



IBM Tivoli NetView for z/OS 6.1: REXX Programming

Student's Training Guide

Course: TZ223 ERC: 1.0

August 2011

© Copyright IBM Corp. 2011. All Rights Reserved.

US Government Users Restricted Rights: Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

References in this publication to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth, savings or other results.

Printed in Ireland

Table of contents

Unit 1: Introduction to REXX programming with NetView for z/OS

Introduction	1-2
Objectives	1-2
Lesson 1: REXX introduction	1-3
REXX overview	1-4
NetView REXX EXECs	1-5
Storage of NetView EXECs	1-6
Common uses of REXX EXECs	1-7
Calling a REXX EXEC	1-8
REXX basics (1 of 2)	1-9
REXX basics (2 of 2)	1-10
Parsing input parameters	1-11
Parsing input parameters with REXX	1-12
Parsing input parameters with NetView	1-13
Sending messages to operators	1-14
Using the SAY instruction	1-15
REXX example	1-16
NetView REXX example	1-17
Pausing for operator input	1-18
Lesson 2: REXX and NetView	1-19
REXX addressing environment	1-20
Defining EXECs to NetView	1-21
EXECs on an automated operator	1-22
EXECs under the PPT	1-23
EXECs on another NetView	1-25
%INCLUDE with interpreted REXX	1-26
Nested EXECs	1-27
External call example	1-28
NetView add-ons	1-29
REXX functions that NetView provides	1-30
LISTVAR example	1-31
Commonly used REXX built-in functions (1 of 2)	1-32
Commonly used REXX built-in functions (2 of 2)	1-33
Student exercise	1-34
Lesson 3: Variables	1-35
Variables in REXX	1-36
Overview of NetView global variables	1-37
GLOBALV command parameters (1 of 2)	1-38
GLOBALV PUTT PUTC:	1-38
GLOBALV GETT GETC:	1-39
GLOBALV DEFT DEFC:	1-39
GLOBALV SAVEx RESTOREx PURGEEx:	1-39
GLOBALV command parameters (2 of 2)	1-40
GLOBALV AUTOx:	1-40
GLOBALV TRACEx	1-41
Task global variables	1-42
Common global variables	1-43
Global variable notes (1 of 2)	1-44
Global variable notes (2 of 2)	1-45

Table of contents

GLOBALV with stem variables1-46
Displaying global variables1-48
QRYGLOBL example1-49
Setting common global variables1-50
CGED command1-51
Student exercise1-52
Lesson 4: Processing messages in REXX1-53
Overview of REXX to trap messages (1 of 2)1-54
Overview of REXX to trap messages (2 of 2)1-55
Functions that MSGREAD sets1-56
Example: Trapping a single message1-57
Multiline messages1-58
Messages from automation1-59
Subset of functions that automation sets1-60
Student exercise1-61
Lesson 5: Analyzing problems1-62
RXTRACE entry/exit tracing1-63
RXTRACE program tracing1-64
Example: Trace i program1-65
Processing return codes1-66
Processing standard errors1-67
SIGNAL instructions1-68
Example: NOVALUE1-69
Lesson 6: Additional topics1-70
Sending messages to the z/OS system console1-71
REXX environments1-72
Controlling the REXX environment1-73
Loading EXECs in NetView storage1-74
MemStore function1-75
Managing memStore members1-76
Performance1-77
Security1-78
Summary1-79

Unit 1: Introduction to REXX programming with NetView for z/OS

Tivoli software

IBM

Introduction to REXX programming with NetView for z/OS



© 2011 IBM Corp.

Introduction

This unit provides a background in programming REXX EXECs for your NetView® for z/OS® environment. REXX EXECs are key to improving operator productivity and providing automation.

Objectives

Tivoli software

IBM

Objectives

After completing this unit, you should be able to:

- Explain the basics of REXX EXECs in NetView
- Write NetView REXX EXECS to do the following tasks:
 - Issue commands
 - Trap and parse messages
 - Set and retrieve global variables
 - Trace and automate global variables
 - Perform automation

1-2

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Lesson 1: REXX introduction

Tivoli software

IBM

Lesson 1: REXX introduction

- Overview
- REXX and NetView
- REXX basics
- Simple examples

1-3

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This lesson provides a brief introduction to REXX and NetView REXX.

REXX overview

Tivoli software 

REXX overview

- REXX = Restructured Extended Executor language
- Set of commands and instructions that form a program, called an EXEC
For example, IF-THEN
- Interpreted when executed, statement-by-statement
Compile for improved performance
- Stored as a member in a data set
- Structured programming language
- Able to run on multiple platforms

1-4

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXX is a structured programming language that is **interpreted** when the program runs. The REXX interpreter operates directly on the program as it runs, line by line and word by word.

Basic REXX EXECs are capable of running on many platforms. One of the operating systems that supports REXX is z/OS. The operating system provides additional commands, environments, or functions in addition to the base that REXX provides. Applications can provide additional commands, environments, or functions. For example, NetView extends the available z/OS REXX functions with some of its own.

NetView REXX EXECs

Tivoli software

IBM

NetView REXX EXECs

- Members of DSICLD concatenation
- NetView supplies many EXECs
- Additional NetView functions
 - Available in NetView environment only
 - For example, common global variables
- For improved performance
 - Compile REXX EXECs
 - Load in NetView storage
 - REXX with PIPEs

1-5

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

NetView REXX EXECs can be created before NetView starts, or while NetView runs. REXX EXECs are saved as members of a partitioned data set (PDS) and are ready for NetView operators to use immediately. Many NetView commands are actually REXX EXECs that call command processors.



Tip: To reduce the time required to read an EXEC from a data set, load the EXEC in NetView storage.

Optionally, REXX EXECs can be compiled to significantly improve performance. The IBM REXX/370 compiler product must be installed on the system where the EXECs are to be compiled. The run-time library must be installed on the system that runs the compiled EXECs.

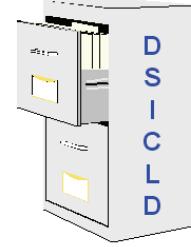
To determine if the REXX compiler is installed, issue ADDRESS LINKMVS 'EAGRTPRQ' within a REXX EXEC. If the return code is -3, it is installed.

Storage of NetView EXECs

Tivoli software **IBM**

Storage of NetView EXECs

- EXECs are data set members
 - Name is eight characters or fewer
 - Data sets are defined in NetView JCL DSICLD concatenation
- Optionally, can be operator data set With OVERRIDE DSICLD= command
- DSICLD concatenation rules apply For example, do **not** define secondary extents



1-6 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Each EXEC is a member of a NetView partitioned data set (PDS), which is allocated to the NetView DSICLD concatenation.

The PDS member name is the EXEC name, the name used by your NetView operators. Optionally, you can define a command synonym name for the EXEC by using the CMDSYN parameter of a command definition. The EXEC name must begin with a nonnumeric character and can be from one to eight characters long. Valid characters are as follows: 0-9 A-Z @ \$ #. Defining EXECs to NetView discussion comes later.

When an EXEC runs, it is read from the partitioned data set. You can load EXECs into storage with the MAPCL and MEMSTORE functions. Discussion of MAPCL and MEMSTORE occurs later.

Avoid using secondary extents when allocating DSICLD data sets. Data sets allocated with secondary extents can create problems for NetView. If the data set goes into secondary extents, you can correct the problem with one of the following actions:

- Issuing the **REACC** command
- Compressing the DSICLD data set by using IEBCOPY in a batch job

You can use the **LISTA** command to list the files that are allocated to the NetView program. Files can be those that are allocated by statements in the JCL and those allocated dynamically by the NetView **ALLOCATE** command.

Common uses of REXX EXECs

Tivoli software

IBM

Common uses of REXX EXECs

- Multiple commands combinable into single program
 - z/OS, NetView, CICS, IMS commands
 - Other REXX EXECs
 - NetView PIPEs
 - And so on
- Simplification of operator keystrokes
- Automation use of REXX
 - Reactive: Respond to an event
 - Proactive: Poll for resource status

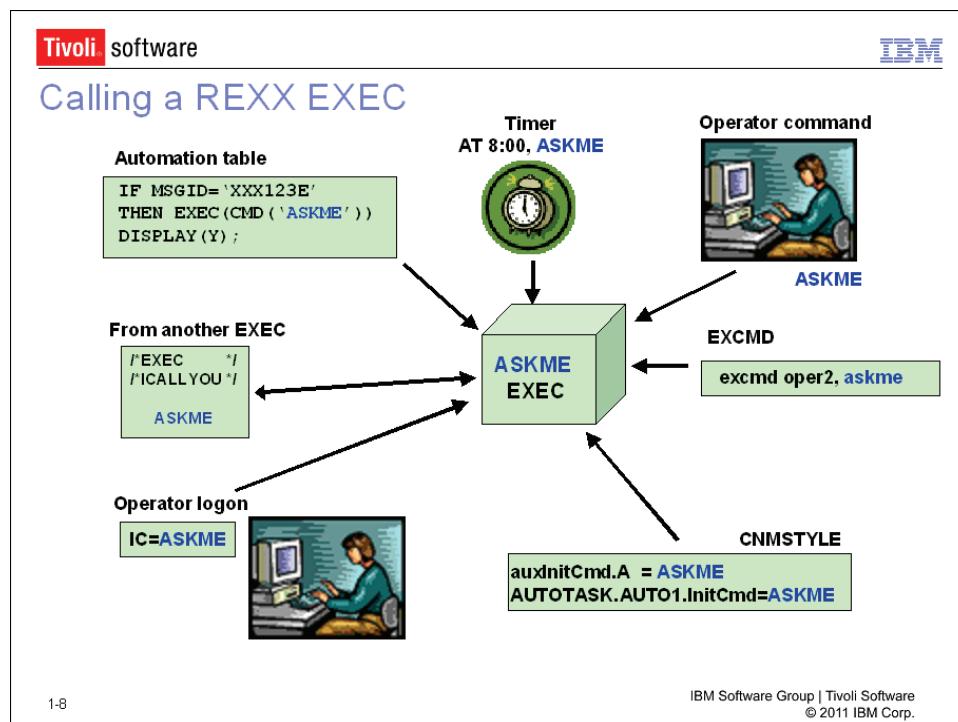
1-7

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Using REXX EXECs, you can combine several commands into one single program, simplifying the number of commands that your operators must issue. EXECs can issue commands, trap and interrogate responses (messages) to the commands, and take more complex actions.

EXECs play a key role in automation. EXECs can be driven directly from an event driving the NetView automation table or scheduled on a timer basis to proactively monitor statuses of resources.

Calling a REXX EXEC



A REXX EXEC can be run in many ways. Calling an EXEC by using some of these methods can force limitations on the commands that can be used within the EXEC. For example, an EXEC that an automated operator runs cannot run full-screen commands.

With an operator station task (OST), an operator can enter commands and receive messages. An OST starts for each operator at logon. This task is used when a NetView operator runs an EXEC. The EXEC can call other EXECs that also run under the OST.

REXX basics (1 of 2)

Tivoli software

IBM

REXX basics (1 of 2)

- First line must be a comment: /* */
- Comments are as follows:
 - Must begin with a /* and end with a */
 - Can start anywhere on a line
 - Can span more than one line (block comment)
- Text strings are as follows:
 - Enclosed within single or double quotation marks
 - 'This is an example of text'
 - "This is another example"
 - Example: SAY "IT'S EIGHT O'CLOCK. TIME TO BRING UP CICS."
- Commands are text strings

1.9

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Each NetView REXX EXEC must begin with a comment. You can also code additional comments in your EXEC wherever necessary.

A series of clauses, each having a separate purpose, make up the functional body of an EXEC. In a simple REXX EXEC, the clauses are interpreted in the sequence that they are coded. The sequence can change by using specific commands that alter the processing order, for example, CALL Label1. (Use quotation marks when you do not want REXX to evaluate the string as a variable.)



Note: Commands are examples of text strings. Place commands within quotations (single or double) so that the REXX interpreter does not perform variable substitution on the string.

Command examples are as follows:

- All NetView commands, including PIPEs
- All MVS commands

REXX basics (2 of 2)

Tivoli software IBM

REXX basics (2 of 2)

- Variable examples
 - TotalCnt = Cntr1 + Cntr2
 - Stem.i = 5
 - CmdString = 'MYCMD' var1 var2
- Command examples
 - 'MVS D A,L'
 - 'DISPLAY NET,PENDING'
 - 'MYCMD' var1 var2
 - 'MYCMD PARM1='var1' PARM2='var2'
- Continuation example

Say 'You can use a comma',
'to continue to another line.'
- Label examples
 - Parse_and_Validate:
 - Failure:

Commands should be enclosed in quotation marks. Otherwise, REXX treats them as variables.

1-10 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This slide illustrates examples of variables, commands, continuation, and labels.

'MYCMD' var1 var2 example:

- Suppose variable *var1* has a value of 1 and variable *var2* has a value of 2.
- When MYCMD is called, it would be passed 1 for the first parameter and 2 for the second parameter.
- In this case, MYCMD is the name of a command or another EXEC.

You should enclose the text string, MYCMD, in quotation marks to prevent the REXX interpreter from treating the command name as a variable. If you also had a variable named *mycmd* and omitted quotation marks, the REXX interpreter substitutes the value of the variable as the command name.

Parsing input parameters

Tivoli software 

Parsing input parameters

- Parse input parameters
 - REXX parse instruction
 - Blank-delimited by default
 - Can change with parse template
 - NetView functions
 - Delimited by blanks, commas, quotation marks, and equal sign
- Use REXX `parse source` to access the command name
 - `parse source . Invoc Command_name .`
- Use REXX parse as an alternative to NetView function calls

1-11

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

EXECs can accept input parameters. The input parameters can be parsed with REXX parsing instructions or NetView functions. REXX parsing is blank-delimited, by default, but can be changed with a parse template that is specified on the **PARSE** instruction.

Using the REXX **PARSE** instruction keeps the EXEC running under the REXX interpreter, which can improve the performance of your REXX EXEC. If you use NetView functions, the REXX interpreter is stopped to call the NetView function.

Use the REXX **parse source** to parse the name of the EXEC (command name). The command name can be used when displaying error messages from the EXEC. REXX also provides a `CmdName()` function to parse the command name.

You can use periods within a parse statement as a placeholder to skip one or more tokens in the text. The parse source example uses periods as follows:

1. Ignore the first token.
2. Place the value of the second token into the *Invoc* variable.
3. Place the value of the third token into the *Command_name* variable.
4. Ignore all remaining tokens.

Parsing input parameters with REXX

Tivoli software

IBM

Parsing input parameters with REXX

- REXX parsing must be done by the EXEC
- Several options
 - **Parse arg argstring** or **arg argstring** or **Arg()**
 - Assigns entire input string to argstring variable
Use **parse var** or **parse value** to further parse argstring variable
 - Use **parse upper** if uppercase is required
Simplifies validation of input parameters
 - By default, REXX parse is blank-delimited
Can change by specifying a parse template

1-12

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXX parsing is flexible. By default, input parameters are blank-delimited. Suppose an EXEC, MYEXEC, is called with four input parameters: **parm1 parm2 parm3 4,5**

- **parse arg argstring**
Sets the variable, argstring, to the entire input string
- **parse arg p1 p2 p3 p4**
Sets p1 to the text string *parm1*
p2 to the text string *parm2*
p3 to the text string *parm4*
p4 to the text string *4,5*

Arg argstring is a shortened version of a **parse arg argstring** instruction. You can use either format. REXX also provides an Arg() function to parse input parameters.

Parsing input parameters with NetView

Tivoli software

IBM

Parsing input parameters with NetView

- NetView parses input parameters automatically
- You can retrieve with following functions:
 - PARMCNT(): Number of input parameters
 - MSGVAR(n): Value of nth parameter
 - Delimited by blank, comma, single quotation mark, and equal sign
 - Maximum of 31 parameters
 - MSGITEM(n): Current message information delimited by blank or comma
 - MSGCNT(): Number of items in a message string
- Note: Input parameters might be a message from the automation table

1-13

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

When a REXX EXEC is invoked, NetView automatically parses the input to the EXEC and sets the values of several NetView functions.

PARMCNT() identifies the number of input parameters. You can use PARMCNT() to test for the exact number of input parameters and issue a message to the user if an error occurs.

MSGVAR(*n*) can be set to one of the following parameters:

- An input parameter to the EXEC
- A parameter of a message that is read from the trapped message queue

Delimiters that are used for setting these NetView functions are comma, apostrophe, blank, or equal sign.

Sending messages to operators

Tivoli software 

Sending messages to operators

- Use REXX **SAY** instruction to display a single-line message to a NetView operator
- Use NetView **WTO** command to display a single-line message on the system console
- Use NetView **WTOR** command to display a single-line message reply on the system console
 - EXEC waits for response to the reply
 - You take action based on response
- NetView PIPES provide similar functions, plus support for multiline messages

1-14 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

You can use the REXX **SAY** instruction to send single-line messages to the NetView operator. If a REXX EXEC is issued within a pipeline, any output that the SAY instruction produces becomes input for the next PIPE stage.

You can send single-line messages to the z/OS system console by issuing NetView **WTO** or **WTOR** commands. The **WTOR** command causes the EXEC to wait for a response to the message. The response is contained in the **WTOREPLY** variable.

Using the SAY instruction

Tivoli software **IBM**

Using the SAY instruction

- Write data to screen (maximum of 32,728 characters)
 - Messages
 - Instructions to the NetView operator
- If first token contains a message ID
 - Can satisfy an outstanding WAIT or result in a match in automation table
- Use SAY to describe expected NetView operator input before PARSE EXTERNAL



IBM Software Group | Tivoli Software
© 2011 IBM Corp.

The REXX **SAY** instruction is useful for communicating information to the NetView operator running an EXEC. The **SAY** instruction generates a single-line message that drives the NetView automation table.



Note: Be careful when using a message ID in the first token of a SAY instruction. The message ID might generate an event that is trapped erroneously in the automation table, for example. To avoid this, a common practice is to prepend text (for example, three asterisks) to the message ID.

If your EXEC needs to pause for input from the operator, issue a say instruction to describe the required input. Follow the instruction with the **PARSE EXTERNAL** instruction to parse the operator input.

REXX example

The screenshot shows a window titled "REXX example". The window has "Tivoli software" in the top left and the IBM logo in the top right. The main area contains REXX code:

```
/* REXX */  
/* SIMPLE EXAMPLE: */  
/* Display input to EXEC with a SAY */  
/* This EXEC can run in TSO and NetView */  
  
parse arg p1 .  
  
IF p1= '' THEN /* no input? */  
  SAY 'Required input parameter missing'  
ELSE  
  Say 'Input parameter was: ' p1  
  
exit
```

At the bottom left is the page number "1-16" and at the bottom right is the copyright notice "IBM Software Group | Tivoli Software © 2011 IBM Corp."

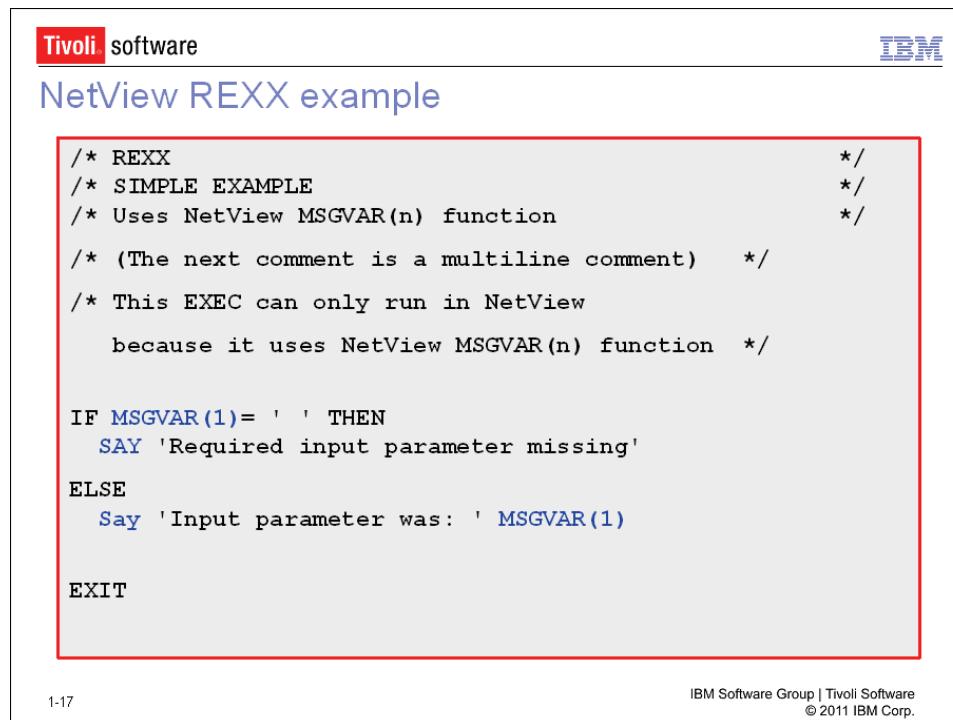
This example REXX EXEC does not use any NetView-specific instructions, functions, or commands. The first four lines are single-line comments. (Note that comments can span multiple lines.)

The **parse arg** instruction parses the input parameters. In this case, the first token is placed into local variable *p1*. All remaining parameters are ignored because of the placement of the period at the end of the statement. The remaining statements in this example EXEC test if variable *p1* is null.

- If *p1* is null, a message routes to the operator: **Required input parameter missing**
- If *p1* is not null, its value is displayed in a message to the operator: **Input parameter was: 5**

Optionally, you can specify a return code on the **exit** instruction. For example, if variable *p1* was null, you can exit with a return code of 8 by coding **exit 8**.

NetView REXX example



The screenshot shows a window titled "NetView REXX example". The window contains REXX code with a red border around the main body. The code is as follows:

```
/* REXX */  
/* SIMPLE EXAMPLE */  
/* Uses NetView MSGVAR(n) function */  
/* (The next comment is a multiline comment) */  
/* This EXEC can only run in NetView  
   because it uses NetView MSGVAR(n) function */  
  
IF MSGVAR(1)= ' ' THEN  
  SAY 'Required input parameter missing'  
ELSE  
  Say 'Input parameter was: ' MSGVAR(1)  
  
EXIT
```

At the bottom left is the number "1-17". At the bottom right is the text "IBM Software Group | Tivoli Software © 2011 IBM Corp."

This is the same REXX example, but using the NetView MSGVAR() function to parse the input to the REXX EXEC. NetView sets MSGVAR(1) automatically to the value of the first input parameter without requiring explicit parsing. Note that lines five and six are a multiline comment.

Pausing for operator input

Tivoli software **IBM**

Pausing for operator input

1. You issue a SAY to request input
`Say 'PLEASE ENTER DATA'`
2. You pause EXEC processing with
`PARSE External p1 p2`
3. "P" is displaced at top right of screen to indicate command is paused
4. EXEC processing stops until operator enters
 - GO command with input
 - CANCEL command
5. PARSE EXTERNAL retrieves data from the command line

`GO MYDATA 123456789`

???

1-18 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

You can use **PARSE EXTERNAL** or **PARSE PULL** instructions to wait for input from the operator. You should issue a **SAY** command to describe the expected input.

If data exists on the REXX *data stack*, it is read before any operator input from the command line if you use **PARSE PULL** or **PULL**. Use **PARSE PULL** for mixed-case text and **PULL** for upper-case text.

To ensure that the data is read from the operator only, use **PARSE EXTERNAL** to place the input in a specified variable. For example, suppose you need an operator to respond with YES or NO before continuing processing within a REXX EXEC. Issue a SAY command with instructions, then a **PARSE EXTERNAL** immediately after the SAY:

```
SAY 'ENTER "GO YES" OR "GO NO" TO CONTINUE'  
PARSE EXTERNAL ANSWER
```

The local variable, *answer*, should contain YES or NO. To require the input in upper case, code as follows:

```
PARSE UPPER EXTERNAL ANSWER
```

Lesson 2: REXX and NetView

Tivoli software



Lesson 2: REXX and NetView

- REXX environments
- Defining EXECs to NetView
- REXX and NetView tasks
- REXX and NetView functions
- Nesting EXECs

1-19

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXX addressing environment

Tivoli software 

REXX addressing environment

TSO

NETVIEW  **MVS**



- The REXX addressing environment is used to process REXX commands
- NetView is the default addressing environment for NetView REXX
- Environment can be changed using **REXX ADDRESS** command
- For example, you switch to MVS environment to read a file and then switch back to NetView environment:
 - **ADDRESS MVS**
 - **EXECIO ...** or **ADDRESS MVS EXECIO ...**
 - **ADDRESS NETVIEW**

1-20 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXX EXECs that run in NetView automatically are to run in the NetView environment. Commands can also run in environments such as MVS.

In the NETVIEW addressing environment, the entire command string converts to uppercase characters. If you want to issue a command that contains mixed-case text, change the addressing environment to **NETVASIS**:

```
address netvasis 'WTO This is a mixed case message.'
```

The **WTO** command displays *This is a mixed case message.* on the z/OS system console.

You can use the ADDRESS instruction on a single line or multiple lines. Refer to the TSO/E REXX manuals for more information about the REXX ADDRESS instruction.



Note: With DATA REXX, the only environment that the ADDRESS instruction supports is NETVADATA, the host environment for DATA REXX.

Defining EXECs to NetView

Tivoli software

IBM

Defining EXECs to NetView

- Not necessary to define EXECs
They must exist in a DSICLD data set
- Can define command synonyms
 - CMDDEF.RECOVERY.CMDSYN=RCVR,RV
 - RECOVERY: Real EXEC name
 - RCVR and RV are synonyms

1-21

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

EXECs are available for use by operators immediately after they are created. If the EXEC is in an operator override data set, you must issue an OVERRIDE DSICLD command to refresh the data set.

EXECs can be defined to NetView to create more meaningful synonym names or for security. Use the **CMDDEF** definition statement to define EXECs to NetView. In this case, the command name is RECOVERY. It has two synonyms: RCVR and RV. Operators can call the EXEC by issuing RECOVERY, RCVR, or RV.

Because EXECs can issue commands, many customers require a level of security for controlling the operators who have access to the EXECs. The EXEC must first be defined to NetView with the **CMDDEF** statement. Then corresponding NetView Command Authorization Table (CAT) or Security Access Facility (SAF) definitions must be created to restrict access to the EXEC or its parameters.

For more information about CMDDEF, see the *Tivoli NetView for z/OS Administration Reference*. For more information about security, see the *Tivoli NetView for z/OS Security Reference*.

EXECs on an automated operator

Tivoli software 

EXECs on an automated operator

One of the primary purposes for writing an EXEC is providing automation

- Automation EXECs run on an automated operator
- Routes directly from the automation table
- Routes from another task with EXCMD command

1-22

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Automation is typically performed on an *automated operator*. REXX EXECs can be specified as the automated actions driven from the automation table on the automated operator. Automated operators do not support the following items:

- Full-screen commands
- Keyboard functions, such as PF keys
- AUTOWRAP

EXECs under the PPT

Tivoli software

IBM

EXECs under the PPT

EXECs can be run under the NetView primary programmed operator interface task (PPT) as follows:

- Automation table
- Routed from another task with EXCMD
 - For example, EXCMD PPT, MVS START TCPIP
- Additional PPT processes:
 - Timers
 - For example, EVERY 24:00:00, PPT, ID=Daily, MONON PU1
 - NetView initialization
 - From CNMSTYLE processing
- Only one PPT task, which is associated with network NetView

1-23

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

The primary program operator interface task (PPT) receives the following message types:

- Unassigned, unsolicited messages from VTAM
- Message traffic that are exchanged between the z/OS system console and VTAM.

The PPT opens a (primary) programmed operator interface (POI) to VTAM to receive these messages.

Each NetView program has one PPT. The PPT starts when the NetView program starts. NetView cannot run without the PPT. If automated tasks (autotasks) are not active during initialization, the PPT task can be used to run critical NetView EXECs.



Note: Ensure that the PPT is not overly utilized. Otherwise, NetView (network) performance problems might occur.

The primary purpose of the PPT is handling VTAM unsolicited messages. At network startup, or times of network problems, such messages might be significant. PPT output is displayed on the z/OS system operator console if it is not routed elsewhere.

The PPT does not support the following commands:

- AUTOWRAP
- BGNSESS
- INPUT
- LOGOFF
- ROUTE
- SET
- SUBMIT
- The following REXX instructions under the PPT:
 - FLUSHQ
 - MSGREAD
 - PARSE EXTERNAL
 - PARSE PULL if there is nothing in the REXX data stack
 - PULL if there is nothing in the REXX data stack
 - TRAP
 - WAIT

EXECs on another NetView

Tivoli software **IBM**

EXECs on another NetView

- RMTCMD examples
 - RMTCMD DOMAIN=NETV2,RECOVERY
 - RMTCMD LU=NETV2,OPERID=AUTO1,SHUTD
- Labeled command examples
 - NETV2: RECOVERY
 - NETV2/AUTO1: SHUTD
- Target operator is logged on as a distributed autotask.
Process starts if operator not already logged on

```
graph LR; A[NetView] -- "RMTCMD autotask started" --> B[NetView]
```

1-24 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

EXECs can run in other NetView domains that use the RMTCMD command. The EXEC must be defined in the DSICLD concatenation of the remote NetView domain.

The RMTCMD command routes system, subsystem, and network commands to a remote NetView domain for processing. The responses to these commands are returned to the RMTCMD issuer. To do this, RMTCMD can use an existing task, if it is already active. If the task specified by the OPERID value is not active, RMTCMD processing automatically starts the task. The command specified with RMTCMD is processed by the task. Responses are returned to the RMTCMD issuer.

RMTCMD supports SNA LU 6.2 and TCP/IP session connectivity.

%INCLUDE with interpreted REXX

Tivoli software **IBM**

%INCLUDE with Interpreted REXX

- Create members of the DSICLD data definition with segments of REXX code
- Use the %INCLUDE capability that the NetView program provided
- Code %INCLUDE function on a separate line, followed by the member name you prepared

1-25

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

To facilitate code reuse, you can create members of the DSICLD data definition with segments of REXX code and use the %INCLUDE capability that the NetView program provides. To include a prepared code segment, the first line of your program must begin with /*%NETVINCL, followed by comments of your choice.

Code %INCLUDE anywhere in your program on a separate line, followed by the member name that you prepared. The prepared code segment is present when your program is interpreted as if it had been imbedded in your program.



Important: Do not use the INCLUDE function if you intend to compile your REXX program.

Nested EXECs

Tivoli software **IBM**

Nested EXECs

EXECs can call other EXECs

- Directly, by name
Variables not shared
- With the CALL instruction
 - CALL can invoke an external program
 - Behaves like a program subroutine
 - External program can be REXX or assembler
 - Variables can be shared between programs
 - Called program has access to all variables of the caller
 - Caller has access to all variables used by called program
 - Can be controlled with procedure expose instruction
 - CALL can also invoke a subroutine within the EXEC

```
graph TD; MYPGM1[MYPGM1] --> MYPGM2[MYPGM2]; MYPGM1 --> MYPGM3[MYPGM3]; MYPGM2 -- "Return_Code=0" --> Exit[Exit Return_Code]; MYPGM3 -- "Return_Code=0" --> Return[Return My_Var];
```

1.26 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXX EXECs can call other REXX EXECs with operands that are passed from the calling EXEC to the nested EXEC as follows:

- Directly using the EXEC name. When the nested EXEC finishes, only a return code routes to the calling EXEC.
- Using the CALL instruction. In this case, data can return in a variable and the return code can be checked.

To pass variables between the calling EXEC and the nested EXEC, you can use NetView global variables, PIPE SAFE, or PIPE INSTORE.

External call example

Tivoli software IBM

External call example

```
/* PROG1 REXX */
.
.
P1 = 1
P2 = 5
Call prog2 p1 p2 'This is p3'
Say 'Total =' Result
.
.
Exit
```

```
/* PROG2 REXX */
PARSE UPPER ARG R1 R2 MSG
.
.
Say 'Parameter 1 =' r1
Say 'Parameter 2 =' r2
Say 'Parameter 3 =' MSG
Sum_Total = r1 + r2
.
.
Return Sum_Total
```

When PROG1 runs, operator detects:

Parameter 1 = 1
Parameter 2 = 5
Parameter 3 = THIS IS P3
Total = 6

PARSE UPPER converts value of third parameter to uppercase.

REXX automatically sets Result variable

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This example shows a use of the CALL instruction. PROG1 calls PROG2 with three parameters:

1. 1
2. 5
3. This is p3

PROG2 parses the input parameters into three variables, displaying them with a SAY instruction. Note that the third parameter is a text string and is also converted to upper case, based on the PARSE UPPER instruction in PROG2.

When an operator runs PROG1, the following information is displayed:

Parameter 1 = 1 (from the SAY in PROG2)
Parameter 2 = 5 (from the SAY in PROG2)
Parameter 3 = THIS IS P3 (from the SAY in PROG2)
Total = 6 (from the SAY in PROG1)

The CALL instruction supports a maximum of 16 nested EXECs. **SIGNAL ON HALT** detects an error in a nested EXEC and terminates all EXECs. Discussion about SIGNAL ON conditions occurs in more detail later.

NetView add-ons

Tivoli software

IBM

NetView add-ons

- NetView provides many commands and built-in functions for NetView REXX EXECs only (NetView environment).
 - Commands: GLOBALV, TRAP, WAIT, and more
 - Built-in functions: OPID(), SYSID(), JOBNAME(), and more
- See *Programming: REXX and the NetView Command List Language* manual for more information.

1-28

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

NetView provides several instructions to perform NetView specific activities. The NetView-provided instructions can be used only in EXECs that run in a NetView environment. Examples are as follows:

- TRAP: Defines messages that are to be trapped, and specifies whether or not the messages should be displayed to the operator.
- MSGREAD: Causes NetView to read a trapped message from the message queue.
- FLUSHQ: Removes all trapped messages from the message queue that using MSGREAD had not removed.
- WAIT: Temporarily suspends processing of that command procedure until a specified event occurs.
- WAIT CONTINUE: Specifies to continue waiting for additional messages, operator input, or data before command procedure processing resumes.
- GLOBALV: Enables you to define, put, get, automate, and trace global variables in REXX and NetView EXECs.

Discussion about global variables and trapping and processing messages occurs later.

REXX functions that NetView provides

Tivoli software



REXX functions that NetView provides

- OPID(): Returns the NetView task ID
- DOMAIN(): Returns the NetView domain ID
- NETVIEW(): Returns NetView version and release
- SYSID(): Identifies z/OS system that originated the message
- CURRSYS(): Returns the one to eight character current system name
- MVSLEVEL(): Returns level of z/OS
- TASK(): Returns three character string indicating type of task (PPT, OST, and so on)
- SYSPLEX(): Returns the name of the z/OS sysplex
- TOWER(string): Returns a binary value indicating if a tower or subtower is enabled (1) or disabled (0)
- TOWER(*): Returns the list of enabled towers and subtowers
- MSGID(): ID of a trapped message or a message from the automation table
- ROUTECDE(): Returns the z/OS route codes assigned to the message
- JOBNAME(): Returns the z/OS job name that originated a message

[For more, see Programming: REXX and the NetView Command List Language](#)

1-29

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

NetView provides functions to perform NetView-specific activities. Because these are NetView-specific and not standard REXX functions, you can use them only in EXECs that run in a NetView environment. NetView provides functions as follows:

- Translation
- EXEC information
- Cross-domain information
- Data set information
- Message processing information
- REXX Management Services Units (MSU) information
- Operator information
- Session information
- REXX environment information
- Console information

LISTVAR example

Tivoli software

IBM

LISTVAR example

- LISTVAR uses several NetView REXX functions
- CNM353I LISTVAR : OPSYSTEM = MVS/ESA
- CNM353I LISTVAR : MVSLEVEL = SP7.1.2
- CNM353I LISTVAR : ECVTPSEQ = 01011200
- CNM353I LISTVAR : CURSYS = ADCD1
- CNM353I LISTVAR : VTAMLVL = V61C
- CNM353I LISTVAR : VTCOMPID = 5695-11701-1C0
- CNM353I LISTVAR : NetView = Tivoli NetView for z/OS V6R1
- CNM353I LISTVAR : NETID = ADCD
- CNM353I LISTVAR : DOMAIN = CNM01
- CNM353I LISTVAR : APPLID = CNM01016
- CNM353I LISTVAR : OPID = NETOP1
- CNM353I LISTVAR : LU = SC0TCP22
- CNM353I LISTVAR : TASK = OST
- CNM353I LISTVAR : NCCFCNT = 2
- CNM353I LISTVAR : HCOPY =
- CNM353I LISTVAR : IPV6ENV = MIXED
- CNM353I LISTVAR : TOWERS = NPDA NLDM TCPIPCOLLECT TEMA IPMGT NVSOA DISCOVERY
- CNM353I LISTVAR : CURCONID =

1-30

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

The slide shows an example of the NetView **LISTVAR** command. **LISTVAR** displays information that several of the NetView REXX functions contain: APPLID(), OPID(), and others. Issue **BROWSE LISTVAR** to view more details on the LISTVAR command and functions that it uses.

Commonly used REXX built-in functions (1 of 2)

Tivoli software

IBM

Commonly used REXX built-in functions (1 of 2)

- DATE(option): Returns value of date in several formats
 - DATE('U'): mm/dd/yyyy
 - DATE('E'): dd/mm/yyyy
 - DATE('W'): Returns current day of week
- TIME(option): Returns local time
 - TIME('N'): hh:mm:ss
 - TIME('C'): 1:12pm
 - TIME(): 13:12:25
- SUBSTR(string,start,length[,pad]): Returns substring of string that begins at the start position for length characters, right padded with pad character
- STRIP(string[[,option][,char]]): Removes leading, trailing, or both leading and trailing characters (char) from string
Strip(input): Removes leading and trailing blanks from variable input
- POS(needle,haystack[,start]): Returns position of one string, needle, in another, haystack, beginning at start position

1-31

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

You can use many REXX built-in functions in your NetView REXX EXECs. This slide shows examples of using the DATE() and TIME() built-in functions.

You can manipulate text strings by using the SUBSTR(), STRIP(), and POS() built-in functions. For example, after parsing a text string, remove all leading and trailing blanks from the string by coding as follows:

```
Input = Strip(input)
```

For more information about these and other REXX functions, see *TSO/E REXX Reference Guide*.

Commonly used REXX built-in functions (2 of 2)

Tivoli software

IBM

Commonly used REXX built-in functions (2 of 2)

- COMPARE(string1,string2,pad): Compares two character strings, returns position of first character that does not match
- SYMBOL(variable): Returns VAR if variable is defined, LIT if variable is not
(very important when coding NetView PIPEs)
- WORDS(string): Returns total number of blank-delimited words in string
- WORD(string,n): Returns the nth blank-delimited word in string
- LENGTH(string): Returns the length of string
- Other functions for converting decimal, hexadecimal, and character strings
 - D2X(), X2D()
 - C2X(), X2C()
 - D2C(), C2D()

1-32

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This slide shows more REXX built-in functions that you can use for manipulating text strings. The SYMBOL() function is very useful when using REXX with NetView PIPEs. In some cases, after using a PIPE, you need to test a variable. PIPEs resets variables and, if not set within the pipeline, a NOVALUE condition can occur. To avoid the NOVALUE condition, you can use the SYMBOL() function to test if the variable is set (VAR) or not (LIT). An example is as follows:

```
A = 1
'PIPE NETV command | ... | VAR A'
if SYMBOL(A) = 'LIT' then
    SAY 'Error encountered - variable A is not set'
```

Student exercise

Tivoli software

Student exercise



1-33

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Open your *Student Exercises* book and perform Exercise 1.

Lesson 3: Variables

Tivoli software

IBM

Lesson 3: Variables

- REXX reserved variables
 - **RC** is set by REXX as the return code after every instruction
 - **Result** is set after a CALL instruction
 - **SIGL** is set after a SIGNAL ON condition has been satisfied
 - Your programs should not set these variables
- User variables: Standard REXX
 - Variables available to local EXEC only
 - Also called local variables
 - Can be stem variables
 - Each EXEC has its own set of local variables
- Global variables: NetView specific
 - Variables available to local EXEC, other EXECs on the same task, or other EXECs on other NetView tasks
- **INTERPRET** instruction, usable for dynamically building variable name and value

1-34

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

The three types of variables in REXX EXECs are as follows:

- Reserved variables, which REXX sets and that your EXECs should not use.
- User variables, which are the variables that are used within your EXECs.
- Global variables, which are specific to NetView, providing a mechanism for sharing data between EXECs on one task or EXECs on several tasks.

Variables in REXX

Tivoli software 

Variables in REXX

- Mixed case
- Local variables: Name can be one to 255 alphanumeric characters
- Global variables: Name can be one to 31 uppercase alphanumeric characters
- Stem variables supported
- Can be any of the following:
 - A through Z
 - Zero through nine
 - Special characters: . , #, @, \$, _, !, or ?
- First character not to be a number or a period

1-35

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Products from IBM prefix their global variables with a three-character product prefix (for example, CNM or FKX). You can use any other global variable names. If you use a naming convention, you can issue a **QRYGLOBL** command to help display them more easily.

Overview of NetView global variables

Tivoli software 

Overview of NetView global variables

You can share data between EXECs running on a single task or multiple tasks

- Two types of NetView global variables
 - Task: Shared across EXECs that run on the same task
 - Common: Shared across EXECs that run on all NetView tasks
- **GLOBALV** command to do the following items:
 - Define global variables
Optional, initial value is null
 - Set or retrieve value
 - Save, restore, or purge saved variables
 - Trace global variables
 - Automate on changes to global variables
- **QRYGLOBL** command to display information



1.36 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

NetView provides global variables for use in REXX EXECs.

- Task global variables can be used for sharing data between EXECs that are running on the same task.
- Common global variables can be used for sharing between EXECs that are running on different tasks.

The NetView **GLOBALV** command has several operands to define, set, retrieve, save, restore, trace, automate, and purge saved variables. The NetView **QRYGLOBL** command can display the values of task or common global variables. QRYBALV and QRYGLOBL discussions occur later in this lesson.



Tip: When using GLOBALV in a REXX EXEC, remember to enclose it in single or double quotes.

GLOBALV command parameters (1 of 2)

Tivoli software 

GLOBALV command parameters (1 of 2)

- **PUTT** or **PUTC**: Store global variable value
Stores value of local (user) variable into NetView global dictionary
- **GETT** or **GETC**: Get global variable value
- **SAVET** or **SAVEC**: Save to Save/Restore VSAM file
Wildcard supported (*)
- **DEFT** or **DEFc**: Defines global variable, does not alter value
 - Defines variable with null value
 - Other EXECs can access the variable after defined
 - Optional
- **RESTORET** or **RESTOREC**: Restore from Save/Restore VSAM file
Wildcard supported (*)
- **PURGET** or **PURGEC**: Purge in Save/Restore VSAM file
Wildcard supported (*)

1-37 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

GLOBALV is a NetView command for managing task and common global variables. This section discusses the parameters.

GLOBALV PUTT | PUTC:

Use PUTx to create or update a global variable in the NetView *dictionary* and save its value.



Tip: The local REXX variable must be set to a value before issuing the PUTx. When an EXEC updates the value of the local REXX variable, it should issue another PUTx.

GLOBALV GETT | GETC:

Use GETx to retrieve the value of a global variable. If the global variable is not set, its value becomes null, and the GLOBALV command return code is 144.

```
-----  
V      !  
>>-'GLOBALV --+GETT+---- variable---'-----><  
     '-GETC-'
```

GLOBALV DEFT | DEFC:

Use DEFx to define a global variable in the NetView *dictionary*. It has an initial value of null.

```
-----  
V      !  
>>-'GLOBALV --+DEFT+---- variable---'-----><  
     '-DEFC-'
```

GLOBALV SAVEx | RESTOREx | PURGEx:

Use these options to save, restore, or purge global variables in the NetView Save/Restore VSAM file. SAVEx, RESTOREx, and PURGEx support **wildcards** with an asterisk (*).

```
-----  
V      !  
>>-'GLOBALV --+SAVET+---- variable---'-----><  
     '-SAVEC-'    +- var*-----+  
           '_ *-----'
```

```
-----  
V      !  
>>-'GLOBALV --+RESTORET+---- variable---'-----><  
     '-RESTOREC-'   +- var*-----+  
           '_ *-----'
```

```
-----  
V      !  
>>-'GLOBALV --+PURGET+---- variable---'-----><  
     '-PURGEc-'    +- var*-----+  
           '_ *-----'
```

GLOBALV command parameters (2 of 2)

Tivoli software 

GLOBALV command parameters (2 of 2)

- **AUTOT** or **AUTO_C**: Automate on global variable value
Stores value of local (user) variable into NetView global dictionary
- **TRACET** or **TRACEC**: Trace on global variable value change

1-38 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

GLOBALV AUTO_X:

Use these options to turn on or off automation of global variables. The LIST option displays current automation settings. When a change occurs to a automated variable, a DWO994I message is issued with variable name, old value, and new value of variable. Automation table entries can be coded to process this message. By default, this message is not displayed or logged unless modified by an automation table entry.

```
GLOBALV AUTOT!AUTOC
-----.
. - *-----. V !  
>>-'GLOBALV -+-AUTOT---+----+-----+----+ variable---'-><
     '-AUTOC-' '-OFF-' '- groupID-' +- var-----+
     '- *-----'  
  
GLOBALV AUTOT!AUTOC
-----.
. - *-----.
>>-'GLOBALV -+-AUTOT---+----+-----+-----><
     '-AUTOC-' '-LIST-' +- *ALL----+
     '- groupID-'
```

GLOBALV TRACEx

Use these options to turn on or off trace of global variable changes. The LIST option displays current traces. When a change occurs to a global variable that is being traced, a DWO990I message is issued with name of the global variable, old value, and new value.

```
GLOBALV TRACET!TRACEC
          .-----.
          . - *-----. V           !
>>-'GLOBALV -+-TRACET-++-ON---+-----+--+ variable-++-'-><
     '-TRACEC-'  '-OFF-'  '- groupID-'  +- var*-----+
          ' - *-----'

GLOBALV TRACET!TRACEC
          .-----.
>>-'GLOBALV -+-TRACET-++-END---+-----+'-----><
     '-TRACEC-'  '-LIST-'  +- *ALL----+
          '- groupID-'
```

Task global variables

Tivoli software **IBM**

Task global variables

```
/* REXX */
'GLOBALV DEFT VAR1'      /* define task global var1 */
VAR1 = 'abcdefg'          /* update local VAR1 */
'GLOBALV PUTT VAR1'      /* put value into dictionary */

/* VAR1 is available to defining NetView task only */
```

Each task can have its own *local* and *task global* VAR1 with unique values.

1-39 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Task global variables are available to all EXECs running on one task. This slide example shows the following sequence:

1. Defines VAR1 as a task global variable (optional).
2. Sets the value of local REXX variable VAR1 to a character string.
3. Stores the value of local variable VAR1 into the NetView dictionary as task global variable VAR1.



Note: Each time a global variable updates, you should replace its value in the dictionary. For example, if VAR1 updates later in the same EXEC, another **GLOBALV PUTT VAR1** must be issued to save its current value.

Common global variables

Tivoli software
IBM

Common global variables

```
/* REXX */
'GLOBALV DEFC VAR1'          /* define common global VAR1 */
VAR1 = 'abcdefg'             /* update local VAR1           */
'GLOBALV PUTC VAR1'          /* put value into dictionary */

/* Common global VAR1 is available to any NetView task */
```

Each task can have a *local* VAR1 until it issues a GLOBALV DEFC, GETC, or PUTC for VAR1.

1-40

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Common global variables are available to all EXECs running on all tasks. This slide example shows the following sequence:

1. Defines VAR1 as a common global variable (optional).
2. Sets the value of local REXX variable VAR1 to a character string.
3. Stores the value of local variable VAR1 into the NetView dictionary as common global variable VAR1.

Each time a global variable updates, you should replace its value in the dictionary. For example, if VAR1 updates later in the same EXEC, another **GLOBALV PUTC VAR1** must be issued to save its value. VAR1 can also be updated by another EXEC.

If VAR1 is a counter (for example, counting of the number of times that a resource failed), you should synchronize updates under one task to ensure no loss of data.



Note: You can define VAR1 as a local REXX variable, task global variable, **and** common global variable at the same time.

Global variable notes (1 of 2)

Tivoli software 

Global variable notes (1 of 2)

- The task that is running EXEC maintains local variables in storage
- An internal dictionary that is maintained by NetView stores global variables
Updated with GLOBALV PUTx
- GLOBALV GETx copies value of NetView global variable from the dictionary into local REXX variable of same name
If no global variable exists, null value returned
- GLOBALV PUTx copies local REXX variable to NetView dictionary under global variable of same name

1-41

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Global variables are *snapshots* of local variables that are stored in a dictionary that NetView maintains.



Note: If you put a global variable value to the dictionary and update its value, only the local REXX variable is updated. You must issue another GLOBALV PUTx command to update the dictionary.

Global variable values in the dictionary are not updated dynamically.

Global variable notes (2 of 2)

Tivoli software

IBM

Global variable notes (2 of 2)

- You delete value of NetView global (task or common) variable as follows:
 - Set variable to null (' ') or use REXX **DROP** instruction
 - Issue a **GLOBALV PUTx**
The next **GETx** returns null value
- **PURGE**x deletes from external storage only
- Depending upon use, you might need to serialize common global updates if updates are performed from multiple tasks
For example, counter of failures for a resource



1-42

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

To delete a global variable value, you must set its local variable to null (or use **DROP**). Update the dictionary value with a **GLOBALV PUTx** command. If the global variable is saved to the Save/Restore file, you must issue a **GLOBALV PURGE**x command to remove the global variable from the Save/Restore file.

GLOBALV with stem variables

Tivoli software **IBM**

GLOBALV with stem variables

You must handle PUT and GET stem variables individually, including the stem index variable

```
/* EXEC #1 */
Lcl_Var.0 = 5
Lcl_Var.1 = '1st stem var'
Lcl_Var.2 = '2nd stem var'
Lcl_Var.3 = '3rd stem var'
Lcl_Var.4 = '4th stem var'
Lcl_Var.5 = '5th stem var'

'GLOBALV PUTC LCL_VAR.0 LCL_VAR.1 LCL_VAR.2',
'   LCL_VAR.3 LCL_VAR.4 LCL_VAR.5'
```

Set each stem variable, `Lcl_Var.`, plus the stem index, `Lcl_Var.0`, and issue a GLOBALV `PUTC` for all six

```
/* EXEC #2 */
'GLOBALV GETC LCL_VAR.0'

Do i = 1 to LCL_VAR.0
  'GLOBALV GETC LCL_VAR.'i
  Say 'var num' i 'is ...' value('Lcl_Var.'i)
End
```

Issue a GLOBALV `GETC` for `LCL_VAR.0`.
Loop: Issue GLOBALV `GETC` for each stem variable.

Put six global variables **Output from second EXEC:**
`var num 1 is ... 1st stem var`
`var num 2 is ... 2nd stem var`
`var num 3 is ... 3rd stem var`
`var num 4 is ... 4th stem var`
`var num 5 is ... 5th stem var`

Get six global variables

1-43 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This slide illustrates special handling of REXX stem variables that are used as global variables. EXEC#1 creates a stem variable, `Lcl_Var`, with five elements. Each stem variable (`Lcl_Var.1` to `Lcl_Var.5`) is set to a value and put into the common global dictionary. The index (`Lcl_Var.0`) is also set and put into the common global dictionary.

Before using the stem variable values, EXEC#2 must issue a GLOBALV GETC for each. A typical approach is as follows:

1. Retrieve the stem index (`Lcl_Var.0`).
2. Create a DO loop within the EXEC to retrieve the *n*th element of the stem variable (`Lcl_Var.n`).

This set of actions results in putting six global variables in EXEC#1 and retrieving six global variables in EXEC#2.



Note: For maximum performance, minimize the use of GLOBALV GETC commands in EXEC#2 by building a list of common global variables to retrieve. An example follows:

```
VarList = ''  
Do i = 1 to LCL_VAR.0  
    VarList = VarList || Lcl_Var.i  
End  
VarList = Strip(VarList)  
'GLOBALV GETC 'VarList
```

Displaying global variables

Tivoli software 

Displaying global variables

- Operators can use the **QRYGLOBL** command to display common and task global variables

```
. - BOTH---.
>>-QRYGLOBL--+-+-----+-----+
   +- COMMON-+  '- VARS=varspec'
   '- TASK---'

>-+-----+-----+-----+<
   '- FILE=membername--+-+-----+
   '- MODE=modename-'  '- REPLACE-'
```

- Examples
 - QRYGLOBL COMMON VARS=*: Displays all defined common global variables
 - QRYGLOBL TASK VARS=MYVAR*: Displays all defined task global variables that begin with the characters MYVAR
For the issuing task only



IBM Software Group | Tivoli Software
© 2011 IBM Corp.

QRYGLOBL, which either a NetView operator or an EXEC can issue, can be used for displaying NetView common and task global variables.

QRYGLOBL example

Tivoli software

IBM

QRYGLOBL example

QRYGLOBL COMMON VARS=CNMSTYLE.AUTO.*

BNH031I NETVIEW GLOBAL VARIABLE INFORMATION

BNH031I COMMAND ISSUED AT: 06/16/11 13:54:03

BNH031I

BNH031I COMMON GLOBAL VARIABLES

BNH031I GLOBAL VARIABLE NAME: GLOBAL VARIABLE VALUE:

BNH031I

BNH039I CNMSTYLE AUTO MVSCLDRREVISION MVSCLDRS

BNH039I CNMSTYLE AUTO SERVERDBMAINT DBAUTO01

BNH039I CNMSTYLE AUTO EXCARBOTM AUTO01

BNH039I CNMSTYLE AUTO AUTOIP AUTOPON

BNH039I CNMSTYLE AUTO PETS TCPPIP AUTOPPTS

BNH039I CNMSTYLE AUTO REONDBMAINT DBAUTO02

BNH039I CNMSTYLE AUTO TCP TCPPIP AUTOTCP5

BNH039I CNMSTYLE AUTO NAPOLTSK1 AUTOBC1

BNH039I CNMSTYLE AUTO NAPOLTSK2 AUTOBC2

BNH039I CNMSTYLE AUTO NAPOLTSK3 AUTOBC3

BNH039I CNMSTYLE AUTO NAPOLTSK4 AUTOBC4

BNH039I CNMSTYLE AUTO SWONDBMAINT DBAUTO01

BNH039I CNMSTYLE AUTO OPKT TCPPIP AUTOOPKT

BNH039I CNMSTYLE AUTO COLTSK5 AUTOTC5

BNH039I CNMSTYLE AUTO XCFDISC AUTOXOSC

BNH039I CNMSTYLE AUTO TCPDBMAINT DBAUTO02

BNH039I CNMSTYLE AUTO MEMSTORE AUTO02

BNH039I CNMSTYLE AUTO ENTDATA AUTOEDAT

BNH039I CNMSTYLE AUTO ISOCAPTSK AUTONVSP

BNH039I CNMSTYLE AUTO IDEOF AUTO1

BNH039I CNMSTYLE AUTO NCF AUTONCF

BNH039I CNMSTYLE AUTO COLTSK7 AUTOCT7

BNH039I CNMSTYLE AUTO AP.SERV AUTOTMSI

BNH039I CNMSTYLE AUTO MASTER AUTO1

BNH039I CNMSTYLE AUTO NLCLCLOP AUTONALC

BNH039I CNMSTYLE AUTO POLICY AUTORON

BNH039I CNMSTYLE AUTO TCPCONN.TCPPIP AUTOTCPC

BNH039I CNMSTYLE AUTO NETCONV AUTO2

BNH039I CNMSTYLE AUTO DLAAUTO AUTO2

BNH039I CNMSTYLE AUTO COLTSK6 AUTOCT6

BNH039I CNMSTYLE AUTO PREARY AUTO1

BNH039I CNMSTYLE AUTO EXCAPTSK AUTOPSARV

BNH035I NUMBER OF VARIABLES FOUND 32

BNH037I NETVIEW GLOBAL VARIABLE INFORMATION COMPLETE

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This slide shows an example of using the QRYGLOBL command to list all of the automation operators (autotasks) defined in CNMSTYLE.

Suppose you want to determine which functions are using AUTO2. You can look for the BNH039I messages and correlate a value of AUTO2 with each global variable name. Or, you could code a NetView PIPE similar to the following one:

```
'PIPE NETV QRYGLOBL COMMON VARS=CNMSTYLE.AUTO.*',
  '| CORR',
  '| SEP',
  '| LOC /BNH039I/ ,
  '| LOC / AUTO2/ ,
  '| CONS'
```

The results should look similar to examples as follows:

BNH039I CNMSTYLE.AUTO.NETCONV	AUTO2
BNH039I CNMSTYLE.AUTO.MEMSTORE	AUTO2
BNH039I CNMSTYLE.AUTO.DLAAUTO	AUTO2
BNH039I CNMSTYLE.AUTO.MASTER	AUTO2

NetView PIPEs discussion is in the NetView pipelines training module.

Setting common global variables

Tivoli software 

Setting common global variables

- Operators can set common global variables with the **SETCGLOB** and **UPDCGLOB** commands
 - >> SETCGLOB varname TO value -----><
 - >> UPDCGLOB varname -- BY increment -- MAX maxvalue ---><
- Examples
 - SETCGLOB counter TO 101
 - UPDCGLOB counter BY 3 MAX 69
- UPDCGLOB provides serialization of updates
Values are numeric
- Most common use of SETCGLOB and UPDCGLOB come from the automation table



1-46

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

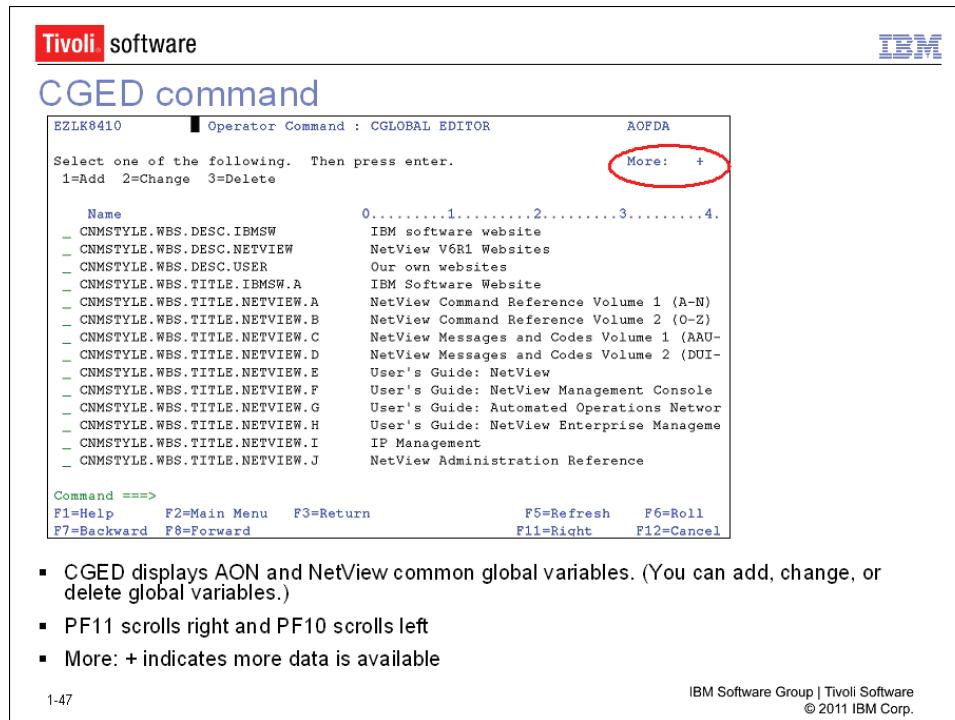
SETCGLOB and **UPDCGLOB** are NetView REXX EXECs for setting common global variables from the automation table. You can use **SETCGLOB** for setting the value of the specified common global variable. Note the following restrictions to this command:

- The use of SETCGLOB is limited to the command procedure and automation environments. (The command must originate in REXX, HLL, automation, or an optional task.)
- The SETCGLOB command sets a common global variable to any value. This command is appropriate to use in any situation where serializing the access among multiple tasks is not important. If serialization of updates is important, use PIPE VARLOAD. If the value is numeric, you can use the UPDCGLOB command.

The **UPDCGLOB** command adds the value that is specified as an increment to the current value of the common global variable. The command runs if the result does not exceed the specified *maxvalue*. If the specified variable is not set (has a null value), it is treated as zero. The new value becomes the value of the increment.

The UPDCGLOB command serializes updates for common global variables by using PIPE VARLOAD to give the effect of a compare and swap logic. If serialization between tasks is not important, you can use the SETCGLOB command instead.

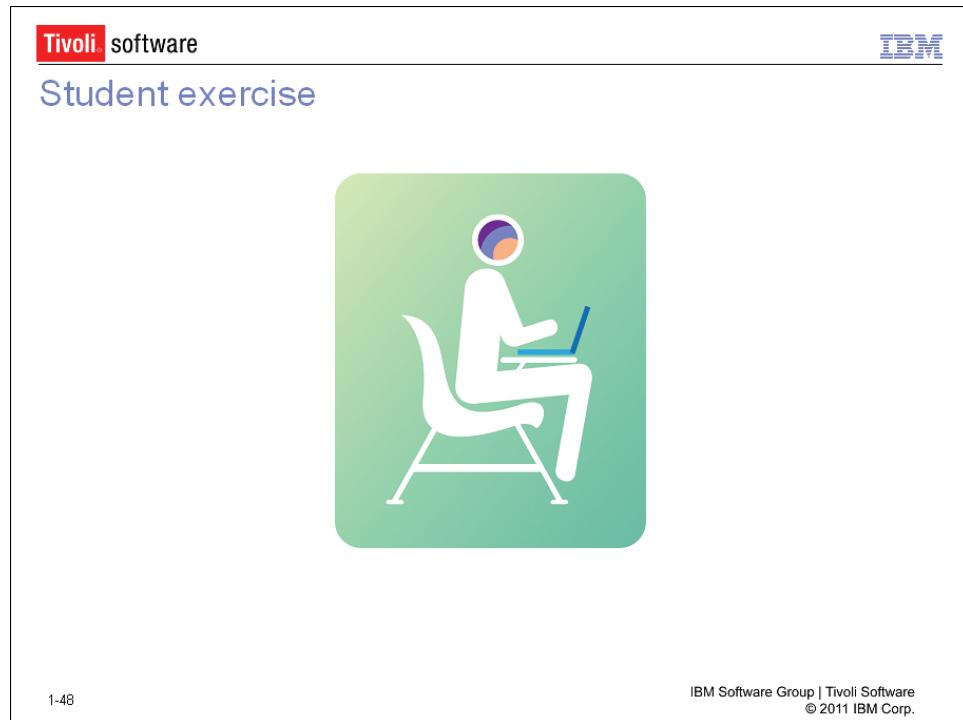
CGED command



CGED displays both AON and NetView common global variables. Options are provided so that you can create new common global variables and delete or modify an existing common global variable. You can also see more information by using your keyboard keys as follows:

- When you see “**More: +**” press PF8 to see more data.
- Press PF10 and PF11 to scroll left and right.

Student exercise



Open your *Student Exercises* book and perform Exercises 2 and 3.

Lesson 4: Processing messages in REXX

Tivoli software

IBM

Lesson 4: Processing messages in REXX

- NetView supplies several instructions to trap and process messages
 - TRAP: Identifies one or more messages to trap
 - Messages must arrive on the task that requests the trap
 - Can be the first line of a multiline message
 - WAIT: Waits to receive messages from a TRAP statement
 - WAIT CONTINUE: Continues to wait for further messages
 - MSGREAD: Reads a message from the trapped message queue
 - FLUSHQ: Purges all messages in the trapped message queue
- Specific to NetView environment only:
Operator (OST), automation operator (AOST), or RMTCMD
- When message is read from trapped message queue, NetView sets several functions, such as MSGID()
- NetView PIPEs provide similar functionality

1-49

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

The **TRAP** instruction causes NetView to monitor the operator task for specified messages. If the messages occur, they are trapped and added to the *trap message queue*. Trapped messages that fulfil a **WAIT** instruction and can be read by using **MSGREAD**. MSGREAD removes the message from the trap message queue and sets several NetView functions with information pertaining to the message. An example is **MSGID()**.

A **WAIT CONTINUE** instruction can be coded to specify that processing should continue to wait for additional messages, operator input, or data before EXEC processing resumes. When processing resumes, the next instruction after the **WAIT CONTINUE** runs.

The **FLUSHQ** instruction is used for removing all trapped messages from the trap message queue that have not been read (removed) using **MSGREAD**.



Note: EXECs that wait for messages result in an error when they run under the PPT task.



Tip: NetView PIPEs provide functions to issue commands and process their responses in a pipeline. *In most cases, you can replace the TRAP, WAIT, and MSGREAD commands (plus the REXX code) with a single pipeline.*

Overview of REXX to trap messages (1 of 2)

Tivoli software 

Overview of REXX to trap messages (1 of 2)

REXX EXECs trapping messages should do the following tasks:

- Issue a TRAP command for defining the messages of interest
 - Both solicited and unsolicited messages can be trapped
 - You must be careful of message assignments and automation table
- Issue the command that is to generate the trapped messages
- Issue a WAIT command to wait for the messages
- Check the type of event that ended the trap
Message is one of the possible event types
- For messages, issue a MSGREAD to access items, such as the message ID and text
- Optionally, code a WAIT CONTINUE to wait for further trapped messages to arrive
- Issue a TRAP NO MESSAGES when finished trapping messages
- Issue a FLUSHQ to purge the trapped message queue to prevent older messages from satisfying a subsequent TRAP

1-50 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

You can trap a single message, multiple messages, or a multiline message. The basic logic for trapping messages is as follows:

1. Define the messages to trap with a TRAP command.
2. Issue the command that generates the messages being trapped.
3. WAIT for the messages.
During the wait, the operator can see a **W** in the upper corner of the screen.
4. Test the type of event that ended the trap.
Discussion of the four different event types is on the next slide.
5. Use MSGREAD to read the message from the trapped message queue.
6. Continue waiting, if necessary, for more messages.
7. End trapping messages when done to prevent newer messages that would fulfill a message trap.
8. Flush the messages in the trapped message queue.

Messages that occur as a result of a command are *solicited* messages. When specifying a TRAP, be aware that *unsolicited* messages can be trapped. You can test if a message is solicited or unsolicited in your REXX EXECs.

Overview of REXX to trap messages (2 of 2)

Tivoli software

IBM

Overview of REXX to trap messages (2 of 2)

- Check return code set for TRAP, MSGREAD, WAIT commands
- Use NetView **EVENT()** function to check event type that satisfied the wait
 - M: Message received
 - T: Timeout occurred
 - E: Error occurred
 - G: Operator entered a GO command
 - Null: No WAIT was coded
- Use **MSGREAD** to read trapped message
 - Removes message from trapped message queue
 - Sets several NetView functions
 - MSGORIGN(), MSGID(), MSGSTR(), MSGVAR(*n*), SYSID(), and so on
 - Simplifies your REXX coding
- Use REXX to parse the message text: PARSE VAR, SUBSTR(), and so on

1-51

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

The REXX-reserved variable, RC, is set after each instruction within an EXEC, including MSGREAD, TRAP, and WAIT instructions.

When a wait is satisfied, you should check the event type that satisfied the wait. In typical operation, the event is message (EVENT()='M'). If the command response does not arrive within the timeout specified on the WAIT instruction, the event becomes a timeout (EVENT()='T').

MSGREAD reads a message from the trap message queue and sets several NetView functions. You can use the NetView functions to parse the message: MSGID(), MSGSTR(), MSGVAR(*n*), and so on. You can also use a REXX PARSE of MSGSTR() to further parse the text of the message.

Functions that MSGREAD sets

Tivoli software

IBM

Functions that MSGREAD sets

- **MSGORIGIN()**: NetView domain where message was generated
- **MSGSTR()**: Text of message, not including message ID
- **MSGID()**: Message ID
- **MSGCNT()**: Count of tokens in MSGSTR()
- **MSGVAR(*n*)**: Message tokens
Maximum of 31 tokens
- Also settable for messages that are routed from the automation table

1-52

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

NetView sets many built-in functions when you issue a MSGREAD command.

Example: Trapping a single message

Tivoli software
IBM

Example: Trapping a single message

```

/*
 * GREETING - SHOW SIMPLE EXAMPLE OF WAITING AND TRAPPING USING THE DATE COMMAND */
/* NOTE: WHEN DATE IS ENTERED, THE FOLLOWING IS RETURNED: */
/* CNM359I DATE : TIME = HH:MM DATE = MM/DD/YX */
/*
*TRAP AND SUPPRESS ONLY MESSAGES CNM359I '           /* TRAP DATE MESSAGE */
'DATE'          /* ISSUE COMMAND */
'WAIT 10 SECONDS FOR MESSAGES'                      /* WAIT FOR ANSWER */
SELECT          /* RESULT IS BACK, PROCESS IT... */
    WHEN (EVENT()='M') THEN                          /* DID WE GET A MESSAGE? */
        DO
            'MSGREAD'                                /* Message received */
            HOUR=SUBSTR(MSGVAR(5),1,2)                /* ... READ IT IN */
            SELECT                                     /* PARSE OUT THE HOUR */
                WHEN (HOUR<12) THEN                  /* GIVE APPROPRIATE GREETING... */
                    SAY 'GOOD MORNING'                 /* ...BEFORE NOON? */
                WHEN (HOUR<18) THEN                  /* ...BEFORE SIX? */
                    SAY 'GOOD AFTERNOON'
                OTHERWISE                            /* ...MUST BE NIGHT */
                    SAY 'GOOD EVENING'
            END                                     /* OF SELECT (APPROPRIATE) */
        END                                     /* Message received */
    WHEN (EVENT()='E') THEN                          /* DID WE GET AN ERROR? */
        SAY 'ERROR OCCURRED WAITING FOR DATE COMMAND RESPONSE'
    WHEN (EVENT()='T') THEN                          /* DID WE GET A TIME-OUT? */
        SAY 'NO MESSAGE RETURNED FROM DATE COMMAND'
    OTHERWISE                                         /* OF SELECT (RESULT) */
END

```

1-53

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This slide shows an example of a TRAP for one message, CNM359I, that results from issuing the NetView **DATE** command. The message is suppressed. When the message is received, the following actions occur:

1. NetView sets the event function: EVENT()='M'
2. The EXEC parses the fifth message token.
3. Based on the value of the fifth token, a greeting message goes out.

This example is also coded to handle errors, EVENT()='E,' and wait timeout conditions, EVENT()='T.'

You can use the following example as a sample for coding many other REXX EXECs to TRAP and WAIT for messages.

1. Modify the TRAP AND SUPPRESS statement.
2. Issue the command to generate the trapped messages.
3. Replace the Select-When logic after the MSGREAD command to parse the trapped messages.

Multiline messages

Tivoli software 

Multiline messages

- NetView treats a multiline write to operator (MLWTO) as a single message
- Only the first line satisfies message trap
 - All other message lines are ignored by trap
 - EXEC can access and parse all lines
- MLWTO can contain message ID in each line or in only the first line (header)
- First column (HDRMTYPE) identifies source
 - ' If MLWTO from NetView
 - " If MLWTO not from NetView (for example, z/OS)
 - = If user generated MLWTO

1-54

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

In addition to single-line messages and multiple single-line messages, multiline messages can be trapped and processed. Multiline messages are commonly called multiline write-to-operator (MLWTO) messages.

Only the first line of an MLWTO can be trapped. REXX EXECs can parse any line of the MLWTO. MLWTO messages can be identified by using the HDRMTYPE function. Some MLWTO messages contain a message ID on every line, making the MLWTO appear to be multiple single-line messages.

A method for parsing MLWTO messages is using NetView PIPEs. Discussion about NetView PIPEs is in the NetView Pipelines training module.

Messages from automation

Tivoli software

IBM

Messages from automation

- The automation table can route messages
- You can drive REXX EXECs to issue commands to perform tasks as follows:
 - Collect more detailed data
 - Perform automated actions
 - Perform root cause analysis
 - Update status of failed resources
 - Notify appropriate personnel of the problem
 - More
- Automated messages set many NetView functions that REXX can access

1-55

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

One of the primary purposes of a REXX EXEC is to take action based on events that drive the NetView automation table. Actions can range from collecting more data (pertaining to the event or the resource) to notifying appropriate personnel of the failure. For example, you can use one of the two following notification methods:

- SMTP on z/OS for sending an email
- **System Automation for Integrated Operations Management** (SA IOM) for sending an email or voice notifications

Subset of functions that automation sets

Tivoli software



Subset of functions that automation sets

- **MSGORIGIN()**: NetView domain where message was generated
- **MSGSTR()**: Text of message, not including message ID
- **MSGID()**: Message ID
- **MSGTSTMP()**: Message time stamp in form of hhmmss
- **MSGCNT()**: Count of tokens in MSGSTR()
- **MSGVAR(n)**: Message tokens
Maximum of 31 tokens
- **HDRMTYPE()**: One-character NetView buffer type of the received message
For example, solicited versus unsolicited
- **JOBNAME()**: One- to eight-character z/OS job name identifier
- **JOBNUM()**: Eight-character z/OS job number identifier
- **MCSFLAG()**: Returns of the system message flags in a series of eight on (1) and off (0) EBCDIC characters representing the bits in order
- **AUTOTOKE()**: One- to eight-character name of the MPF automation token
- More

See Programming: REXX and the NetView Command List Language for complete list

1-56

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

When a REXX EXEC is called from the automation table, many functions are set with information that pertain to the event. MSGID() and MSGSTR() have been discussed in relation to trapping messages. They work the same way when called from the automation table. More details about the functions that the automation table processes set are in the *IBM Tivoli NetView for z/OS: Programming: REXX and the NetView Command List Language* manual.

Student exercise

Tivoli software

IBM

Student exercise



1-57

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Open your *Student Exercises* book and perform Exercise 4.

Lesson 5: Analyzing problems

Tivoli software

IBM

Lesson 5: Analyzing problems

- Use tracing to aid with debug of EXECs
- Code REXX TRACE instruction within an EXEC
Options: C, E, F, I, L, O, R
- Use RXTRACE command to manage traces for most NetView EXECs
Displays panel to set trace options by operator, NetView domain, or specific EXEC name

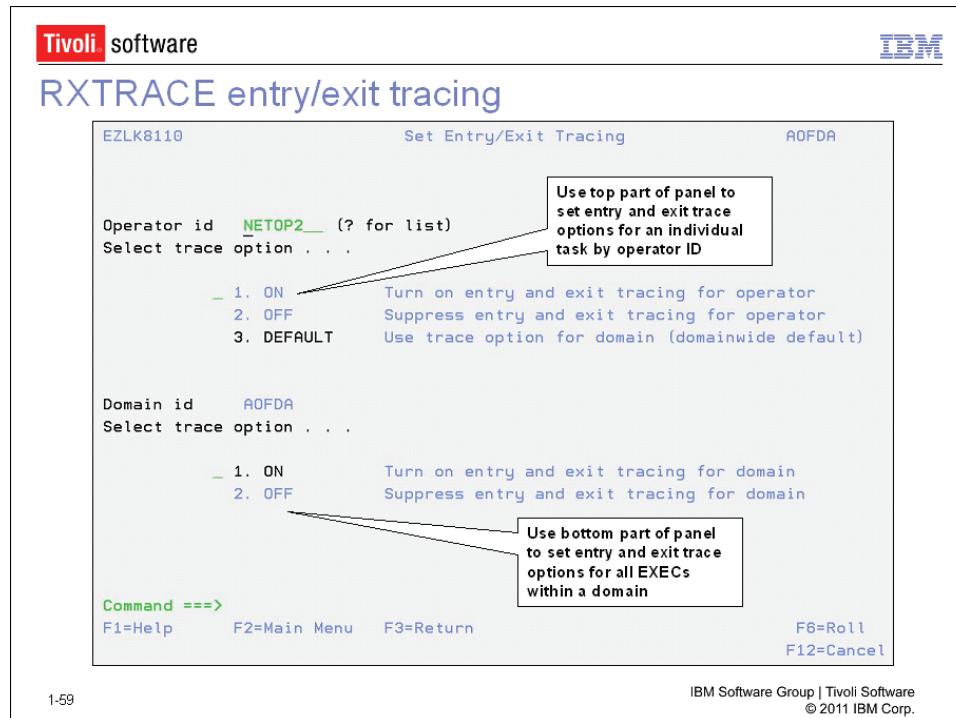
1-58

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Supported REXX TRACE options are as follows:

- **C** (Command): All commands are traced before they run, and any error return code from the commands is displayed.
- **E** (Error): Any command that runs and results in error or failure is traced afterward. Any error return code from the command is also displayed.
- **F** (Failure): Any command that runs and results in failure is traced.
- **I** (Intermediate): All clauses are traced before they run, and any intermediate results are traced during expression evaluation and substitution.
- **L** (Label): All labels that are passed during execution are traced, making it convenient to note all subroutine calls and signals.
- **O** (Off): All trace is turned off, and any previous trace settings are reset.
- **R** (Result): All clauses are traced before they run, and final results of evaluating expressions are traced. This option is useful for general debugging.

RXTRACE entry/exit tracing



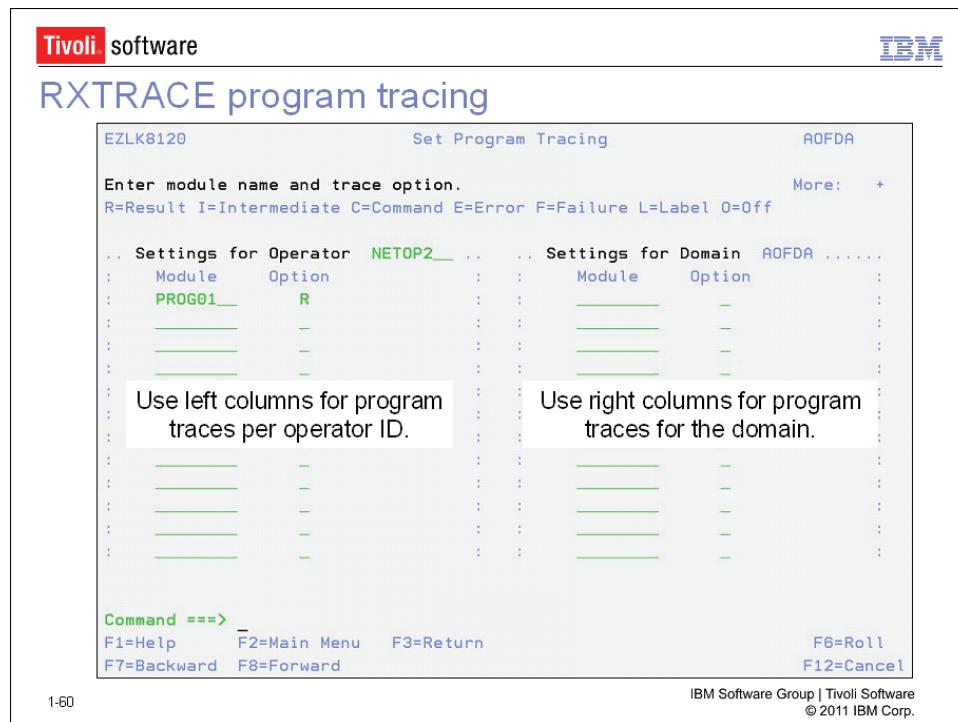
In this slide example, entry/exit tracing is active for the AOFDA domain. NETOP2 is running with the default settings. All supported EXECs create entry and exit trace messages. When entry/exit traces are enabled, EZL260I messages in DSILOG are similar to the following message:

```

EZL260I 09/03/07 17:49:22 NETOP2      EZLE600A ENTRY
EZL260I 09/03/07 17:49:22 NETOP2      EZLE6000 ENTRY
EZL260I 09/03/07 17:49:24 NETOP2      EZLE6000 EXIT      RC=0
EZL260I 09/03/07 17:49:24 NETOP2      EZLE600A EXIT      RC=0
  
```

- The entry trace identifies the command and input parameters, if any are specified.
- The exit trace identifies the return code for the program.

RXTRACE program tracing



You can enable program traces for many NetView EXECs. You can specify the traces for a single task (for example, NETOP2) or for all tasks in the NetView domain (AOFDA). If you need to trace an EXEC on AUTO2, the following steps apply:

1. Tab over to the *Settings for Operator* field.
2. Enter AUTO2.
3. Press the Enter key.

The EZLK8120 panel displays the current settings for AUTO2. You can add your program trace to the list.

Example: Trace i program

Tivoli software
IBM

Example: Trace i program

```

PROG1
 3 *--* P1 = 1
 >L>   "1"
 4 *--* P2 = 5
 >L>   "5"
 5 *--* Call prog2 p1 p2 'This is p3'
 >V>   "1"
 >V>   "5"
 >O>   "1 5"
 >L>   "This is p3"
 >O>   "1 5 This is p3"

Parameter 1 = 1
Parameter 2 = 5
Parameter 3 = THIS IS P3
    >>>   "6"
 6 *--* If Result = (p1 + p2) ←
 >V>   "6" → If result (value of 6)
 >V>   "1" → p1 (value of 1)
 >V>   "5" → p2 (value of 5)
 >O>   "6" → p1 + p2 (value of 6)
 >O>   "1" → Equals (returns 1 indicating the
             expression evaluated true)
 *--* then
 7 *--* Say 'Total =' Result
 >L>   "Total =" →
 >V>   "6" →
 >O>   "Total = 6" →

Total = 6
10 *--* Exit

```

Trace i:

- Evaluates all clauses, terms, and intermediate results before instruction is run
- Is most comprehensive trace

Line evaluates as follows:

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

1-61

Processing return codes

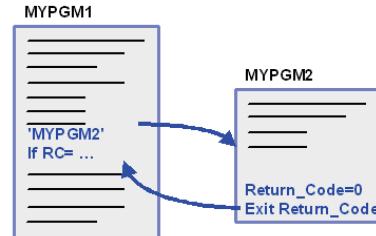
Tivoli software **IBM**

Processing return codes

- REXX sets a reserved variable for the return code, RC, after every instruction, command, or nested EXEC
- Example RC values
 - **0**: No error
 - **-1**: Indication of an error
SIGNAL ON FAILURE
 - **-3**: NetView command authorization error
 - **-5**: EXEC canceled
SIGNAL ON HALT
 - **8**: Syntax error
SIGNAL ON SYNTAX
- Other values are possible and depend on NetView component

Do not use RC as a local variable in your EXECs, as unpredictable results can occur

1-62 IBM Software Group | Tivoli Software
© 2011 IBM Corp.



Important: RC is one of several variables reserved for REXX use. If you set RC, you create a user variable and nullify the use of the reserved REXX variable.

Processing standard errors

Tivoli software

IBM

Processing standard errors

Use REXX **SIGNAL** instruction to handle error conditions

- Interrupts normal flow, passes control to specified label that matches the signal condition
- Does not return control
- Used for testing and emergency actions
 - For example, save local variables or issue a message about the error
- An example is SIGNAL ON FAILURE
- REXX sets a variable, **SIGL**, to the line number that triggered the condition

1-63

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

SIGNAL instructions

Tivoli software 

SIGNAL instructions



- Code SIGNAL instructions at start of EXEC
 - **SIGNAL ON FAILURE:** Driven for conditions that are not normally recoverable, such as a missing command
 - **SIGNAL ON ERROR:** Driven for recoverable conditions or if failure not coded
 - **SIGNAL ON HALT:** Driven when EXEC is canceled
 - **SIGNAL ON SYNTAX:** Driven when a REXX syntax error is detected
 - **SIGNAL ON NOVALUE:** Driven when an unassigned variable is referenced
- Code label in EXEC to process each following condition:
 - Failure:
 - Error:
 - Halt:
 - Syntax:
 - Novalue:

NetView-supplied EXECs do not use SIGNAL ON ERROR

1-64 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

SIGNAL ON HALT should set return code -5 to cancel all calling EXECs.

Example: NOVALUE

Tivoli software

IBM

Example: NOVALUE

```
/*-----*/  
/* COMMAND exec FAILED : LINE Sig1 HAD A VARIABLE WITH NO VALUE */  
/*-----*/  
NOVALUE:  
    Say 'COMMAND' cmd_name 'FAILED : LINE' Sig1 'HAD A VARIABLE WITH NO  
    VALUE: ' Condition('D')  
    Return_Code = 7  
    /* return to caller */  
    Exit return_code
```

Actions to take when a variable novalue condition is detected

- Branch to the NOVALUE: label
- In this case, issue an error message and exit with a return code 7
 - *Cmd_name*: Command that was parsed within the REXX EXEC with a parse source statement
 - *Sig1*: Line number that the novalue condition sets
 - *Condition('D')*: Function that contains the variable name

1-65

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Lesson 6: Additional topics

Tivoli software



Lesson 6: Additional topics

- Sending messages to the z/OS system console
- REXX environments
- Loading EXECs into NetView storage
- Performance
- Security

1-66

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Sending messages to the z/OS system console

Tivoli software

IBM

Sending messages to the z/OS system console

- **WTO** (write to operator): Sends single-line message
- **WTOR** (write to operator with reply): Sends single-line message. EXEC is suspended until response is received
- **DOM** (delete operator message): Removes WTO or WTOR message from one or more z/OS consoles if the message is held and highlighted

1-67

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXX environments

Tivoli software 

REXX environments

- Each time a REXX EXEC runs in NetView, a REXX environment is built to support the EXEC
 - Nested EXECs share the same environment
 - Environment is reusable
- If two EXECs run, two environments are necessary
- By default, NetView retains up to three environments per task
 - Can improve overall performance
 - Freed when you log off
 - Controllable with **DEFAULTS** and **OVERRIDE** commands
 - Number of environments retained
 - Amount of storage used

1-68

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

If you exceed the number of REXX environments, a CNM416I message is displayed as follows:

CNM416I REXX INTERPRETER ENVIRONMENT INITIALIZATION FAILED FOR TASK
task, RETURN CODE = 20, REASON CODE = 24

A reason for a CNM416I message is that the number of entries in IRXANCHR is insufficient for your NetView environment.

Request that your system administrator increase the number of REXX environments that are defined in IRXANCHR. IRXANCHR is in SYS1.SAMPLIB. Refer to the *TSO/E REXX Reference* for an explanation about increasing the number of entries in IRXANCHR. To determine the number of IRXANCHR entries needed, see the *IBM Tivoli NetView for z/OS Tuning Guide*.



Note: Any EXEC running on your system requires REXX environments, including EXECs from other products or TSO/E EXECs you write.

Controlling the REXX environment

Tivoli software

IBM

Controlling the REXX environment

- Use DEFAULTS or OVERRIDE commands to change settings
- LIST DEFAULTS:
 - REXXSTOR: DEFAULT
 - REXXENV: 3
 - REXXSLMT: 250
 - REXXSTRF: DISABLE

1-69

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

REXXSTOR: Specifies the amount of storage, in 1K increments, to be acquired by REXX environment initialization processing. The default: NetView is to specify that TSO/E REXX use the TSO/E REXX default.

REXXENV: Specifies the number of active and inactive, but initialized, REXX environments to be retained for each operator. The default is **3**.

REXXSLMT: Specifies the amount of storage, in 1K increments, that a REXX environment is allowed to accumulate before being stopped after its current use completes. The default value is **250**.

REXXSTRF: Specifies if the NetView operator can run REXX EXECs that use the REXX STORAGE function. By default, this is disabled.

Loading EXECs in NetView storage

Tivoli software

IBM

Loading EXECs in NetView storage

- EXECs is loadable in NetView storage to reduce data set read time
- NetView dynamically loads EXECs in storage
 - MEMSTORE statements in CNMSTYLE
 - LIST MEMSTAT=*: Shows information about members loaded in storage with the MEMSTORE function
 - MEMSTOUT: Controls functions of MEMSTORE
- You can preload EXECs in storage
 - LOADCL: Loads EXEC into NetView virtual storage
 - MAPCL: Displays statistics for EXEC loaded in NetView virtual storage
 - DROPCL: Removes EXEC from NetView virtual storage
 - AUTODROP: Drops less frequently used EXECs that are loaded in storage

1-70

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

Loading EXECs in NetView storage reduces the allotted time for reading the EXEC from a data set. By default, NetView uses MEMSTORE statements in CNMSTYLE to automatically load the most commonly used EXECs (and other data set members) into storage.

You can also use the LOADCL command to load EXECs in NetView storage. You can schedule the AUTODROP command to monitor the use of the EXECs and drop less-often used EXECs.

MemStore function

Tivoli software 

MemStore function

- Default CNMSTYLE statements
 - function.autotask.memStore = auto2
Autotask, where MEMSTORE runs
 - memStore.stgLimit = 5%
Limit total storage used (% only)
 - memStore.minHits = 5
Minimum hits for caching
 - memStore.frequency = 2
How often to check, in minutes
- Schedule a timer (every two minutes) under AUTO2 to load members that have been used more than 5 times.
(Allocate 5 percent of NetView storage maximum.)

1.71

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

By default, NetView loads members that are used more than five times into storage, up to a maximum of five percent of NetView storage. Members can be excluded from being loaded in storage with the memStore.never statement, as the following example shows.

```
memStore.never =
    DSIPARM.DSIOPF
    DSIPARM.DSIOPFU
    DSILIST.*
    *.USERMEM
```

You can disable the memStore function by setting the maximum storage limit to zero percent as follows:

```
memStore.stgLimit = 0%
```

Managing memStore members

Tivoli software **IBM**

Managing memStore members

Use MEMSTOUT to control the members that memStore loads

- Refresh a member
MEMSTOUT REFRESH *ddname.member*
- Unload a member
MEMSTOUT UNLOAD *ddname.member*

1-72 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

You can use the MEMSTOUT command to delete (unload) or refresh members that are controlled with the memStore function. You can also use MEMSTOUT to update the list of members to exclude from memStore.

REFRESH specifies that members cached by MEMSTORE are removed from the cache, but can be cached again after being read from disk. While UNLOAD specifies that members cached by MEMSTORE are removed from the cache and not cached again.



Note: A common occurrence pertaining to memStore occurs when you initially create and test a REXX EXEC. After several tests, changes you make do not seem to run. This condition can occur because the EXEC loads in storage. Use MEMSTOUT to remove the EXEC from storage.

Performance

Tivoli software

IBM

Performance

- Achieve optimum performance when EXEC remains in REXX interpreter
Issuing non-REXX commands suspends REXX interpreter and waits for command to complete
- Use REXX instructions instead of NetView, for example
 - Use REXX PARSE instead of NetView PARSEL2R or MSGID() and MSGVAR(n)
 - Use REXX PARSE UPPER instead of NetView UPPER command
 - Store multiline messages into REXX stem variable instead of accessing with NetView GETMSIZE and GETMLINE commands
- Issue a single GLOBALV to retrieve or store many (task or common) global variables instead of issuing a GLOBALV per variable
- Preload heavily used EXECs into NetView storage or use MEMSTORE function to dynamically load EXECs
- Consider use of NetView PIPEs when possible

1.73

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

This slide provides several methods for improving the overall performance of your REXX EXECs. Use REXX instructions and functions whenever possible. Doing so can keep your EXEC in the REXX interpreter and improve performance.

Consider the use of NetView PIPEs as follows:

- Use PIPE QSAM to read from and write to files instead of using REXX EXECIO.
- Use PIPE NETV or MVS with LOC, TOSTRING, TAKE, DROP, and so on, instead of TRAP, WAIT, and MSGREAD.

In addition, limit the use of the REXX INTERPRET for dynamically building statements.

Security

Tivoli software **IBM**

Security

- Checks EXECs, keywords, and keyword values for proper authorization
Security checks must be explicitly coded
- Requires SAF or NetView command authorization table (CAT) definitions



1-74 IBM Software Group | Tivoli Software
© 2011 IBM Corp.

EXECs, their keywords, and their keyword values can be secured by using two NetView functions:

- `AUTHCHK()`: Checks keywords and keyword values.
- `AUTHCHKX()`: Checks command, keywords, and keyword values.

Documentation is in the *IBM Tivoli NetView for z/OS Programming: REXX and the NetView Command List Language* manual

Your system administrator can create entries in the NetView Command Authorization Table (CAT) or a SAF product to authorize users to the EXEC, keywords, and keyword values. For more information, see the *IBM Tivoli NetView for z/OS Security Reference*.

Summary

Tivoli software

IBM

Summary

Now that you have completed this unit, you can perform the following tasks:

- Explain the basics of REXX EXECs in NetView
- Write NetView REXX EXECs to do the following tasks:
 - Issue commands
 - Trap and parse messages
 - Set and retrieve global variables
 - Trace and automate global variables
 - Perform automation

1-75

IBM Software Group | Tivoli Software
© 2011 IBM Corp.

IBM Tivoli certification and training

In today's global business world, enhancing and maintaining skills is essential to keeping pace with rapidly changing technologies. Businesses need to maximize technology potential and employees need to keep up to date with the latest information. Training and professional certification are two powerful solutions.

Certification

There are many reasons for certification:

- You demonstrate value to your customer through increased overall performance with shorter time cycles to deliver applications.
- Technical certifications assist technical professionals to obtain more visibility to potential customers.
- You differentiate your skills and knowledge from other professionals and stand out as the committed technical professional in today's competitive global world.

Online certification paths are available to guide you through the process for achieving certification in many IBM Tivoli areas. See ibm.com/tivoli/education for more information.

Special offer for having taken this course

Now through 31 December 2011: For having completed this course, you are entitled to a 15% discount on your next examination at any Thomson Prometric testing center worldwide. Use this special promotion code when registering online or by telephone to receive the discount: **15CSWR**. (This offer might be withdrawn. Check with the testing center as described later in this section.)

Role-based certification

All IBM certifications are based on job roles. They focus on a job a person must do with a product, not just the product's features and functions. Tivoli Professional Certification uses the following job roles used:

- IBM Certified Advanced Deployment Professional
- IBM Certified Deployment Professional
- IBM Certified Administrator
- IBM Certified Solution Advisor
- IBM Certified Specialist
- IBM Certified Operator

Training

A broad spectrum of courses, delivery options, and tools helps keep your employees up to date with the latest IBM Tivoli information:

- *Instructor-led training (ILT)*
Live interaction with an IBM instructor, hands-on lab exercises, and networking with your peers from other companies
ibm.com/tivoli/education
- *Instructor-led online (ILO)*
All the benefits of ILT, but savings on travel dollars and training costs
ibm.com/training/ilo
- *Self-paced virtual classes (SPVC)*
Interactive and hands-on exercises on your schedule
ibm.com/training/us/spvc
- *Web-based training (WBT)*
Training anywhere, any time, that saves you money and travel
ibm.com/training/us/tivoli/wbt
- *Multimedia library*
Modules supporting new and experienced learners with fully animated multimedia clips, step-by-step audio, and companion text
ibm.com/software/tivoli/education/multimedialibrary
- *IBM Education Assistant*
More specific, granular web-based training with individual presentations on specific topics
www-01.ibm.com/software/info/education/assistant/
- *Corporate Education Licensing Program (CELP)*
Solutions for large IBM customers who need to adopt IBM Tivoli's tools and technologies
ibm.com/training/us/tivoli/celp
- *Tivoli training paths*
Course maps with flow charts and course descriptions to help you find the right course
[ibm.com/training/us/tivoli\(paths](http://ibm.com/training/us/tivoli(paths)



Printed in Ireland