AUGUST
2009

PHOENIX SOFTWARE INTERNATIONAL®

Bulletin

IN THIS ISSUE

Upcoming Events

SHARE
August 23–28, 2009
Denver, Colorado

IBM System z Expo
October 5–9, 2009
Orlando, Florida

Current Product Releases

CONDOR® z/OS 18.2

CONDOR® z/VSE 24.2

(E)JES® V4R6.0

Entrypoint® Plus 3.0.18

FALCON® z/OS 18.2

FALCON® z/VSE 24.2

Falcon32® V8R1

Key/101® 8.0

PHX-Adders® 07.0

PHX-KeyPlus® 05.0

PHX-ODE® 07.0

Harness the Power of (E)JES® from REXX Programs

By Ed Jaffe

Ed Jaffe, Chief Technology Officer at Phoenix Software and head of the (E)JES team, has been involved in the mainframe community through GUIDE and SHARE since the early 1990s. He now serves as SHARE MVS Core Technologies Project Manager and is presenting five sessions at SHARE in Denver next month. He is a three-time recipient of a SHARE Best Session Award.

The Restructured Extended Executor (REXX) language originated on the VM platform, but has been an integral part of z/OS® for decades. REXX is a procedural, structured programming language—generally interpreted rather than compiled—with interfaces to z/OS components such as CONSOLE, ISPF, z/OS UNIX®, TCP/IP, and others.

With (E)JES V4R5 and higher, you can harness the versatility of REXX to interface and interact with (E)JES. This support provides a REXX program with nearly all of the capabilities available to an interactive (E)JES user.

(E)JES API functions are provided to your REXX program through the EJESREXX environment.

The environment is initially established using the **`ejesrexx()`** function. Once established, the environment will accept additional requests via the **`ejesrexx()`** function or **`EJES`** host command environment. Your program sends (E)JES commands to the environment, using the **`EXECAPI`** command, and it passes back the results via REXX variables. Your program interrogates those variables and optionally issues additional commands. When finished, your program terminates the EJESREXX environment. Terminating the environment does not erase any REXX variables set during the course of execution.

```
01 /* REXX to clean-up a user's jobs */
02 /*                               */
03 /* Input parm is userid. Blanks   */
04 /* defaults to current userid.    */
05 ClupJobs:
06   parse arg userid
07   if LENGTH(userid)=0 then userid=USERID()
08   delstack
09   queue "SRESET;OWNER" userid";ST OJOB"
10   queue "MASKCHAR *%;FILTER MAXC = 'CC 0000';UPDATE"
11   queue ""
12   rc = ejesrexx("execapi 0 (prefix ST_")
13   if rc>0 then do i=1 to ST_Msg.0;say ST_Msg.i;end
14   else do
15     say "Deleting" ST_Lines "jobs owned by" userid "..."
16     rc = ejesrexx("execapi 0 'SELECT * C' (prefix CMD_")
17     if rc>0 then do i=1 to CMD_Msg.0;say CMD_Msg.i;end
18     rc = ejesrexx("termapi")
19   end
20   exit
```

A Couple of Sample EJESREXX Programs

Let's take a look at a couple of sample programs to show the basics of how EJESREXX works.

CLUPJOBS

This simple but powerful program deletes from the system all jobs owned by a given userid that completed with return code 0000.

- This article assumes you already know how to program in REXX. Rather than cover what every line does, I'll point out what this program is doing as it pertains to EJESREXX.

Lines 06 and 07:

- The variable **`userid`** is set to either the parameter passed when invoking the REXX program or the current user-id if no parameter is passed.

Lines 08 through 11:

- The program queues some (E)JES commands onto the data stack, followed by a null stack entry (to signify no more commands).
- The first group of commands issued are **`SRESET`** (reset all primary selection filters), **`OWNER`** (set the owner primary selection filter), and **`ST OJOB`** (navigate to STATUS display showing only jobs).
- The next group of commands are **`MASKCHAR`** (set mask characters), **`FILTER`** (to show only rows where the **`MaxComp`** column contains **`CC 0000`**), and **`UPDATE`** (to re-build the display with fresh information and apply the filters).
- The command strings work exactly as if entered in the interactive environment. The core (E)JES code has no idea your REXX program is controlling it.
- After executing these commands, the STATUS display should show only those jobs owned by **`userid`** that completed with return code zero.

Line 12:

- The program tells EJESREXX to execute the commands on the data stack using **`EXECAPI`**. The first invocation automatically initializes the EJESREXX host command environment.
- EJESREXX pulls an entry from the data stack and runs it as if entered from the (E)JES command line. If an error message is produced, processing stops, a non-zero return code is set, and the stack is not cleared (so you can interrogate the remaining entries if necessary). Otherwise, the next stack entry is processed. When a null entry is detected, processing returns to the REXX with return code zero.
- **`prefix ST_`** tells EJESREXX to use **`ST_`** as the prefix for the REXX variables it creates. If you don't specify a variable prefix, the default value of **`EJES_`** is used. Variable prefixes help keep things straight when navigating different displays.

Line 13:

- If the return code is greater than zero, something went wrong. The program then displays the contents of the stem **`ST_Msg`**, which contains messages. **`ST_Msg.0`** contains the number of elements in the stem.

Line 15:

- Here we write a message indicating the number of jobs to be deleted from spool. The variable **`ST_Lines`** contains this value. It's the number of lines/rows on the STATUS display.

Line 16:

- Rather than queuing (E)JES commands to the data stack, this alternate form of **`EXECAPI`** issues the **`SELECT * C`** command directly.
- **`SELECT`** issues a line command against all rows that match the specified mask. An asterisk specifies all rows. Therefore, the **`C`** line command (cancel) is issued against every row on the display.
- Though not necessary, a different variable prefix (**`CMD_`**) was chosen.

Line 18:

- The EJESREXX host command environment is terminated.

This sample program could prove useful for cleaning up your work at the end of the day. And, because it supports a parameter for the userid, it can be used to clean up other people's work (assuming you're authorized to delete their jobs).

SRCHJOBS

Here's a slightly more complex REXX program that searches through a user's jobs looking for a given search string. The names and ids of jobs containing the string are displayed in a little report.

Many of the techniques are similar to the previous example. Here are some notable differences:

Lines 06 through 08:

- Two parameters are required: **`userid`** and **`string`**. If **`userid`** is only one character long, it is replaced by the current userid. **`string`** is a required parameter. If not supplied, the REXX program terminates.

Line 09:

- A variable called **`jobix`** is initialized to zero here. This variable is used to create the stems to hold job name and id for the subset of jobs found to contain the string.

Line 13:

- The simplest form of **`EXECAPI`** is used. Commands come from the data stack. No variable prefix is specified.

Line 17:

- The **`SELECT`** command issues the **`B`** (browse) line command against all rows of the display. The program uses **`address ejes`** to demonstrate how that syntax looks. When you invoke **`EXECAPI`** this way, the REXX variable **`RC`** contains the return code.

Lines 20 and 31:

- This do loop processes every selected browser. It keeps looping until the function type (**`EJES_FunType`**) is no longer **`BROWSER`**. That will happen when processing eventually returns to the STATUS display.

Lines 21 through 23:

- These three statements issue the (E)JES **`FIND`** command.

Line 24:

- The stack is cleared if an error occurred i.e., the string is not found.

Lines 26 through 28:

- The variable **`jobix`** is incremented. Then **`jobname.jobix`** is set to the current job name from **`EJES_JobName`** and **`jobid.jobix`** is set to the current job id from **`EJES_JobId`**. The variables **`EJES_JobName`** and **`EJES_JobId`** (and others) are set by EJESREXX so they're available when you need them.

Line 30:

- **`EXECAPI`** is used to issue the (E)JES **`END`** command. This terminates the current job browser. If there are more jobs to process, the next browser is started

```
01 /* REXX to search a user's jobs   */
02 /*                               */
03 /* Input parms are userid and    */
04 /* the string to search.         */
05 Search:
06   parse arg userid string
07   if LENGTH(userid)<=1 then userid=USERID()
08   if LENGTH(string)=0 then do;say "String not provided";exit 99;end
09   jobix=0
10   delstack
11   queue "SRESET;OWNER" userid";ST OJOB"
12   queue ""
13   rc = ejesrexx("execapi")
14   if rc>0 then do i=1 to EJES_Msg.0;say EJES_Msg.i;end
15   else do
16     say "Searching" EJES_Lines "jobs owned by" userid "..."
17     address ejes "execapi 0 'SELECT * B'"
18     if rc>0 then do i=1 to EJES_Msg.0;say EJES_Msg.i;end
19     else,
20       do while EJES_FunType = "BROWSER"
21         queue "FIND" string
22         queue ""
23         address ejes "execapi"
24         if rc>0 then delstack
25         else do
26           jobix = jobix + 1
27           jobname.jobix = EJES_JobName
28           jobid.jobix = EJES_JobId
29         end
30         address ejes "execapi 0 END"
31       end
32   end
33   address ejes "termapi"
34   if jobix=0 then say "String not found in jobs owned by" userid
35   else do
36     say "String found in" jobix "jobs owned by" userid":"
37     say "Jobname  JobId"
38     say "-------- --------"
39     do i=1 to jobix
40       say LEFT(jobname.i,8," ") LEFT(jobid.i,8," ")
41     end
42   end
43   exit
```

and the loop continues. Otherwise, control returns to the STATUS display and the loop ends.

Line 33:

- The EJESREXX host command environment is ended.

Line 34:

- If **jobix** is still zero, it means the string was not found in any job. A message is produced to that effect.

Lines 36 through 41:

- A very simple report is generated showing the job name and id of each job containing the string. Variables set by EJESREXX are blank stripped. The LEFT function is used in line 40 to pad the values to eight characters so everything lines up.

Normally, searching through multiple jobs is a manual effort. A sample like this will greatly simplify that task. Of course, just listing the jobs is the easiest thing for the sample REXX program to do. In real-life situations, you might want it to issue commands, archive the jobs, send email, etc.

System REXX

One of the cool things about REXX on z/OS is System REXX. This component was integrated with z/OS 1.9, but can be installed on earlier releases as well.

REXX programs that leverage EJESREXX will run just fine under System REXX. That means they can be called from programs that use the AXREXX macro/service or be invoked directly from the z/OS console.

Here I'm invoking both REXX samples from a console. In my environment, @ is the command character for System REXX.

Summary

Writing this article was fun. In the past, all of my experience with EJESREXX dealt with implementing, testing and debugging the code. For once, I got to put on my creative user "hat" and actually "play" with EJESREXX. In doing so, I hope I've provided you with incentive and ideas for writing your own scripted functions using EJESREXX.

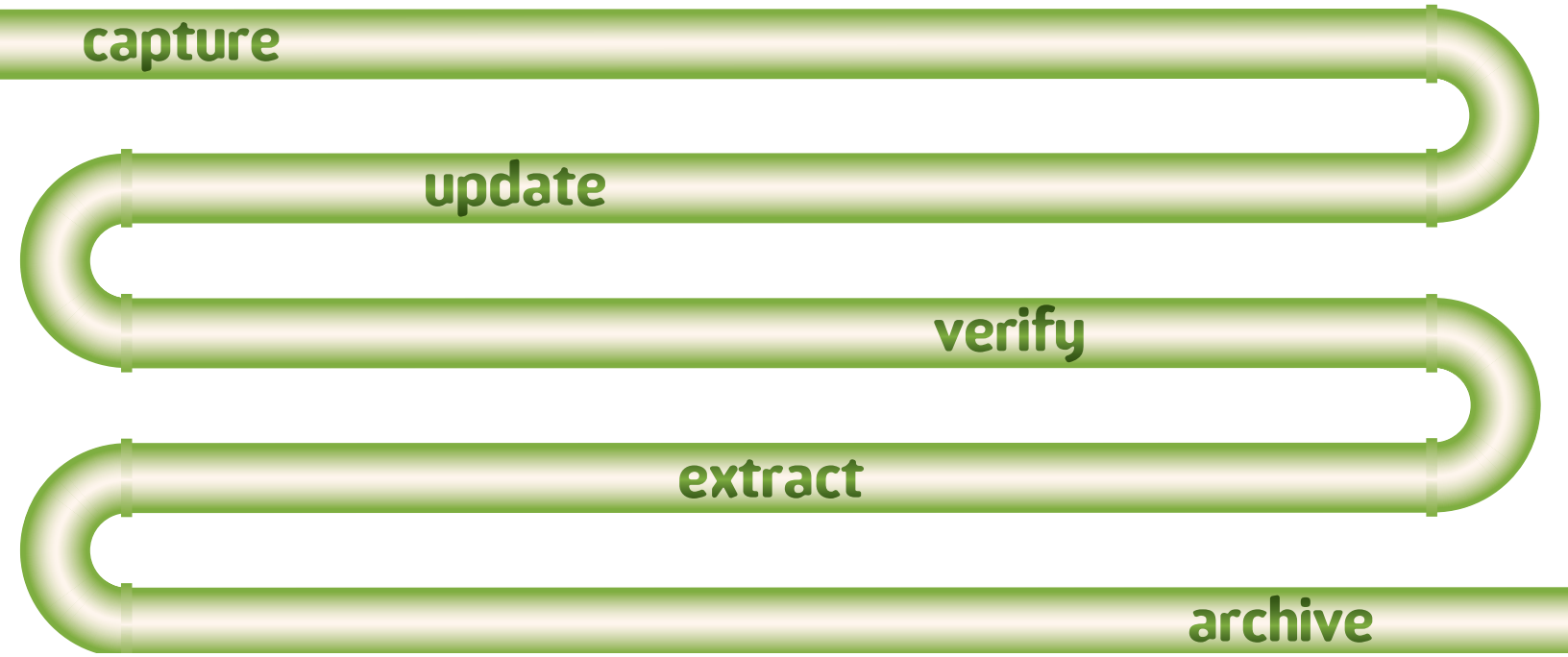
I know (E)JES pretty well, so I was able to put both sample programs together in an afternoon. I think you'll agree that they do quite a lot for such simple programs. The speed of this development readily illustrates the value proposition of the REXX language coupled with the power and simplicity of the EJESREXX interface.

And, given how painless this was, I'll probably agree to contribute future articles demonstrating more advanced EJESREXX techniques. If you have ideas for specific samples you'd like to see, send me an email. I'll be sure to have any scripts I write included as samples in the next (E)JES release. ■

```
@SRCHJOBS JOEUSER CVT
AXR0500I AXREXX OUTPUT DISPLAY 415
EXECNAME=SRCHJOBS REQTOKEN=00004000000000000C3FBC96E8D03F284
Searching 3 jobs owned by JOEUSER ...
String found in 1 jobs owned by JOEUSER:
Jobname  JobId
-------- --------
GENRCVT  J0060765

@SRCHJOBS JOEUSER ABEND
AXR0500I AXREXX OUTPUT DISPLAY 417
EXECNAME=SRCHJOBS REQTOKEN=00004000000000000C3FBC97F4009A705
Searching 3 jobs owned by JOEUSER ...
String found in 2 jobs owned by JOEUSER:
Jobname  JobId
-------- --------
GENRABND J0060764
GENRCVT  J0060765

@CLUPJOBS JOEUSER
IEA630I  OPERATOR AXRUSER  NOW ACTIVE,   SYSTEM=MVSA0    , LU=ISZ001
EJES510 USER-AXRUSER- $CJ(60764),P
$CJ(60764),P
EJES510 USER-AXRUSER- $CJ(60765),P
$CJ(60765),P
EJES510 USER-AXRUSER- $CJ(60766),P
$HASP890 JOB(GENRABND) 438
$HASP890 JOB(GENRABND)  STATUS=(AWAITING PURGE),CLASS=A,
$HASP890                PRIORITY=1,SYSAFF=(ANY),HOLD=(NONE),
$HASP890                PURGE=YES,CANCEL=YES
$CJ(60766),P
$HASP890 JOB(GENRCVT) 440
$HASP890 JOB(GENRCVT)   STATUS=(AWAITING PURGE),CLASS=A,
$HASP890                PRIORITY=1,SYSAFF=(ANY),HOLD=(NONE),
$HASP890                PURGE=YES,CANCEL=YES
$HASP890 JOB(GENRGET) 442
$HASP890 JOB(GENRGET)   STATUS=(AWAITING PURGE),CLASS=A,
$HASP890                PRIORITY=1,SYSAFF=(ANY),HOLD=(NONE),
$HASP890                PURGE=YES,CANCEL=YES
$HASP250 GENRABND PURGED -- (JOB KEY WAS C3FBC8E7)
$HASP250 GENRCVT PURGED -- (JOB KEY WAS C3FBC8E8)
$HASP250 GENRGET PURGED -- (JOB KEY WAS C3FBC8E9)
IEA631I  OPERATOR AXRUSER  NOW INACTIVE, SYSTEM=MVSA0    , LU=ISZ001
AXR0500I AXREXX OUTPUT DISPLAY 432
EXECNAME=CLUPJOBS REQTOKEN=00004000000000000C3FBCC3ED700B604
Deleting 3 jobs.
```

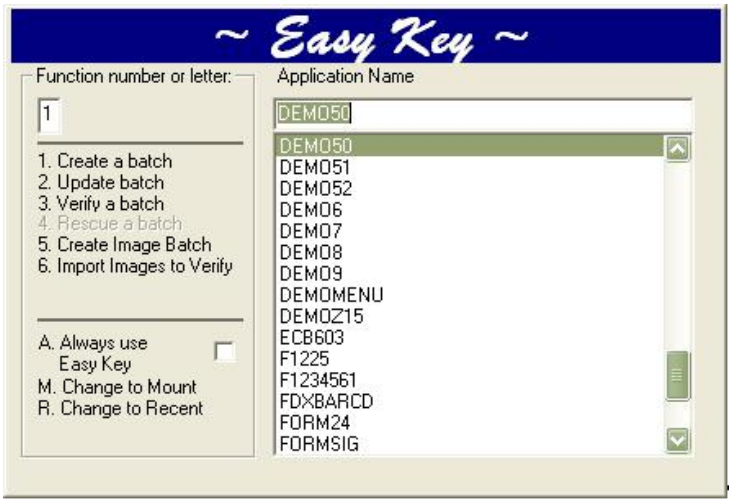


Falcon32® V8R1 Presents a New Way to Work

By Robert Blum

Robert Blum is a Senior Software Developer at Phoenix Software and project manager for FALCON® on the workstation. He oversaw the launch of PC/Falcon® in 1987 and Falcon32 in 1995. He and his team have taken PC/Falcon from a simple workstation implementation of mainframe FALCON to the robust Windows®-based product Falcon32 is today.

Falcon32 V8R1 has a new workflow management feature. Now, Falcon32 customers have a choice between two work processing modes—the classic mode or a workflow-managed mode.

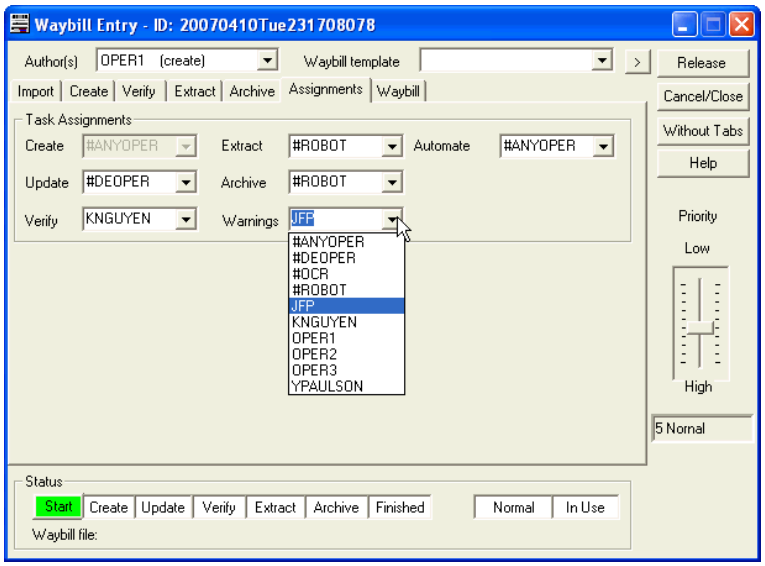


Classic Falcon32 Data Entry

In this mode, professional data entry operators control the flow of documents through the system. This is the way you have probably been using Falcon32, that is, using **Easy Key** to start the application.

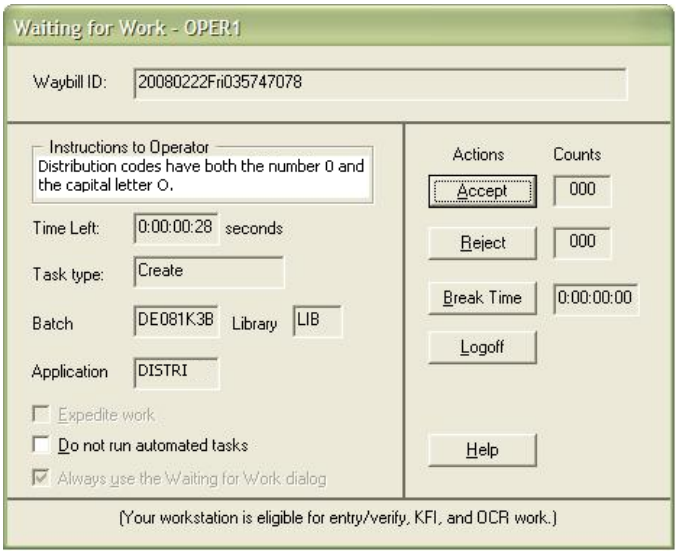
New Workflow Management

New with V8R1, is the workflow-managed mode. In this mode, supervisors use a simple procedure to automate the intake of images to create batches. An analogous process exists for paper workflow. Supervisors can prioritize work and designate which operators are eligible to work on each job. Data entry and verify operators don't need to learn to navigate the program's menus or toolbars, they only need to learn how to log on.



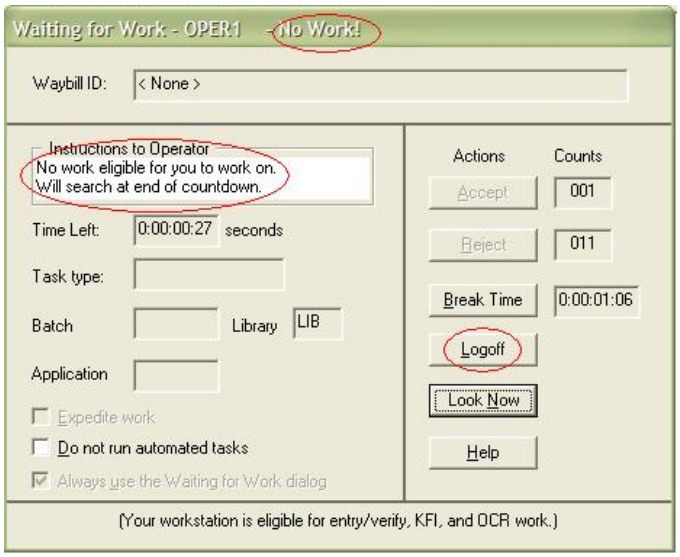
So, when the data entry operator logs on to Falcon32…

…the next job is pushed to that workstation using the **Waiting for Work** dialog. The operator presses **Accept** to begin work.



The operator keys or verifies the data, then saves the batch. Immediately after saving, Falcon32 displays more work for the operator. Verify or update batches can also be presented, as permitted for the operator.

When Falcon32 finds no more work for the operator, the **Waiting for Work** dialog offers only the options of logging off or waiting.



Non-supervisory personnel are therefore restricted based on the requirements of the work site. This new feature offers security, increased productivity, and reduced training costs. ■

Our Community Involvement at SHARE, Summer 2009

MVS Core Technologies Project Volunteer Coordination
Session #2201
Sun Aug 23, 4:30 - 5:30 PM

MVS Core Technologies Project (MVSE) Opening and z/OS Hot Topics
Session #2200
Mon Aug 24, 11:00 - 12:00 PM

zNextGen Open Panel Discussion: The Experienced Weigh in on What YOU, the New Mainframer, Need to Know
Session #2505
Mon Aug 24, 4:30 - 6:00 PM

System z10: Enterprise Class (EC) - User Experience
Session #2215
Tue Aug 25, 11:00 - 12:00 PM

JES Spool Interface Product Shoot-Out: (E)JES, IOF, OMC-FLASH, and SYSVIEW
Session #2342
Tue Aug 25, 1:30 - 2:30 PM

MVS Core Technologies Project Dinner Event
Session #2202
Tue Aug 25, 7:15 - 9:15 PM

(E)JES Lost Treasures Hands-On Lab
Session #2355
Thu Aug 27, 4:30 - 5:30 PM

MVS Core Technologies (EPS, ISPF, JES2, JES3, MVSE, and MVSO) Q&A Free For All
Session #2243
Thu Aug 27, 6:00 - 7:00 PM

Bit Bucket x'26'
Session #2208
Fri Aug 28, 11:00 - 12:00 PM

PHOENIX Software International®
www.phoenixsoftware.com
(310) 338-0400
info@phoenixsoftware.com