REXX APPLICATIONS IN AUTOMATED OPERATIONS

PETE ZYBRICK
FUTURESYS, INC.

REXX Applications in Automated Operations

Pete Zybrick

FutureSys, Inc.
20 Dogwood Trail
Kinnelon, NJ 07405
(201) 492-2777

I. Overview

1. What is Automated Operations? The progressive minimization of computer operator intervention by
    1. Replacing the need for intervention whenever possible by the design and implementation of hardware/software problem determination and correction processes.
    2. Increase problem determination and correction efficiency by filtering and combining only the critical system status information, eliminating redundant and trivial information.

2. Automation Types
    1. Reactive - Event/Response
    2. Proactive - Question/Answer
    3. Administrative/Management

II. Why use REXX

1. Good
    1. PARSE Instruction, especially Literal String
    2. Relatively simple to use/debug/maintain
    3. Relatively easy to create structured code
    4. Function libraries

2. Bad
    1. Simplicity has been oversold by vendors
    2. Unskilled programmers can write bad code in any language
    3. Simplicity masks potential errors
    4. CLIST programmers rarely take advantage of REXX features
    5. Reliance on environment for global variables, poor variable sharing between procedures

III. Features and AO Application

1. Subcom (Host Command Environment Table) - Creating an Environment

    1. Advantages
        1. Speed - commands are directly targeted
        2. No changes to REXX itself are required

    2. Disadvantages
        1. Development - must be written in lower level language, initialization exit configured (MVS) or DLL created (OS/2)
        2. Programmer must remember to use ADDRESS both initially and when switching environments (ie. ADDRESS MVS "EXECIO..." and ADDRESS NETVIEW "GETMLINE..."

2. Shared Variable Interface

    1. Advantages
        1. Large blocks of variables can be created with one command/function
        2. Same basic processing sequence and control block structure on different platforms

    2. Disadvantages
        1. Uses more storage than the stack
        2. Programmers usually forget to DROP, possibly causing storage problems

3. Function Libraries

    1. Advantages
        1. Speed development time and consistency
        2. Can be written in lower level language for improved performance
        3. Can accept and return very large plists
        4. Third party vendors and SHARE

    2. Disadvantages
        1. Definition of requirements
        2. Someone has to write/maintain the functions
        3. Will anyone know they are there?

4. External Programs

    1. Advantages
        1. Can be REXX or load module. Load modules can use the Shared Variable Interface
        2. Interface to external products
        3. Command response/screen capture

    2. Disadvantages
        1. Search time (for load modules, faster to use Subcom and ADDRESS)
        2. Poor global variable handling forces large values to be passed/duplicated between programs

IV. Suggested Methods

Objectives:

1. Keep it simple
2. Minimize redundant coding/maintenance

1. Centralized Routines
    1. Objectives
        1. Maximize the capabilities of the most skilled programmers to produce common 'black box' routines to simplify the most difficult tasks
        2. Maintenance - if the program is broken, it is fixed in one place

    2. Example: NetView returns command responses asynchronously, if at all. Even experienced programmers can have a conceptual problem with async events. Create an external function to serialize command execution/response under NetView, returning the responses on the stack.

```
/* REXX - LINKSTN */
call stkmsgs    ,
     "D NET,ID=someappl,E"  ,     ,
          "IST097I IST075I"  , "IST314I"
. . .  read from stack and process messages  . . .
exit


/* REXX - STKMSGS */
parse arg CmdText , TrapMsgs , EndMsg
"TRAP AND SUPPRESS MESSAGES" TrapMsgs
CmdText
"WAIT 5 SECONDS FOR MESSAGES"
"MSGREAD"
getresps: do while 'EVENT'() = "M"
     "GETMSIZE MAXMLWTO"
     getmlwto: do mlcnt = 1 to maxmlwto
          "GETMLINE CURML" mlcnt
          queue curml
          if 'WORD'(curml,1) = EndMsg then leave
        end /* getmlwto */
     "WAIT CONTINUE"
     "MSGREAD"
   end /* getresps */
return /* stkmsgs */
```

2. Literal String Parsing

Objectives:

1. Parse messages based on text fields to extract variable-length values.

Example: The NetView TSOUSER command describes the status of a TSO user. Display the TSO (application name) and LU of a particular user.

a. Command Format:

"TSOUSER tsologonid"

b. Output:

IST097I DISPLAY ACCEPTED
IST075I VTAM DISPLAY - NODE TYPE = TSO USERID
IST486I NAME=TSOPJZ, STATUS=ACTIV,DESIRED...
IST576I TSO TRACE=OFF
IST262I APPLNAME=TSOA, STATUS = ACTIV
IST262I LUNAME=A01T1234, STATUS=ACTIV
IST314I END

c. Program:

```
/* REXX */
parse upper arg tsoid .
call 'STKMSGS' "TSOUSER' tsoid , "IST097I IST075I",  ,
      "IST314I"
do queued()
     parse pull MsgID MsgText
     if MsgID = "IST262I" then do
          parse var MsgText hdr"= "name", STATUS= "status
          if hdr = "APPLNAME" then do
            TSOName = name
            TSOStatus = status
           end
          if hdr = "LUNAME" then do
            LUName = name
            LUStatus = status
           end
       end
  end
```

3. Global Variables - Logical/Stem/Associative Arrays

Objectives:

1. Simplify the status setting and determination of a particular subsystem
2. Can be used to drive a graphic status panel (ie. subsystem name in green if up, yellow if brought down cleanly, red if crashed, etc.)

Example: Set status variables for group of CICS's. Retain the time each CICS was last brought up or down. There is nothing 'CICS-unique' about this example - any subsystem on any platform can be substituted (just the type of global variable handling would have to change).

a. Executed during System Initialization

```
/* REXX */
AllCICS = "PROD01 PROD02 ... PRODxx"
"GLOBALV PUTC ALLCICS"
CICSUp. = 0
do until AllCICS = ""
      parse var AllCICS CurrCICS AllCICS
      "GLOBALV PUTC CICSUP."CurrCICS
      call 'STRTCICS' CurrCICS
  end
```

b. Start a given CICS region (ie. STRTCICS PROD01)

```
/* REXX */
parse upper arg CurrCICS
  .  .  .
/* Current CICS brought up OK */
CICSUp.CurrCICS = 1
CICSDtTm.CurrCICS = 'DATE'("U") 'TIME'()
"GLOBALV PUTC CICSUP."CurrCICS "CICSDTTM."CurrCICS
```

c. Stop a given CICS region (ie. STOPCICS PROD01)

```
/* REXX */
parse upper arg CurrCICS
  .  .  .
/* Current CICS brought down OK */
CICSUp.CurrCICS = 0
CICSDtTm.CurrCICS = 'DATE'("U") 'TIME'()
CICSWhyDown.CurrCICS = "Stopped by" 'OP'()
"GLOBALV PUTC CICSUP."CurrCICS "CICSDTTM."CurrCICS ,
      "CICSWHYDOWN."CurrCICS
```

d. Restart CICS due to some error (ie. RSTCICS PROD01, probably called from NetView Message Automation Table after hit on abend message)

```
/* REXX */
parse upper arg CurrCICS AbendInfo
CICSUp.CurrCICS = 0
CICSDtTm.CurrCICS = 'DATE'("U") 'TIME'()
CICSWhyDown.CurrCICS = "Abended:" AbendInfo
"GLOBALV PUTC CICSUP."CurrCICS "CICSDTTM."CurrCICS ,
      "CICSWHYDOWN."CurrCICS
/* Restart Current CICS */
  . . .
```

e. Status of CICS regions

```
"GLOBALV GETC ALLCICS"
do until AllCICS = ""
      "GLOBALV GETC CICSUP."CurrCICS  ,
            "CICSDTTM."CurrCICS "CICSWHYDOWN."CurrCICS
      select
            when CICSUP.CurrCICS then
                  say "UP  " CurrCICS
            when ^CICSUp.CurrCICS  &
                     CICSWhyDown.CurrCICS <> "" then
                  say "DOWN" CurrCICS  CICSWhyDown.CurrCICS
            when ^CICSUp.CurrCICS  &
                     CICSWhyDown.CurrCICS = "" then
                  say "DOWN" CurrCICS "Never Started"
            otherwise say  "Unknown" CurrCICS
         end
   end
```

4. Log Processing

Objectives:

1. Perform filtering and summary information against log files (ie. MVS system log, VM operator console log, NetView log, etc.).

Example 1: Create a subset of a large log file. Scan an entire log and write only VTAM messages to another dataset.

```
/* REXX */
/* Scan a log and filter messages */
/* Delete/Erase the Output File */
/* if MVS/NetView, ALLOCATE here */
ReadLoop: do until ExecioRC <> 0
      "EXECIO *nnnnn DISKR <InputFile>"
      ExecioRC = rc
      PullLoop: do queued()
            /* Message ID starts in 10 */
            /* Save only VTAM (IST) Messages */
            parse pull . 10 MsgID 13 1 MsgRec
            if MsgID = "IST" then queue MsgRec
        end /* PullLoop */
      /* if any matches on IST then write */
      if queued() > 0 then
            "EXECIO" queued() "DISKW <OutputFile>"
  end /* ReadLoop */
/* Close files here */
```

Example 2: Display a summary of message occurances

```
/* REXX */
/* Scan a log and sum by message id */

/* if MVS/NetView, ALLOCATE here */

UniqueMsg = ""
GotMsg. = 0
SumMsg. = 0
TotMsgs = 0
ReadLoop: do until ExecioRC <> 0
      "EXECIO nnnnn DISKR ...."
      ExecioRC = rc
      TotMsgs = TotMsgs + queued()
      PullLoop: do queued()
            /* Message ID is in cols 10-19 */
            parse pull . 10 MsgID 20 .
            SumMsg.MsgID = SumMsg.MsgID + 1
            if ^GotMsg.MsgID then do
                  UniqueMsg = UniqueMsg || MsgID" "
                  GotMsg.MsgID = 1
               end
         end /* PullLoop */
   end  /* ReadLoop */

/* Close the log file here */

/* Display Msgid # % */
do until UniqueMsg = ""
      parse var UniqueMsg MsgID UniqueMsg
      Pct = 100 * (SumMsg.MsgID/TotMsgs)
      say 'LEFT'(MsgID,12) 'RIGHT'(SumMsg.MsgID,8) ,
        'FORMAT'(Pct,3,0) || "%"
   end
```

5. Screen Image Parsing

Objectives:

1. Parse screen images to isolate critical information

Example: The following screen image was trapped into one variable, SCREEN. Extract the CPU utilization for the displayed applications.

```
          Performance Stuff
before
before
App          Util
= = = = = = = =|= = = = = = = =| ...
ME           22
YOU          15
= = = = = = = =|= = = = = = = =| ...
after
after
```

```
/* REXX */
GotHdr = 0
do while Screen <> ""
      parse var Screen 1 Line 81 Screen
      parse var Line 1 Hdr 8 1 SubSys 10 UtilCPU 15 .
      select
            when ^GotHdr & Hdr = '= = = = = = = =' then
                  GotHdr = 1
            when GotHdr & Hdr = '= = = = = = = =' then
                  leave
            when GotHdr then say SubSys UtilCPU
            otherwise nop  /* 'Before' stuff */
        end
  end
```

6. Table Driven Automation

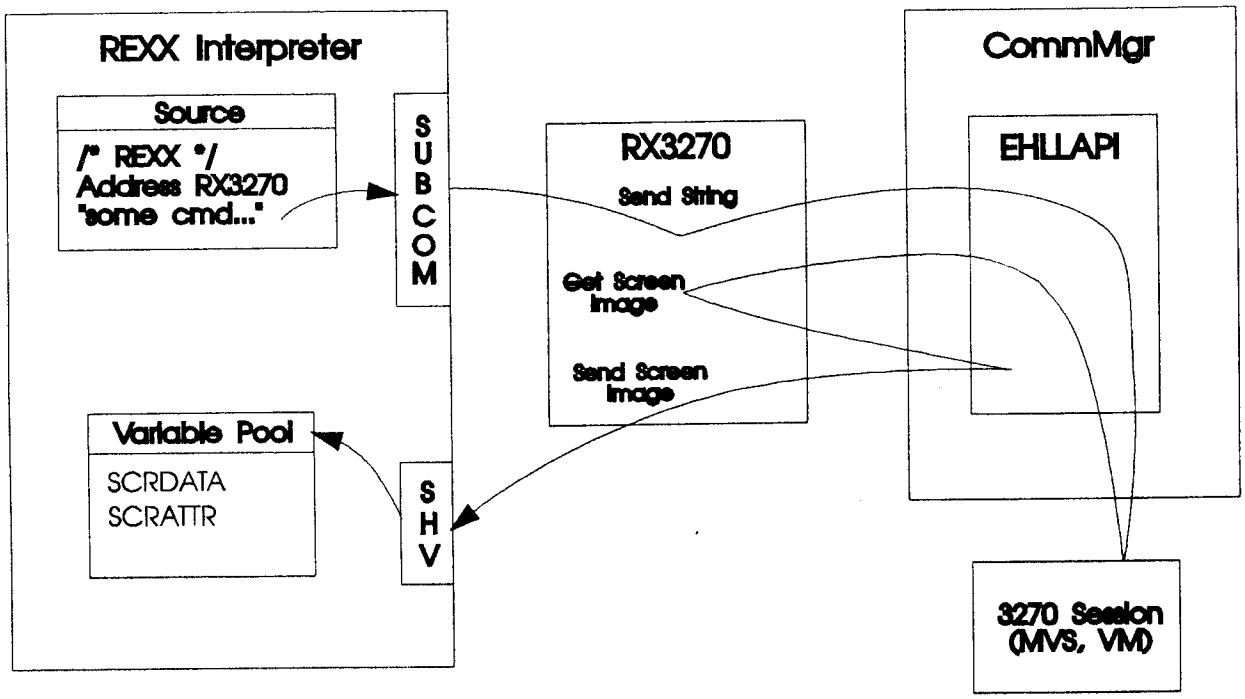
7. Testing and Simulation

8. Selective/Blanket Restart Enable/Disable

9. System/NCP/etc. Generation File
   Scanning/Parsing/Comparing

V. OS/2 CommMgr as an AO Tool

1. REXX is supplied with OS/2
2. CommMgr uses EHLLAPI to allow session management, namely:
   1. Issuing text strings to a 3270 session
   2. Retrieving 3270 screen images
3. REXX API's support Environments, Shared Variable Interface, Function Libs
4. REXX3270 tool:



VI. Indirect Benefits

1. Table driven status/recovery routines allow ownership of resources to be rapidly moved to alleviate performance/failure considerations

2. Disaster Recovery
    1. A 'disaster' table can exist which contains only critical devices mapped to the ownership of critical systems
    2. A 'snapshot' program can display/query critical system components/values on a periodic basis and save this info into a table. After and disaster and recovery, a display/query job can be run to verify critical component availability and differences.

3. Job Automation. Experience/confidence gained during AO implementation can be extended to automating nightly job cycles, replacing JCL with REXX to allow for more intelligent and automatic job monitoring/restart/correction.

VII. The Future...

1. Dynamic Configuration Management. Access external matrix switches to reconfigure devices from one system to another 'on the fly', both for performance and failure recovery purposes.

2. Enterprise Automation

3. DMS?

4. NetWare?

5. ???

The programs/ideas in this document are in the public domain. Use them in any manner. Most were written to run under NetView and/or MVS, but should, with minor changes, run anywhere. Be careful - I either clipped them out of larger programs or wrote them from memory based on projects I worked on in the past - typos are probable. More importantly, to keep things concise, I removed all the error handling code. If you have any questions, feel free to call/fax me at (201) 492-2777. I'm always willing to help and curious to hear how different sites implement automated operations.

Thanks,
Pete Zybrick