Using System REXX to send z/OS Command Responses to the IBM Tivoli Enterprise Portal

Mike Bonett, IBM Advanced Technical Skills
bonett@us.ibm.com

System REXX, provided in z/OS® 1.8 and higher, allows monitoring and automation products such as IBM® Tivoli® Monitoring (ITM) to directly invoke REXX programs from their automation interfaces. From the IBM Tivoli Enterprise Portal (TEP) it is possible to invoke a z/OS command, including a System REXX procedure. A limitation of this function is that the command response does not come back to the TEP, just an indicator if the command execution attempt was successful or not. However, using System REXX, along with the IBM Tivoli Universal Agent (which is part of IBM Tivoli Monitoring), command responses can be returned to the TEP, and treated as other monitoring metrics in the TEP – for example, the contexts of the response can be used to trigger situations and automation policies.

To return the command responses to the TEP two functions must be implemented:

1) The Tivoli Universal Agent must be installed and its SOCKET Data Provider activated, with customization (called a metafile) for receiving the command response data.

2) A REXX program must be available under System REXX that can execute the program, capture the command response, and return that response via TCP/IP to the Universal Agent.

This Technote will provide examples of implementing these two functions, and an scenario showing their use.

**Universal Agent customization**

Usage of the Universal Agent is documented in the *IBM Tivoli Monitoring Universal Agent Users Guide*. The Universal Agent can retrieve or receive data from many sources, via built-in “data providers”.

The agent does not execute on z/OS. The two easiest methods to send z/OS data to the agent it are (1) via TCP/IP, or (2) via a Unix System Services (USS) log file that is accessible, via NFS or SMB sharing, from the Universal Agent platform. This particular implementation will use TCP/IP, which requires the Universal Agent SOCKET data provider to receive the data.
To enable the received data to be displayed in the TEP a metafile definition is required. The metafile defines how the data will be received, and the format of the data. For z/OS messages a simple metafile can be used, such as the following one:

```plaintext
//APPL ZMGREXX
//NAME MSG01 E 7200
//ATTRIBUTES ';
RDATE D 8
RTIME D 6
RGROUP D 8
RLINE D 3
MSGTEXT D 120
```

The metafile contents define the following:

- The //APPL statement defines the name that will be used for the message data. The first three letters of the name must be unique, as this will be used to generate various files behind the scenes used for real time and historical data capture.
- The //NAME statement names a group of attributes to be associated with the application name. A metafile can have multiple sets of attributes, but for this implementation only one is needed. The parameters on the NAME statement are important:
  - The E indicates that the incoming data is in EBCDIC format.
  - The 7200 indicates that the data source will be kept online for 7200 seconds (2 hours) after the last receipt of data. For example, if a command response is returned at 11AM, if no additional command responses are received the data source will go offline in the TEP at 1 PM.
- The //ATTRIBUTES statement starts the definition of the data layout. The “;” indicates that the incoming attributes will be separated by a semi-colon.
- Each line after the //ATTRIBUTES statement defines an attribute, type, and size. For this implementation all of the attributes will be in character format:
  - RDATE is the date in YYYYMMDD format, a length of 8
  - RTIME is the time in HHMMSS format, a length of 6
  - RGROUP is a character string of length 8 that will be used to identify a group of command response lines.
  - RLINE is a character string of length 3 that will represent the line number of each line returned in a command response.
  - MSGTEXT is the text for a command response line, and will be up to 120 characters long.

The above metafile definitions are placed in a plain text file and the contents are validated and activated by using the KUMPCON command (fully documented in the Users Guide). Once this is done, the metafile information will appear in the TEP navigation tree.
With no active data it will be offline (grayed out). The application name will be used in the child branch of the Universal Agent (along with the name of the system from which the command response data is received), with the attribute group workspace (MSG01) a branch under it.

System REXX customization

For System REXX, a REXX program that receives the command to run, executes the command, captures the response, and formats and returns the response to the Universal Agent is needed. The following REXX program performs those functions:
001 /*** SYSTEM REXX - ISSUE COMMAND AND SEND RESULTS TO
002 UNIVERSAL AGENT
003 - input parameters:
004 - groupid: 1-6 characters to
005 identify response
006 - input: command string to execute
007 Author: Mike Bonett, IBM Corporation, copyright 2010
008 SAMPLE CODE - no warranties intended, implied, or assumed
009 /***/
010 arg groupid input
011 /*** Define Universal Agent connection information ***/
012 uahost=10.1.1.37
013 uaport=7500
014 /*** Execute the z/OS command and capture the command response ***/
015 rc = AXRCMD(input,CMDRESP.,10)
016 /*** The command may return multiple WTOs - check for them ***/
017 do while rc=0
018 rc=AXRCMD(CMDRESP2.,5)
019 if rc=0
020 then do
021 j=cmdresp.0 + 1
022 cmdresp.0=cmdresp.0+cmdresp2.0
023 do k=j to cmdresp.0
024 zidx=k-(j-1)
025 cmdresp.k=cmdresp2.zidx
026 end
027 end
028 end
029 /*** Initialize the socket environment ***/
030 z=Socket("Terminate","mwbskt")
031 y=Socket("Initialize","mwbskt")
032 /*** Create local socket and get socket endpoint id ***/
033 sockrc =Socket('Socket',2,'SOCK_STREAM','IPPROTO_TCP')
034 /*** Bind socket to network interface and connect to UA ***/
035 sockid = word(sockrc,2)
036 sock_network_name = 'AF_INET' uaport uahost
037 sockconnrc = Socket('Connect',sockid,sock_network_name)
038 /*** Send initial record to identify metafile to be used ***/
039 targetapp = "\ZMGREXX"
040 Writerc= Socket('Write',sockid,targetapp)
041 /*** Send the command response data ***/
042 rdate=date('S')
043 crlf="0D"X
044 rtime=substr(time(),1,2)||substr(time(),4,2)||substr(time(),7,2)
045 do i = 1 to cmdresp.0
046 lct=translate(format(i,3),'0',' ')
The REXX program takes the following actions:

- Line 10 defines the program input parameters
  - **groupid** – a string that will be used to identify the group the response lines are part of
  - **input** – the command to be executed
- Lines 11-13 define the hostname/ip address where the Universal Agent is running, and the IP port the SOCKET data provider is listening on. While this implementation provides the information as part of the REXX program, more flexible alternatives exits, such as passing this data as a parameter, or having the program obtain the information from another source (e.g. reading a file or database).
- Lines 14-15 execute the command using the System REXX AXRCMD function, which places the command response lines into a stem variable.
- Lines 16-28 are needed for those commands which return multiple lines as separate Write to Operator (WTO) messages, instead of a single multiple line WTO message. The AXRCMD, when first invoked, only returns the first WTO (single line or multiple line) response from the command. However, calling the AXRCMD function again, with an empty command parameter, will retrieve the next WTO, or return a non-zero return code if there are no more WTOs. These lines call AXRCMD repeatedly to retrieve all of the WTOs, and add them onto the command response stem variable.
- Lines 29-31 activate the REXX socket functions.
- Lines 32-33 create a local socket.
- Lines 34-37 bind to the local socket and create a network connection to the Universal Agent host and SOCKET data provider port.
- Lines 38-40 send an initial record to the SOCKET data provider. This initial record identifies the metafile whose attributes the data that follows will match.
- Lines 41-49 format the data as expected by the metafile attributes and send it:
  - The date, time, group ID, and line number are prefixed to the command response; all items are delineated by a semi-colon.
  - A carriage return (hex 0D) is appended, so that the SOCKET data provider recognizes the end of a record
- Lines 50-52 close the connection and exit the program.
The REX program is stored in the System REXX datasets (SYS1.SAXREXEC by default; additional datasets can be defined in the AXR00 member of SYS1.PARMLIB). Once there it is ready for use.

Example Scenario

An IBM Tivoli Monitoring “Take Action” command will be defined to send a z/OS console command to the REXX program, which will return the results to the Universal Agent and then displayed in the TEP.

The “Take Action” command is defined as follows:

The action is named **runzOScommand**. When invoked, it will prompt for three parameters:

- SystemREXXID – the command prefix for System REXX on the target system
- GroupID – A string (any desired) to identify the command response group
- Command – the command to be executed

**MSG2UA** is the name of the REXX program that will be run.

The following pictures show the Take Action being invoked to run the $D SPOOL command on a z/OS system:
The result is immediately seen on the MSG01 workspace:

![Navigator](image)

![Physical](image)

A version of MSG2UA was run that displayed the progress through the program, which can be seen on the z/OS Console:

```
REXXCB MSG2UA RG1 $D SPOOL
$D SPOOL
$HASP893 VOLUME(IAM200) STATUS=ACTIVE,PERCENT=97
$HASP893 VOLUME(IAM201) STATUS=ACTIVE,PERCENT=55
$HASP646 60.8782 PERCENT SPOOL UTILIZATION
AXR0500I AXREXX OUTPUT DISPLAY 711
EXECNAME=MSG2UA REQTOKEN=0000400000000000C6409BAD3E3D4452
First terminate call: 2005 ESUBTASKNOTACTIVE Subtask not active
First Initialize call: 0 mwbskt 40 TCPIP
REXX SOCKET CLIENT GOING TO 10.1.1.37 USING PORT 7500
SOCKET create result: 0 1
Connection Result: 0
Writing explicit specification record
Explication Specification Write Result: 0 9
Writing data record 1
Write result 1 : 0 75
Writing data record 2
Write result 2 : 0 75
Writing data record 3
Write result 3 : 0 68
0 mwbskt
```

Now that the command response is in the TEP, functions since as defining situations and policies can be applied to it. For example, a situation can be triggered to carry out an action, or to cancel out another situation, based on the contents of a command response.
The “Take Action” is just one method of using this integration. Other options include:

- Using a policy to issue the command based on one or more situation states, and then using the response to determine further actions.
- Having a task on z/OS periodically run the REXX program and populate the TEP with data from specific commands.

As with any control function, planning for security to use this function is important. Assigning which TEP users can issue commands and which commands they can issue requires a coordinated effort between TEP and z/OS security definitions.

Summary

This sample implementation illustrates the added value of using System REXX and IBM Tivoli Monitoring together to provide a solution. Capturing command responses from z/OS enhances the ability to manage the z/OS environment from the TEP, and can provide additional efficiency and flexibility from integrating this information with other monitored data in the TEP.

The following items provide additional information and background on the topics discussed in this paper:

- **z/OS MVS Programming Authorized Assembler Services Guide** (SA-22-7608) – section on System REXX planning and usage. This document is available in the online z/OS Information Center for the respective z/OS release.
- **IBM Tivoli Monitoring Universal Agent Users Guide** (SC32-9459). This document is available online in IBM Tivoli Monitoring Information Center for the respective ITM release.
- **Invoking System REXX Programs from IBM Tivoli Monitoring Command and Automation Functions**
  
- **Using the IBM Tivoli Universal Agent to Enhance z/OS Monitoring**
  