Tips for Developing REXX Programs on z/OS

Pedro Vera (pedro@us.ibm.com)

August 30, 2016
I will talk about various topics…

For some, there is not a good transition between the topics, for which I apologize
Hello, my name is Pedro Vera
I am a long time IBMer, with about 30 years of working with TSO REXX.

I worked for 15 years in the IMS development group.
After creating the REXX SPOC API for IMS and getting great customer feedback, I was able to convince my IBM colleagues to provide additional rexx support, including:
1. System REXX,
2. the SDSF REXX API,
3. the REXX interface for RACF,
4. inline REXX for ISPF panels,
5. inline REXX for ISPF skeletons,
6. Trace highlighting

Currently, I am working as a developer for a product called DB2 Administration Tool, which uses rexx heavily.
**DB2 Administration Tool**

**IBM DB2 Administration Tool for z/OS, V11.2 (5655-DAT)** allows you to manage the tasks that keep DB2 for z/OS performing at peak levels. It provides a comprehensive set of functions that helps your DB2 for z/OS personnel manage their DB2 for z/OS environments efficiently and effectively, including:

Providing in-depth catalog navigation by:
- Displaying and interpreting objects in the DB2 for z/OS catalog
- Executing dynamic SQL statements

Integrating with other DB2 Tools to simplify the task process flow:
- Table editing
- SQL cost analysis

Adding summary reports to provide an overview of complex data changes and migrations

---

I am currently a developer in the DB2 Administration Tool. I will give a quick demo and then go over some of the unseen changes that might be of interest to you.
Let me switch to a demo of the DB2 Administration Tool.

This is one of the primary panels for DB2 Administration Tool. It shows the DB2 objects supported by DB2 Admin.

In the name field, type the name of the object.

When you press the Enter key, DB2 Admin generates an SQL statement that searches the DB2 catalog using an SQL LIKE operator to qualify the search for the search criteria, Then displays the results in the subsequent panel.

<< USE D option on this panel >>
Demo of DB2 Admin Tool

List of Tablespaces

You can navigate the catalog, which contains information about various DB2 objects. If you enter the S line command in the Select field next to database name in the panel.

One example is the DIS command.

Another example is the UTIL command. It will display a list of DB2 utilities. From the list, you can select the utility and it will build the JCL required to run the utility.
Demo of DB2 Admin Tool

Utility Options

This example is the REORG utility.

It will display a panel with the utility options.
Demo of DB2 Admin Tool

Produces JCL to execute the utility

```
R00018 // STEP BEGIN: REGC, TABLESPACE PSVDB.PSVTSEM2
R00019 //************************************************************************
R00020 //BEGIN EXEC DSNUPROC, SYSTEM=PSNDB,
R00021 //
R00022 // UID=''
R00023 //DSNUPROC.SYSFUNC DD DSN=PSNDB,DSNB,CNTL=PSVDB.PSVTSEM2,
R00024 // DISP=(MOD,CAILG),
R00025 // SPACE=(IRK,(5,5),RLSE),
R00026 // UNIT=SYSDA
R00027 //DSNUPROC.SYSPROC DD DSN=PSNDB,DSNB,UNLD=PSVDB.PSVTSEM2,
R00028 // DISP=(MOD,CAILG),
R00029 // SPACE=(CYL,(5,5),RLSE),
R00030 // UNIT=SYSDA
R00031 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
R00032 // 8 Line(s) not Displayed
R00034 //DSNUPROC.SYSIN DD *
R00040 //TEMPLATE PSVCPY1
R00041 //DSN
R00042 'PSNDB.SSISD(3,2).DBB.BSN..8IC..8LB.8PB..DBYE(3).8BD.'
R00043 //UNIT SYSDA
R00044 //TEMPLATE PSVDISC
R00045 //DSN
R00046 'RUS.FRIB..DISC..RSU0.
R00047 //UNIT SYSDA
R00048 //REGC TABLESPACE PSVDB.PSVTSEM2
R00049 //PARALLEL (GO)
R00050 //LOG NO
R00051 //SORTDATA
R00052 //SHLEVEL REFERENCE
R00053 //COPYDDN(PSVCPY1)
R00054 //DISCARDON(PSVDISC)
R00055 //************************************************************************
```

When you press the Enter key, it will produces the JCL to execute the utility.
I joined the DB2 Admin Tool team several years ago. It is a fairly mature product. It is 15 years old. The component I work on is somewhat tricky. I want to talk about some changes that I have made to Admin Tool. These are unseen changes…

Much of the work is done by rexx programs, but it is somewhat difficult to trace a program because of the environment and because of the size of the program. There are 350 members in the rexx data set. Some programs are thousands of lines long.

The user can select a long list of objects… the programs will execute numerous times.

One of the problems I have is that Line mode output scrolls off of the screen.

My goals are to be able to debug a program and allow customers to provide a trace when it is needed.
Here is a typical program flow.

We have a main program that we call the ‘driver’. It will call ISPF panels, rexx programs, other PLI programs, as well as DB2 itself.

I mention the ISPF panels and the ISPF skeletons because they can invoke Rexx programs themselves, as well as have inline rexx.
Editor Basics

Use HI REXX and HI PAREN commands.

The editor on z/OS is the ISPF editor. It supports a HILITE command.

REXX is one of the languages that it supports.

I use it with HI AUTO and HI PAREN. You type those commands in the primary command line.
Admin Tool Start Up

Through LIBDEF and ALTLIB

address ISPEXEC
"LIBDEF ISPLLIB DATASET ID('DB2.DSNLOAD'","'ADB.SADBLLIB')"
"LIBDEF ISPMPLIB DATASET ID('ADB.SADBMLIB')"
"LIBDEF ISPPLIB DATASET ID('ADB.SADBPPLIB')"
"LIBDEF ISPTPLIB DATASET ID('ADB.SADBTPPLIB')"
Address TSO "ALTLIB ACTIVATE APPLIC(EXEC)"
"DSN('DB2.DSNEXEC','ADB.SADSEXEC') QUIET"
"SELECT CMD(%ADBDMT) NEWAPPL(ADB) PASSLIB"
Address TSO "ALTLIB RESET"
"LIBDEF ISPLLIB"
"LIBDEF ISPMPLIB"
"LIBDEF ISPPLIB"
"LIBDEF ISPTPLIB"

The setup for Admin Tool is somewhat like this
(it somewhat like this, but our setup clist is more elaborate):
There is a LIBDEF statement for the ISPF data sets

There is an ALTLIB statement for the exec library.

There is also an ALTLIB statement for the clist library, but
It is not shown here.

ALTLIB lets the TSO user define optional, user-level or application-level
sets of libraries containing rexx execs or clists. These libraries
are searched before the system-level rexx execs and clists.
PLI Calling REXX

PLI calls REXX through an ISPF interface.

```
Buffer = 'SELECT CMD(%myrexx p1)';
Call ISPEXEC(length(buffer), buffer);
```

Our PLI driver program calls REXX programs through an ISPF interface.

That is the way it was written when I got here; so far there is no reason to change that.

Another choice available is to use IRXEXCOM service program.
Serviceability

DB2 Admin Tool supports a debug command:

```
DEBUG ON
DEBUG OFF
```

They set a global variable that is examined by various programs.

If you report a problem to IBM, they may ask you for various traces.

DB2 Admin Tools supports a “debug” command. Actually, there is so much of the trace that it is difficult for customers to capture.

The information in the trace is only useful to IBM.
REXX Trace

A REXX program can use the TRACE('R') command to produce trace records.

```
0145 parse arg parm
0146 /* Debug option */
0147 parse var parm left "DEBUG(" debug ")" right
0148 If debug="DEBUG" Then
0149 Do
0150  Trace('R')
0151 End

0158 Parse Upper Source . . Sysicmd .
0159 If (debug="DEBUG") Then
0160 Do
0161  logtxt = time('L') 'Start %sysicmd parm'"MSG(ADB7811)"
0162 End
```

Tracing records information about a program's execution. This information is typically used by programmers for debugging purposes in order to diagnose problems with software.

In this example, DEBUG is provided as a program parameter. The caller knows the setting and passes it in.

I want to point out the PARSE SOURCE statement on line 158. There are several rexx programs being called. PARSE SOURCE includes the name of the program in the trace.

Also, I use the LOG service of ISPF to add START and END Records in the ISPF log file.
In addition to the REXX language instructions and built-in functions, and the TSO external functions and REXX commands that can be used to writing REXX execs, TSO provides programming services for REXX processing. Some programming services are routines that let you interface with REXX and the language processor.

See  TSO REXX Reference SA32-0972, and chapter 12, “Programming Services”
Currently, PLI driver program produces output through the use of the PUT statement.
• The output goes to the TSO screen, but it is discarded when it rolls off of the screen
• It cannot be captured by the REXX OUTTRAP function
IRXSAY from PLI

A PLI program can use the IRXSAY programming service.

The output will be treated the same as from the REXX SAY output.

```
/* IRXSAY from PLI */

PROC OPTIONS(MAIN) REORDER;

Dcl saylen Bin Fixed(31) Init(length(saymsg));
Dcl saymsg Char(25);
Dcl IRXSAY External
    Entry (Char(8),
        Ptr,
        Bin Fixed(31))
        Options(Asm Retcode Fetchable);

saymsg = 'Hello World!';
Call IRXSAY('WRITE ', addr(saymsg), saylen );

END PLisay2;
```

In order to capture the PLI output, the PLI program should use the IRXSAY programming service module instead of the PUT statement.

IRXSAY lets you write a character string to the same output stream as the REXX SAY instruction.

You can declare it as fetchable and then the PLI runtime will only load the module when it called. This uses assembler parameter conventions: The first parameter is WRITE or WRITEERR
The second parameter is the address of the text and third, is the length of the text.

Sorry, this example does not check the return code.
I want to show you a demo of the DEBUG ON trace.

You issue the DEBUG ON command in the primary command line

Then the results of the TRACE('R') instruction are displayed On the screen. The output fills up the screen from top to Bottom, then the older lines are discarded as they scroll off The screen.
Session Manager

Normally, the 24 line screen fills up and then older information is discarded.

With the Session Manager, TSO keeps a complete journal of everything that happens during your session.

Normally, the older lines are discarded as the scroll off the screen.

With the Session Manager, TSO keeps a complete journal of everything that happens during your session while you are in line mode TSO. It records everything you type in and everything the system displays. Any time during your terminal session, you can look at work you did during the session. That is what the manual says anyway.

The session journal can be compared to a long sheet of paper that fills up with information as you do work.
Session Manager Customization

The Session Manager stream wraps at around 3000 lines. It supports a table that specifies the max size of the stream.

The default for the Session Manager stream is for it to wrap at around 3000 lines. When you get to the end, the first lines of the trace get discarded.

Luckily, the Session Manager supports a configuration table that specifies the limits.

My recollection is that this zap allows about 75K of lines.
Session Manager Logon Proc

Put the modified ADFMDFLT module in an APF data set and then use it in a STEPLIB.

```
//TSOUSER EXEC PGM=ADFMDFT03 DYNAMNBR=256,REGION=1024K,TIME=1440,
// PARM='SM(IKJETF01,Y)EXEC ''STL.LOCAL.CLIST(DEFAULT)''
//STEPLIB DD DISP=SHR, DSN=E11.LOGON_LOAD
//SYSIN   DD TERM=IS
//SYSPRINT DD TERM=IS
//SYSPROC  DD DSN=SIL.LOCAL.CLIST,DISP=SHR
//       DD DSN=SPF.PRODUCT.CLIST,DISP=SHR
//       DD DSN=SPF.PP.CLIST,DISP=SHR
```

Put the modified ADFMDFLT module in an APF data set and then specify the data set name in a STEPLIB statement.

Use that logon proc when you logon.

Setting up the Session Manager is discussed in the TSO Customization manual.

The program name for the Session Manager is different than the normal TSO program (IKJETF01).
Full Screen Mode vs Session Manager Mode

The Session Manager captures line mode messages…
Except when it is in Full Screen Mode!

The TSO terminal can operate in two modes: line mode and full screen mode. When it is operating in full screen mode, the lines are not captured.

The display of an ISPF panel puts it into full screen mode. After the display of a panel, you need to get it back in the Session Manager mode.

1. You have to change your setting to use Session Manager mode
2. After each display, issue CONTROL DISPLAY SM
Demo of Session Manager stream

Use exec to save the stream to a data set.

The session manager lets you capture the trace information, And it lets you scroll up and down, but In reality, it is not as Useful as it could be. For that reason,

I will give you a demo of saving the Session Manager stream to a data set.
This exec uses Session Manager commands to
1. COPY the stream to a TSO data sets
2. Clear the stream for subsequent use.

It also displays the file using the VIEW utility.

I use this very frequently.
This is an editor macro that is associated with the Session manager copy exec. Mainly, I like to preserve the highlighting of the exec; the trace may break up the trace lines at undesirable locations.
Trapping Command Output

Use the OUTTRAP function to Start
then to turn off

```plaintext
000005 parse arg command ; if command = '' then command = "time"
000006 x = outtrap('output. ')
000008 Address TSO command
000009 x = outtrap('OFF')
```

In this example, a parameter is passed in. It is a TSO command to be issued.

Through the use of the OUTTRAP function, the response from the command is captured in the stem variable.

Afterwards, the trapping is ended.
Trapping Within DB2 Admin

Inserts a trap function between calls to rexx programs.

It is either blank or an exec name.

It traps any REXX trace information produced and saves the trace to the ISPF LIST file.

This example inserts a trap function between calls to rexx programs.

-(in lines 412-414) it is either blank or an exec name.

-Depending on whether a trace is desired.

It traps any REXX trace information produced and saves the trace to the ISPF LIST file. Our customers are more used to sending the ISPF LIST file.

But I cannot use this technique throughout because the calls to DB2 Interfere with the underlying task structure. There is something
Like a fence between some of the calls…
Saving to the LIST file

Use LIST service to save lines of trace.

```hll
0063    /* Process all of the captured output from the secondary program
0064    */
0065    Do z = 1 to ADBETRAP.0
0066    /* Send a line of output to the ISPF LIST file.
0068    */
0069    logtext = strip(ADBETrap.z):
0070    Parse var logtext logtext120 logtext2
0071    "LIST BUFNAME(logtext) LINELEN(121)"
0072    If rc > 0 Then
```

After the trace is captured to the stem variable, it is saved to The ISPF LIST file.

I used the LIST service to copy individual trace lines to the ISPF LIST data set (see line 71)
ISPF panels allow inline rexx statements
• Enclose in *REXX and *ENDREXX statements
• Provide the names of variables that are to be used by REXX.

The *REXX statement is used to invoke REXX code in a panel's )INIT section, )REINIT section, or the )PROC section. The REXX can be coded within the panel source immediately after the *REXX statement, or the name of a member containing a REXX program can be supplied.
ISPF panel trace

Start trace with ISPDPTRC command.
Display trace with ISPDPTRC command.

Use U.V with TORBA value filled in.

ISPF provides a way to trace its panels.

Start the panel trace with the ISPDPTRC command in the primary Command line. You can display trace file by issuing the ISPDPTRC Command a second time.
ISPF Skeleton Rexx

ISPF skeletons allow inline rexx statements
- Enclose in )REXX and )ENDREXX statements
- Provide the names of variables that are to be used.

The file tailoring skeleton language is somewhat primitive. It can be extended by adding rexx statements.

ISPF skeletons allow inline rexx statements
- Enclose the program in )REXX and )ENDREXX statements
- In then )REXX statement, provide the names of variables that are to be used.

The )REXX control statement is used to invoke REXX code from within a skeleton. The REXX can be coded within the skeleton immediately after the )REXX control statement, or the name of a member containing a REXX exec can be supplied.
ISPF Skeleton Trace

Start trace with ISPFTTRC command.
Display trace with ISPFTTRC command.

Start the file tailoring trace with the ISPFTTRC command in the primary Command line. You can display trace file by issuing the ISPFTTRC Command a second time.
Thank You