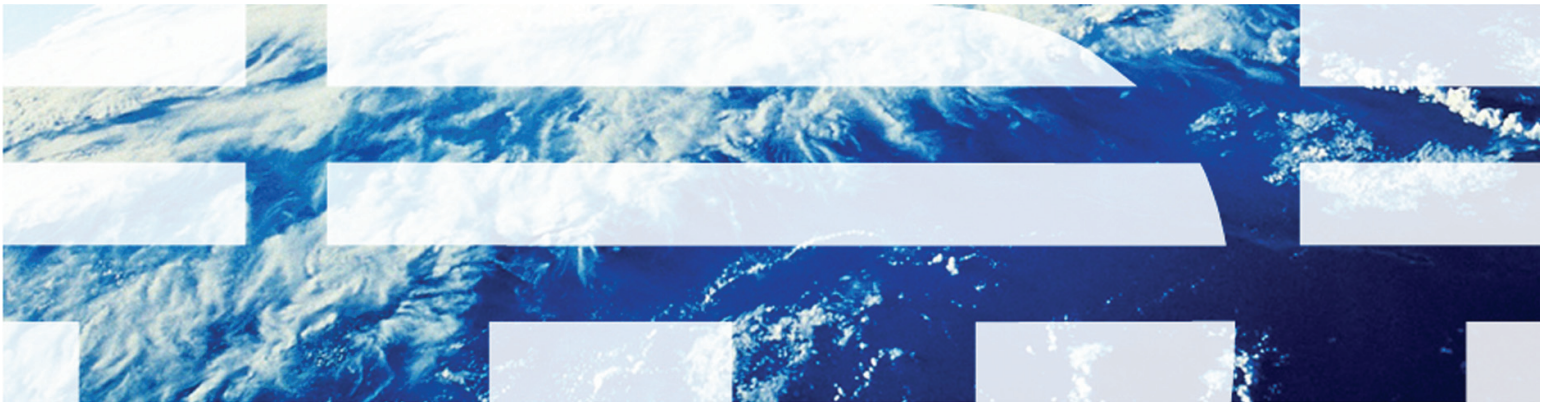

IBM Tivoli NetView for z/OS 6.1: REXX Programming

Student Exercises

Course: TZ223 ERC: 1.0

September 2011

Printed in Ireland

Table of contents

Student exercises for Unit 1

Student exercises for Unit 1

Lab exercise overview

In these exercises, you create several REXX EXECs in TSO and test them in NetView®. You should have two PCOMM sessions for this lab exercise. One session is for NetView domain AOFDA and the other is for TSO.

Data sets used in this lab

NV390.WORKSHOP.CNMCLST_____

Exercise 1-1: Simple REXX EXECs

Lab exercise objectives

This lab provides an introduction to programming REXX EXECs for NetView. At the end of this lab, you should be able to perform the following tasks:

- Parse EXEC input.
- Utilize several NetView built-in functions.
- Display messages to operators.

Lab exercise instructions

___ 1. Access your system and log on to TSO. Follow the instructions that the instructor provides. Log on to NetView AOFDA domain, also following the instructions.

Optionally, issue HILITE REXX on the TSO edit command line. This command modifies the display to use different colors as you create the EXEC. If you do not like the colors, issue HILITE OFF.

___ 2. Create a new REXX EXEC, called MYVARS, to perform the following steps:

___ a. Parse the two following inputs to the EXEC:

- Use the REXX PARSE statement to parse four parameters with the fourth being the rest of the input string.
- Use the NetView MSGVAR(*n*) EXEC to parse four parameters.

___ b. Use several SAY instructions to display the input parameters for each method of parsing.

___ c. When you finish, save your work.

___ 3. Issue the MYVARS EXEC from AOFDA, using more than 4 parameters, as follows:

```
MYVARS parm1 parm2 parm3 parm4 parm5
```

Use lower-case text for the parameters. Your output should look similar to the following text:

```
MYVARS: Input parsed with PARSE ARG
        Parameter 1 is  PARM1
        Parameter 2 is  PARM2
        Parameter 3 is  PARM3
        Parameter 4 is  PARM4 PARM5
MYVARS: Input parsed with NetView MSGVAR(n)
        Parameter 1 is  PARM1
        Parameter 2 is  PARM2
        Parameter 3 is  PARM3
        Parameter 4 is  PARM4
```

Why does the PARSE ARG display two parameters as the last parameter?

_____

Why does the MSGVAR(*n*) ignore the fifth parameter? _____

_____

Why does the EXEC display the parameters in upper case? _____

_____

___ 4. To keep the parameters in mixed-case text, prefix the EXEC with NETVASIS. For example, enter text as follows:

```
NETVASIS MYVARS parm1 parm2 parm3 parm4 parm5
```

The input parameters should appear exactly as you typed them.

___ 5. Issue the MYVARS EXEC with commas (,) as delimiters between each parameter as follows:

```
NETVASIS MYVARS parm1,parm2,parm3,parm4,parm5
```

What changes in the output? Why? _____

_____

___ 6. Update the MYVARS EXEC to query several NetView built-in functions and write their values to the operator:

___ a. Query the following functions:

- CurSys()
- Sysplex()
- MVSlevel()
- NetView('T')
- Tower('*')
- Netid()
- Domain()
- Opid()
- Task()
- Curconid()
- Lu()

___ b. Add the use of several REXX functions as follows:

- Date(): Displays the date in the European format.
- Date('W'): Displays the current day.
- Time(): Displays the date in the 24-hour clock format.
- Words(): Displays the **number** of towers that are enabled.

___ c. Modify the input parameter parsing:

___ d. Test for the existence of input parameters.

___ e. If there are no input parameters, do not bother to use the PARSE ARG and MSGVAR statements.

Tip: Use an IF-THEN construct with a DO-END grouping.

___ 7. Save your work when you finish. Issue the MYVARS EXEC without any input parameters. The response should be similar to the following text:

```
Current sysplex name: PLEX12
Current system name : MVSA
Current z/OS level: SP7.1.2
NetView level: Tivoli NetView for z/OS V6R1
Towers enabled: NPDA NLDM TCPIPCOLLECT                    TEMA IPMGT
NVSOA DISCOVERY
There are  5  towers enabled.
The third tower is: TCPIPCOLLECT
NetView domain ID: AOFDA
NetView operator ID: TSCCW01
NetView task type: OST
Current z/OS console name:
Current VTAM LU name: ESIP1017
Current VTAM Network ID: USIBMES
Current date is: 06/15/11
Current time is: 06:01:06
Today's day is: Wednesday
```

If you receive any error messages, correct your coding in MYVARS and retry.

___ 8. Edit MYVARS again to prompt for input (using PARSE PULL) from the operator before continuing. For example, ask the operator if the day name was parsed correctly.

___ a. Issue a message to the operator identifying the desired input as follows, for example.

```
Is the day correct?  Enter "GO Y" or "GO N"
```

___ b. Use PARSE PULL to pause the EXEC until the operator replies.

___ c. After a reply is provided, display the response back to the operator. Optionally, test if the response is a Y. If not, display the message again and wait for operator response again.

___ d. Issue another message indicating that the lab exercise is done.

Answers to lab exercise 1 questions

Step 3

PARSE ARG displays the last two input parameters as one because of the way you were asked to code the parse:

```
parse arg p1 p2 p3 rest
```

This parses the input string into four parameters as follows:

```
p1: First parameter
p2: Second parameter
p3: Third parameter
Rest: The rest of the parameter string
```

Another example of PARSE ARG is to ignore the remaining text by using a period (.) at the end of the statement. For example:

```
parse arg p1 p2 p3 p4 .
```

MSGVAR(n) does not display a fifth input parameter because it was not coded. An alternative implementation is using the PARMCNT() function to determine the number of parameters with a loop similar to the following text:

```
Do i = 1 to PARMCNT()
Say 'Parameter number' i 'is:' MSGVAR(i)
End
```

By default, the input parameters convert to upper case. You can control the result by using NETVASIS or by defining the EXEC as a command in member CNMCMDU and coding the FOLDUP=Y definition:

```
CMDDEF.MYVARS.MOD=DSICCP
CMDDEF.MYVARS.FOLDUP=Y
```

Step 5

By default, PARSE ARG is blank-delimited. When commas are used in the input, the PARSE ARG treats them as part of the text string.

NetView MSGVAR(*n*) parse is based on blanks, commas, single quotations, and the equal sign. MSGVAR(*n*) uses slightly more CPU cycles than EXEC because the EXEC leaves the REXX interpreter and launches into a NetView environment.

The following text is an example of coding MYVAR for this lab exercise:

```
/* REXX */
/* This is a simple REXX EXEC for L4 EX1 */
/* */

parse arg argstring .
If argstring <> '' then
 Do

  /* Use REXX PARSE ARG: parse input and write to operator */
  parse arg p1 p2 p3 rest /* input parsed into 4 parms  */

  Say 'MYVARS: Input parsed with PARSE ARG'
  Say '       Parameter 1 is ' p1
  Say '       Parameter 2 is ' p2
  Say '       Parameter 3 is ' p3
  Say '       Parameter 4 is ' rest


  /* Use NetView MSGVAR(n): parse input, write to operator */
  Say 'MYVARS: Input parsed with NetView MSGVAR(n)'
  Say '       Parameter 1 is ' MSGVAR(1)
  Say '       Parameter 2 is ' MSGVAR(2)
  Say '       Parameter 3 is ' MSGVAR(3)
  Say '       Parameter 4 is ' MSGVAR(4)

 End

/* Use NetView built-in functions, write to operator */

Say 'Current sysplex name:' Sysplex()
Say 'Current system name :' CurSys()
Say 'Current z/OS level:' MVSlevel()
Say 'NetView level:' NetView('T')
Say 'Towers enabled:' Tower('*')
Say 'There are ' Words(NetView('T')) ' towers enabled.'
Say 'The third tower is:' Word(Tower('*'),3)
Say 'NetView domain ID:' Domain()
Say 'NetView operator ID:' Opid()
Say 'NetView task type:' Task()
Say 'Current z/OS console name:' Curconid()
Say 'Current VTAM LU name:' Lu()
Say 'Current VTAM Network ID:' Netid()

/* Use REXX provided functions and write to operator */
Say 'Current date is:' Date('E')
Say 'Current time is:' Time()

Day_Name = Date('W')

Say "Today's day is:" Day_Name

/* Use PARSE PULL to pause for operator input */
```



```
Resp = ''
Do While Resp = '' /* Loop: Resp Y or N     */
  Say 'Is the day correct?  Enter "GO Y" or "GO N"'
  parse pull resp
   If Resp = 'Y' Then
     Say 'You responded with a:' resp
  Else
    Do
     Say 'Incorrect response.'
     Resp = ''
    End
End /* Loop: Resp Y or N     */


Say 'End of MYVARS lab exercise'


Exit
```

Exercise 1-2: Global variables

Lab exercise objectives

This lab provides an introduction to REXX and global variables for NetView. At the end of this lab, you should be able to perform the following tasks:

- Use the QRYGLOBL command.
- Retrieve, update, and store common global variables.

Lab exercise instructions

___ 1. Access your system and log on to TSO. Follow the instructions that the instructor provides. Log on to NetView AOFDA domain, also following the instructions.

___ 2. From AOFDA, issue **HELP QRYGLOBL** to display the online help for the QRYGLOBL command. Review the information about displaying global variables.

What command do you issue to display the common global variables that begin with CNMSTYLE.AUTO? _____

___ 3. Issue the QRYGLOBL command by using the NetView WINDOW function. Your result should look similar to the following text:

```
BNH031I NETVIEW GLOBAL VARIABLE INFORMATION
BNH103I COMMAND ISSUED AT: 06/15/11 10:13:23
BNH061I
BNH032I COMMON GLOBAL VARIABLES
BNH036I GLOBAL VARIABLE NAME:                GLOBAL VARIABLE VALUE:
BNH061I --------------------                 ----------------------
BNH039I CNMSTYLE.AUTO.MVSCMDREVISION         MVSCMDS
BNH039I CNMSTYLE.AUTO.SAVEDBMAINT            DBAUTO1
BNH039I CNMSTYLE.AUTO.EMAAUTOM               AUTO1
BNH039I CNMSTYLE.AUTO.AUTOIP                 AUTOAON
BNH039I CNMSTYLE.AUTO.PKTS.TCPIP             AUTOPKTS
BNH039I CNMSTYLE.AUTO.HMONDBMAINT            DBAUTO2
BNH039I CNMSTYLE.AUTO.TCPSM.TCPIP            AUTOTCPS
BNH039I CNMSTYLE.AUTO.NAPOLTSK2              AUTODC2
BNH039I CNMSTYLE.AUTO.NAPOLTSK3              AUTODC3
BNH039I CNMSTYLE.AUTO.NAPOLTSK1              AUTODC1
BNH039I CNMSTYLE.AUTO.NAPOLTSK4              AUTODC4
BNH039I CNMSTYLE.AUTO.SMONDBMAINT            DBAUTO1
BNH039I CNMSTYLE.AUTO.OPKT.TCPIP             AUTOOPKT
BNH039I CNMSTYLE.AUTO.COLTSK5                AUTOCT5
BNH039I CNMSTYLE.AUTO.XCFDISC                AUTOXDSC
BNH039I CNMSTYLE.AUTO.TCPCDBMAINT            DBAUTO2
BNH039I CNMSTYLE.AUTO.MEMSTORE               AUTO2
BNH039I CNMSTYLE.AUTO.ENTDATA                AUTOEDAT
BNH039I CNMSTYLE.AUTO.NVSOAPTSK              AUTONVSP
BNH039I CNMSTYLE.AUTO.IDLEOFF                AUTO1
BNH039I CNMSTYLE.AUTO.XCF                    AUTOXCF
BNH039I CNMSTYLE.AUTO.COLTSK7                AUTOCT7
BNH039I CNMSTYLE.AUTO.APSERV                 AUTOTMSI
BNH039I CNMSTYLE.AUTO.MASTER                 AUTO1
BNH039I CNMSTYLE.AUTO.NALCLOP                AUTONALC
BNH039I CNMSTYLE.AUTO.POLICY                 AUTOAON
BNH039I CNMSTYLE.AUTO.TCPCONN.TCPIP          AUTOTCPC
BNH039I CNMSTYLE.AUTO.NETCONV                AUTO2
BNH039I CNMSTYLE.AUTO.DLAAUTO                AUTO2
BNH039I CNMSTYLE.AUTO.COLTSK6                AUTOCT6
BNH039I CNMSTYLE.AUTO.PRIMARY                AUTO1
BNH039I CNMSTYLE.AUTO.FKXPKTS                AUTOPSAV
BNH035I NUMBER OF VARIABLES FOUND: 32
BNH061I
BNH033I TASK GLOBAL VARIABLES FOR NETOP1
BNH036I GLOBAL VARIABLE NAME:                GLOBAL VARIABLE VALUE:
BNH061I --------------------                 ----------------------
BNH035I NUMBER OF VARIABLES FOUND: 0
BNH061I

NETVIEW GLOBAL VARIABLE INFORMATION COMPLETE
```

These are all of the automation operators (autotasks) that are defined to NetView in CNMSTYLE. For example, XCF processing is scheduled under the CNMSTYLE.AUTO.XCF task (AUTOXCF).

Tip: During the lab exercises, you can test to see if a GLOBALV PUT*x* command works by using the QRYGLOBL command to retrieve the global variable value.

___ 4. Issue **HELP GLOBALV** to display the online help for the GLOBALV command. The *Help GLOBALV* panel (EUYSLIST) should open. Select options as follows:

- A: REXX GLOBALV
- **C:** GLOBALV DEF
- **D:** GLOBALV GET
- **F:** GLOBALV PUT

___ 5. Create a new REXX EXEC named MYGLOB as follows:

___ a. Use PARSE ARG or MSGVAR(*n*) to parse the input string into the three following parameters:

- Global function type: DEFT, DEFC, GETT, GETC, PUTT, or PUTC
- Global variable name
- Global variable value

___ b. Issue the appropriate GLOBALV command to define, retrieve, or put the value for the global variable (parameter two).

Tip: Use the REXX **INTERPRET** instruction before your GLOBALV PUT*x* command, such as the following text example:

```
Interpret VarName '=' VarValue
```

In this case, the INTERPRET instruction sets the second parameter (VarName) to the value of the third parameter (VarValue).

___ c. Test the return code from the GLOBALV command and display a message similar to the following text:

```
GLOBALV Return code: Rc
```

___ d. For GET*x* requests, also display the value of the global variable after checking the return code.

Tip: Use the REXX **Value()** function to display the value of the global variable, such as the example text:

```
If Substr(ReqType,1,3) = 'GET' then
 Say 'Value of variable ' VarName ' = ' Value(VarName)
```

This displays the variable (VarName) and its value using Value(VarName).

___ e. When finished, save your work.

___ 6. From AOFDA, issue **MYGLOB GET MYVAR**. What response did you get?

_____

___ 7. Correct the mistake and reissue the MYGLOB EXEC. What response did you get? Why?

_____

___ 8. Issue **MYGLOB GETT MYVAR.***. What does the return code mean?

_____

___ 9. Issue **MYGLOB PUTT MYVAR_COUNT 99**, followed by **MYGLOB GETT MYVAR_COUNT**. Your response should be similar to the following text:

```
Value of variable  MYVAR_COUNT  =  99
```

___ 10. In TSO, create another REXX EXEC, naming it OPERGLOB. OPERGLOB is to issue GLOBALV PUT*x* and GET*x* commands to set several global variables for the following items:

- Your name
- Timestamp: Date and time that OPERGLOB was last issued.
- Counter: Number of times that OPERGLOB has been called.
- Update of the date and time as well as the counter every time the EXEC is called.

These variables need to be saved after you log off. What type of variable do you need to use? _____

When OPERGLOB runs, write messages to the operator similar to the following text:

```
My Name is ... Lab User 1
OPERGLOB was last used ... 06/15/11 06:33:59.469681
OPERGLOB was called 3 times
```

___ 11. Run the OPERGLOB EXEC several times and note the timestamp as well as the counter. Log off from AOFDA.

___ 12. Log on to AOFDA as the same operator ID and issue the OPERGLOB EXEC. The timestamp and counter within the output should update.

```
My Name is ... Lab User 1
OPERGLOB was last used ... 06/15/11 06:34:12.623707
OPERGLOB was called 4 times
```

If update does not occur, correct the problem in OPERGLOB and repeat these steps again.

Explain how you would reset these variable values.

_____

_____

_____

Answers to lab exercise 2 questions

Step 2

Query the common global variables set by CNMSTYLE for autotasks:

```
QRYGLOBL COMMON VARS=CNMSTYLE.AUTO*
```

Step 6

GLOBALV yields a return code 52 because the second parameter was misspelled: GET instead of GETT.

Step 7

GLOBALV yields a return code 0 because the command syntax was correct. The variable value would be null because it had not been set with a GLOBALV PUTT command.

Step 8

GLOBALV yields a return code of 14004 when the variable name is not valid.

Step 10

Because this information is needed after you log off, you use common global variables.

Step 12

Reset the value of each common global variable by setting the value of each variable to null and issuing a GLOBALV PUTC for all three variables.:

Exercise 1-3: Global variables tracing

Lab exercise objectives

This lab provides an introduction to REXX and global variables tracing and automation. At the end of this lab, you should be able to set up tracing and automation for global variables.

Lab exercise instructions

___ 1. Access your system and log on to TSO. Follow the instructions that the instructor provides. Log on to NetView AOFDA domain, also following the instructions.

___ 2. Create a REXX EXEC called MYGLOB2 to perform the following tasks:

___ a. Write common global variable KVAR1 with the initial value of `'AUTOMATION START'`.

___ b. Write common global variable KVAR2 with initial value of `'TRACE START'`.

___ c. Automate common global variable KVAR1.

___ d. Trace common global variable KVAR2.

___ e. Change the value of common global variable KVAR1 to `'AUTOMATION CHANGED'` and write it.

___ f. Change the value of common global variable KVAR2 to `'TRACE CHANGED'` and write it.

___ 3. Run MYGLOB2 and view the NetView log. Entries that resulted from tracing and automating the common global variables should be similar to the following text:

```
DWO994I Common global variable KVAR1 set to value ---
>AUTOMATION CHANGED
DWO990I Common global variable 'KVAR2' set by 'GLOBALV' via
'MYGLOB2--->GLOBALV' to value --->TRACE CHANGED<---
```

Note: By default, the DWO994I message is not displayed or logged. We have defined an automation table entry to log the message:

```
IF MSGID = 'DWO994I' THEN NETLOG(Y);
```

The difference between Trace and Automate is information detail. Automate informs that the global variable is changed and old and new values. Trace provides more information about the change to the global variable. Both DWO990I and DWO994I messages can be automated.

Answers to lab exercise 3 questions

```
/* REXX */
/* MYGLOB2 */
say '*************************************************************'
say '* Now we set KVAR1 and KVAR2                                *'
say '*************************************************************'
KVAR1='AUTOMATION START'
'GLOBALV PUTC KVAR1'
KVAR2='TRACE START'
'GLOBALV PUTC KVAR2'
say '*************************************************************'
say '* Now we automate and trace common variables KVAR1 KVAR2  *'
say '*************************************************************'
'GLOBALV AUTOC ON * KVAR1'
'GLOBALV TRACEC ON * KVAR2'
say '*************************************************************'
say '* Now we change KVAR1 and KVAR2                             *'
say '*************************************************************'
KVAR1='AUTOMATION CHANGED'
'GLOBALV PUTC KVAR1'
KVAR2='TRACE CHANGED'
'GLOBALV PUTC KVAR2'
```

Exercise 1-4: TRAP and WAIT

This lab provides an introduction to programming REXX EXECs for NetView that issue commands and trap their responses. For example, a REXX EXEC can be driven from the automation table to take actions based on notification of a resource failure. The REXX EXEC might need to issue commands to collect more data about the resource or the failure event before taking corrective actions.

Lab exercise objectives

At the end of this lab, you should be able to:

- Identify messages to be trapped.
- Code an EXEC to wait for one or more trapped messages. (You will also have to continue waiting for messages within your REXX EXEC.)
- Read a trapped message.
- Parse the response to a command that is issued to generate one or more trapped messages.
- Code the appropriate automation table statements to drive a user REXX EXEC.

Lab exercise instructions

Create a REXX EXEC to automate the archival and switching of the NetView DSILOG VSAM data set. Also code automation table statements to call your EXEC.

When DSILOG switches from the primary VSAM data set to the secondary VSAM data set, the following messages occur:

```
DSI556I DSILOG : VSAM DATASET 'OPEN' COMPLETED, DDNAME = 'DSILOGS'
RETURN CODE = X'00', ACB ERROR FIELD = X'00'
DSI547I DSILOG : SECONDARY VSAM DATA SET IS NOW ACTIVE
DWO520I DSILOG : VSAM DATASET 'CLOSE' COMPLETED, DDNAME =
'DSILOGP' RETURN CODE = X'00', ACB ERROR FIELD = X'00'
```

You see similar messages when the switch occurs from secondary to primary with a DSI546I issued instead of DSI547I:

```
DSI546I DSILOG : PRIMARY VSAM DATA SET IS NOW ACTIVE
```

___ 1. Create a new REXX EXEC, LOGAUTO, to performs the following steps:

___ a. Parse a single input parameter to the EXEC: PRIMARY or SECONDARY.

___ b. Issue the LOGMAINT (user-written command that the instructor provides) EXEC to perform database maintenance on the inactive log file.

___ c. Test the return code when running LOGMAINT.

- If the return is zero, continue running LOGMAINT.
- If the return code is non-zero, issue an error message.

___ d. When driven for a switch to secondary (DSI547I), define a list of messages to TRAP for a NetView SWITCH command:

- Trap and display the DSI547I message indicating a switch from primary to secondary.
- Trap and suppress the DSI556I and DWO520I messages.

___ e. When driven for a switch from secondary to primary, do not issue any further commands. This archives the contents of the secondary log data set and waits for the primary to fill before it automatically switches.

___ f. WAIT for the messages to be trapped. Use the WAITTIME common global for the amount of time to wait, in seconds.

___ g. Code for each of the possible event types as follows:

- **M:** Process the messages from the trap message queue.
- **T:** Issue a message indicating a timeout occurred.
- **E:** Issue a message indicating an error occurred.
- **G:** Issue a message that the operator entered the **GO** command.

Tip: Refer to the lecture slide, “Example: Trapping a single message”, for an example of the REXX coding that this lab exercise requires.

You can use REXX PARSE or NetView functions, such as MSGID(), MSGSTR(), and MSGVAR(*n*) to parse the messages that are received.

___ h. Make sure you code the **FLUSHQ** and **TRAP NO MESSAGES** commands.

___ 2. Create a new automation table, LOGTBL, to process the DSI546I and DSI547I messages:

___ a. Process only the messages for task name of DSILOG.

___ b. Pass the log data set type (PRIMARY or SECONDARY) to the LOGAUTO EXEC.

See the lab exercise answers for an example automation table.

___ 3. Issue a **LIST DSILOG** command. If the primary data set is not active, issue a **SWITCH DSILOG,P** command to switch logging to the primary data set before you test your EXEC and automation table.

___ 4. Issue **AUTOTBL MEMBER=LOGTBL,TEST** to test the syntax of your automation table. Correct errors before attempting to activate the automation table.

___ 5. Use **AUTOMAN** to insert the new automation table after DSITBL01.

___ 6. Issue a **SWITCH DSILOG,S** command. The automation statements you coded in LOGTBL should perform the following steps:

___ a. Drive LOGAUTO to call LOGMAINT for the primary data set.

___ b. Automatically issue a SWITCH DSILOG,P command.

LOGAUTO should be driven a second time to perform database maintenance on the secondary data set.

Note: Each time LOGAUTO runs, you should notice the *wait indicator* (**W**) on your NetView screen. The *wait indicator* flickers as each new trapped message processes.

You should see the following messages:

```
DSI556I DSILOG : VSAM DATASET 'OPEN' COMPLETED, DDNAME = 'DSILOGS'
RETURN CODE = X'00', ACB ERROR FIELD = X'00'

DSI547I DSILOG : SECONDARY VSAM DATA SET IS NOW ACTIVE .

DWO520I DSILOG : VSAM DATASET 'CLOSE' COMPLETED, DDNAME =
'DSILOGP'RETURN CODE = X'00', ACB ERROR FIELD = X'00'
```

The DSI556I, DSI547I, and DWO520I messages result from the SWITCH command you just entered. Subsequent DSI556I and DWO520I should be suppressed by the LOGAUTO EXEC.

```
++++ LOGMAINT: Performing database maintenance on PRIMARY
DSI546I DSILOG : PRIMARY VSAM DATA SET IS NOW ACTIVE
++++ LOGMAINT: Performing database maintenance on SECONDARY
```

The LOGMAINT EXEC issues an informational message.

The DSI546I and DSI547I messages should be held (alternately) on your screen until the log data set switches, based on statements supplied in DSITBL01. If your results differ, modify your LOGAUTO EXEC and issue the **SWITCH DSILOG,S** command again to test your changes.

___ 7. When you finish, use **AUTOMAN** to unload the LOGTBL automation table.

Answers to lab exercise 4 questions

Step 1

Your automation table should look similar to the following text:

```
if msgid='DSI546I' | msgid='DSI547I' & token(2) = 'DSILOG'
  & token(4) = PriOrSec then
      EXEC(CMD('LOGAUTO ' PriOrSec) ROUTE(ONE *)) HOLD(Y);
```

Exercise 1-5: EXECIO

Lab exercise objectives

This lab is optional. This lab provides an introduction to programming REXX EXECs for NetView that read members of a data set. At the end of this lab, you should be able to perform the following tasks:

- Use TRAP, WAIT and MSGREAD commands, along with MSGID(), MSGSTR(), and MSGVAR() functions.
- Allocate a data set and member to NetView.
- Use EXECIO to read the member from the data set.
- Deallocate the data set and member from NetView.

Note: Your instructor might provide a different data set and member to use for this exercise:

NV390.V6R1M0.USER.DSIPARM(RMTSTGEN)

_____

Lab exercise instructions

___ 1. Create a new EXEC, READMEM, to read member RMTSTGEN to perform the following steps:

___ a. Parse the data set name and member as a single input parameter.

___ b. Test the input and issue an error message if it is null.

___ c. Optionally, test the input to make sure it contains a data set name and member name. Use NV390.V6R1M0.USER.DSIPARM(RMTSTGEN).

___ d. Use the NetView ALLOCATE command to allocate the data set as shared (SHR) and free it when complete (FREE).

Tip: Use the online help for the ALLOCATE command.

If the ALLOCATE command is successful, a CNM272I message is displayed as follows:

```
CNM272I  ddname IS NOW [ALLOCATED | DEALLOCATED]
```

You need to trap the CNM272I and parse the *ddname* so that EXECIO can read from the *ddname*.

___ e. Use EXECIO to read the contents of the data set member (identified by *ddname*) into a REXX stem variable.

___ f. Loop through the REXX stem variable, issuing a SAY command for each line.

___ g. When the loop ends (finished displaying the contents of the stem variable), issue a message identifying that the read has completed. Use EXECIO to close the file.

___ 2. When finished, save your work.

___ 3. In NetView AOFDA domain, issue **READMEM NV390.V6R1M0.USER.DSIPARM(RMTSTGEN)**. When successful, the output should be displayed similar to the following text:

```
File NV390.V6R1M0.USER.DSIPARM(RMTSTGEN)/SYS00275 has 8 lines:
 ****** ONLY USED FOR EXECIO LAB ******
 RMTINIT.PORT    = 4022
 RMTINIT.IP  = Yes
 RMTINIT.TCPNAME = &CNMTCPN
 RMTINIT.SOCKETS = 50          //
 RMTINIT.KEEPALIVE = 10        //
 RMTSYN.&CNMNETID..AOFDB = 10.31.&IPBSUB..&IPBLLQ.
 RMTSYN.&CNMNETID..AOFDA = 10.31.&IPASUB..&IPALLQ.
 READMEM EXEC is now finished
```


Printed in Ireland

CT6E8EN