PARSE VAR msg.i . . jname jnumber . . . . maxrc .
        IF SUBSTR(jname,1,3) = 'JOB' THEN DO
          'QUERYMSG DELETE JOBNAME' jname 'JOBNUM' jnumber
          IF maxrc = '0000'
            THEN SAY 'JOB' jname 'run successfully'
            ELSE SAY 'JOB' jname 'failed with rc= 'maxrc
          number_of_jobs = number_of_jobs - 1
        END
      END
    END
END
```

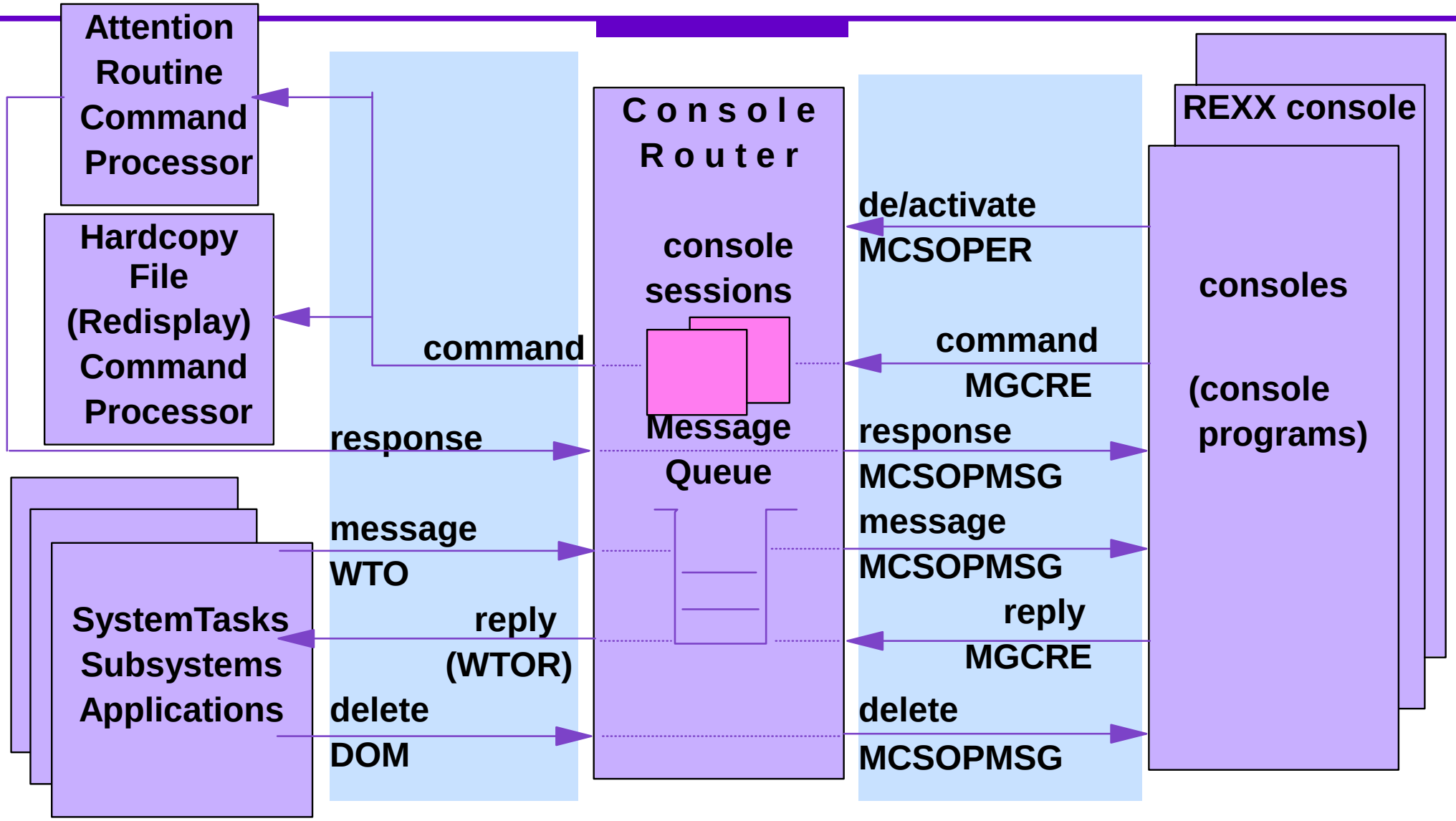


CONSOLE Command Environment



- Host command environment:
  - ADDRESS CONSOLE 'console_command'
- General available with VSE/ESA 2.1.1
  - REXX Console Command Environment
  - External function package ARXEFCO
  - Console Application Framework
- Benefit: automate interactive communication with the operator

VSE/ESA and REXX Consoles



REXX/VSE Console Commands

- Activate/Deactivate console sessions
  - ACTIVATE NAME rexx1 PROFILE rexx
  - DEACTIVATE rexx1
- Switch between console sessions
  - CONSWITCH rexx2
- Associate a CART to a Console Session
  - CART 'abcd'
- Query for current Console Session
  - CONSTATE NAME consname PROFILE profname
    CART cart
- VSE Console commands

REXX Console Profiles

- REXX
  User console that receives messages due to ECHO parameter, WTO console name or command responses
- REXALLRC
  Master console that receives command responses and all routing codes
- REXNORC
  Master console that receives command responses but no routing codes
- REXAUTO
  Master console that receives only automated messages. That is for example VSE/OCCF messages normally sent to NetView

REXX Console Functions

- ***GETMSG***

  retrieves a command response or a console message

  ```
  fc = GETMSG(msgstem,msgtype,cart,mask,time)
  ```

- ***FINDMSG***

  Retrieves VSE console messages until a find criteria is fulfilled

  ```
  msg = FINDMSG('ARX0960E ERROR',10,'ZONE 9 22')
  ```

- ***SENDMSG***

  Sends a message to a specific console

  ```
  CALL SENDMSG 'Hello REXX console','REXX',,'HIGH'
  ```

- ***SENDCMD***

  Issues a console command or a reply without active console session

  ```
  CALL SENDCMD 'PFLUSH BG'
  ```

- ***OPERMSG***

  Enables/Disables operator communication exit

ADDRESS CONSOLE Example

This REXX program finds out whether CICS is up and running. If not, CICS is restarted. A VTAM command is issued and its output analysed.

```
ADDRESS CONSOLE                    /* Switch to Console Environment */
'ACTIVATE NAME rexx PROFILE rexnorc' /* Activate Console session */
'CART VTM'
'D NET,APPLS'                      /* VTAM command shows APPLIDs    */
result = GETMSG(vtam_msg.,'RESP','VTM',,30)
                                   /* Wait for VTAM messages        */
IF result = zero THEN
  DO i=1 TO vtam_msg.0             /* Analyse the VTAM messages     */
    pos = INDEX(vtam_msg.i,'PRODCICS')
    IF pos > zero THEN
      IF WORD(SUBSTR(vtam_msg.i,pos),2) ¬= 'ACTIV' /* CICS down? */
      THEN 'R RDR,PRODCICS'        /* Release CICS                  */
  END
'DEACTIVATE rexx'
```

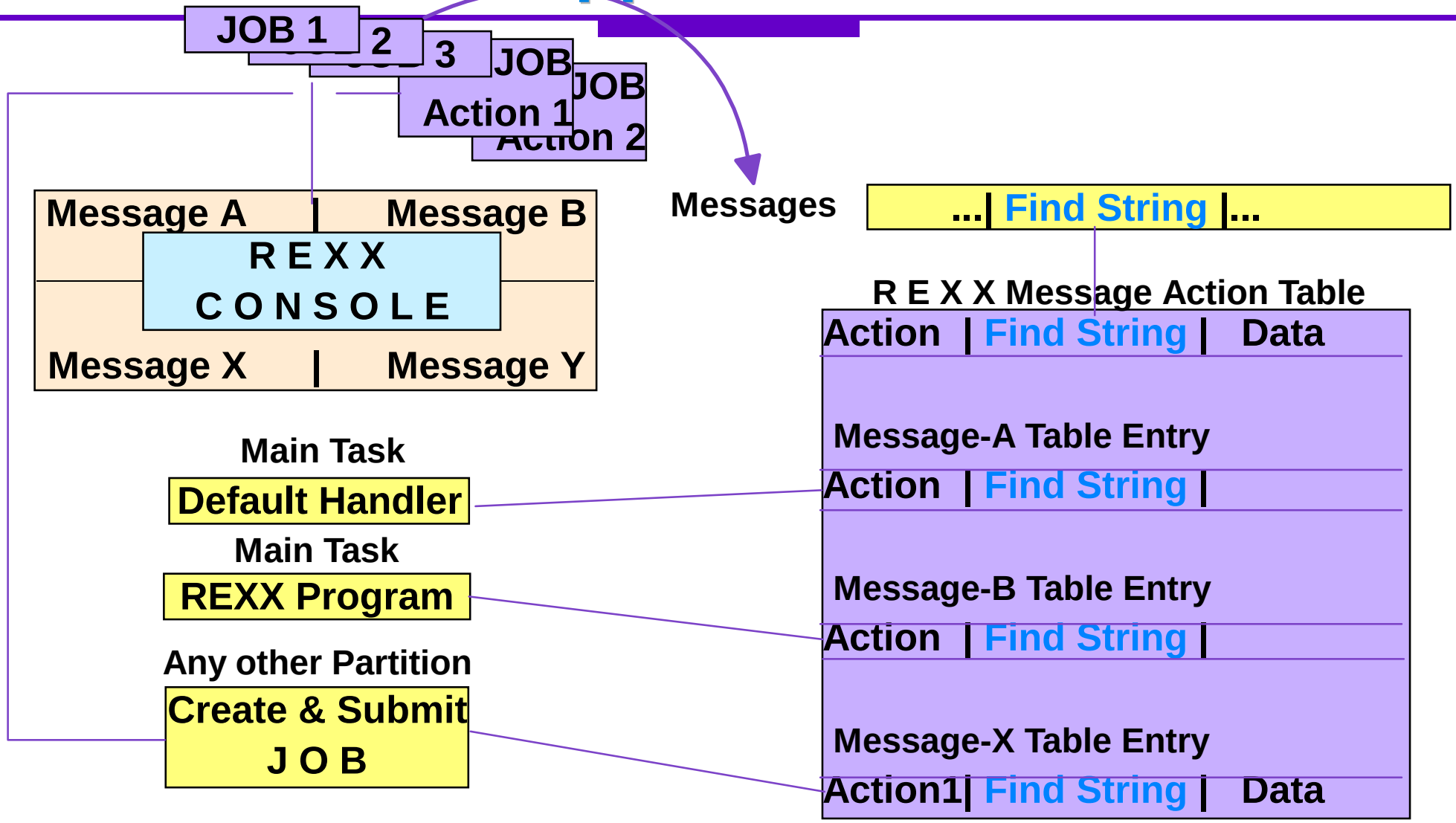
REXX/VSE CPU Monitor *REXXCPUM*

- checks for critical performance values in VSE partitions
  - CPU time
  - CPU rate
  - elapsed time
  - I/O count
  - I/O rate
- issues console messages, if user-defined limits have been exceeded

```
ARX0998I PID Y1 JOB TEST EXCEEDS THE LIMITS:
CPUTIME=10.15
```

- Limit definition with REXX function SYSDEF
- CICS background transaction IEXM
  (VSE/ESA Monitor Activity service)

CONSOLE Application Framework



Sample Message Action Table

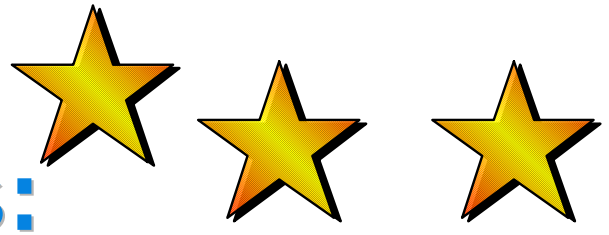
The following sample is shipped with REXX in PRD1.BASE.REXXTABL.Z .

```
/* ____________________________________________________________________
*/
/*     REXX/VSE Message-Action Table for Demonstration Purposes       */
/*                                                                    */
/* Action  Find_String                  (Data, in continuation lines) */
/* ____________________________________________________________________
*/
FLUSH1Q52I 1Q52I
FLUSHCPU   ARX0998I
REPLY      HIT ENTER TO CLOSE THE FILE
REPLY      HALT EXIT REACHED, PRESS ENTER TO END
&REXXSTOP  THIS LOOP CAN BE STOPPED
*          REXXCO<< VSAM WORKFILE:                                    X
           INNAME=PRD1.BASE.SKRXVSAM.Z OUTNAME=PRD1.BASE.REXXSPACE.JOB
X
           VAR001=REXXSPCE VAR002=PRD2.BASE VAR003=PRD1.BASE.SKRXJCL.Z
```

Skeleton of Console Monitor

The following skeleton is the frame of the monitoring program REXXCO.

```
/*     REXXCO: monitors the console messages
*/
/*             according to a Message Action Table
*/
   ...
/* initializations */
   ...
CALL FINDMSG msgtable,,,'LOADACTN'  /* Load message table in storage
*/
CALL OPERMSG 'ON'                   /* Establish operator communication
*/
ADDRESS CONSOLE
'ACTIVATE NAME' consname 'PROFILE REXALLRC'
DO WHILE processing
  msgdata = OPERMSG('MSGDATA')      /* Check for operator message
*/
   ...                              /* if EXIT, stop processing
*/
  message = FINDMSG(,10)            /* Look for messages within table
*/
```



REXX functions: more enhancements

Function *DATE* :

date format conversions incl. specification of separator characters

```
DATE('U','1.Feb.1998','N','+','.')  →  '02+01+98'
```

Function *SORTSTEM* :

sorting of stem variables in ascending or descending order

```
fc = SORTSTEM(svar.,'ZONE 9 14','DESCENDING')
```

Function *FINDMSG* :

reimplemented to reduce the amount of CPU time needed for its execution

REXX Sockets Application Program Interface

Programming Interface to the TCP/IP protocol layers TCP and UDP

Facilities for socket communication directly from REXX programs via SOCKET function

same invocation syntax and same functionality as in REXX/VM and in the REXX TCP/IP API of OS/390

SOCKET function contained in the external function packag ARXEFSO -
available with UQ37224 / UQ37225

description:
http://www.s390.ibm.com/products/vse/vsehtmls/vserxsoc.h

REXX Sockets Application Program Interface

REXX socket functions are provided to:

- Process socket sets
- Initialize, change, and close sockets
- Exchange data
- Resolve names for sockets
- Manage configurations, options and modes for sockets

SOCKET Example: Mini-Server

This REXX program demonstrates a server-typical sequence of SOCKET calls. A corresponding client program follows.

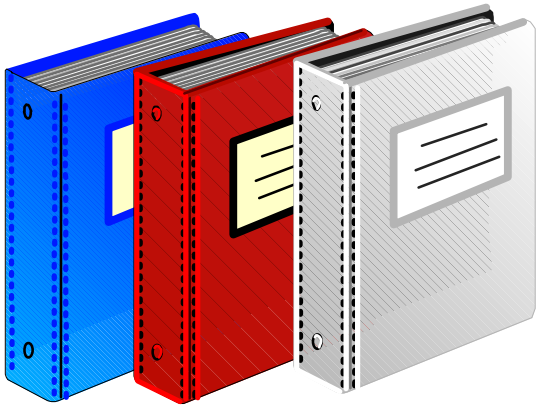
```
fc = SOCKET('INITIALIZE','SERVER')   /* Establish Socket Usage   */
fc = SOCKET('SOCKET','AF_INET','STREAM','TCP')  /* create socket */
parse var fc socket_rc socketid
fc = SOCKET('BIND',socketid,'2 4711 9.164.155.114')
                                /* bind to port 4711            */
fc = SOCKET('LISTEN',socketid)  /* create connection queue       */
client = SOCKET('ACCEPT',socketid)  /* wait for client to connect*/
fc = SOCKET('CLOSE',socketid)   /* no more clients accepted      */
fc = SOCKET('READ',client)      /* receive data from client      */
parse var fc read_rc num_read_bytes read_data
fc = SOCKET('WRITE',client,'some_data')  /* send data to client  */
parse var fc write_rc num_written_bytes
fc = SOCKET('CLOSE',client)     /* terminate communication       */
fc = SOCKET('TERMINATE')        /* terminate Socket Usage        */
```

SOCKET Example: Mini-Client

This REXX program demonstrates a client-typical sequence of SOCKET calls, that fits to the given Mini-Server.

```
fc = SOCKET('INITIALIZE','CLIENT')   /* Establish Socket Usage   */
fc = SOCKET('SOCKET','AF_INET','STREAM','TCP')  /* create socket */
parse var fc socket_rc socketid
fc = SOCKET('CONNECT',socketid,'2 4711 9.164.155.114')
                                     /* connect to server port 4711   */
fc = SOCKET('WRITE',socketid,'some_data') /* send data to server */
parse var fc write_rc num_written_bytes
fc = SOCKET('READ',socketid)    /* receive response from server  */
parse var fc read_rc num_read_bytes read_data
fc = SOCKET('CLOSE',socketid)   /* terminate communication       */
fc = SOCKET('TERMINATE')        /* terminate Socket Usage        */
```

Appendix: Additional Information



- VSE Collection CD/ROM (SK2T-0060-15)
  - REXX/VSE Reference Manual (SC33-6642-03)
  - REXX/VSE User's Guide Manual (SC33-6641-00)
  - VSE User Examples
- Hints and Tips for VSE/ESA
  - http://www.s390.ibm.com/vse/vsehtmls/hintshp.htm
- Publications
  - U. Braun-Krahl: "REXX/VSE Enhancement: REXX Sockets API"
    - http://www.s390.ibm.com/products/vse/vsehtmls/vserxsoc.htm
  - Patrick Talley: "Creating REXX Functions"
    - VSE/ESA Software Newsletter First/Second Quarter, 1999
  - U. Braun-Krahl: "What's New for REXX/VSE in VSE/ESA 2.2 and 2.3"
    - VSE/ESA Software Newsletter Third/Fourth Quarter, 1997
    - http://www.s390.ibm.com/products/vse/vsehtmls/vse9rex.htm
  - B. Dowedeit: "Introduction to REXX/VSE", 09/95
    - http://www.s390.ibm.com/products/vse/vsehtmls/rexx21.htm
  - B. Dowedeit: "REXX/VSE Automated Console Support", 09/95
    - http://www.s390.ibm.com/products/vse/vsehtmls/rexxcon.htm
- REXX/VSE Home Page: http://www.s390.ibm.com/products/vse/rexx/rexxhome.html