IBM WebSphere Liberty Java Batch

Hands-on Labs

Lab Version Date: February 15, 2018

© 2017, IBM Corporation

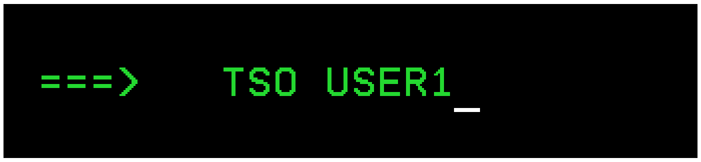
Table of Contents

© 2017, IBM Corporation

Unit 2 - Server Creation and Setup

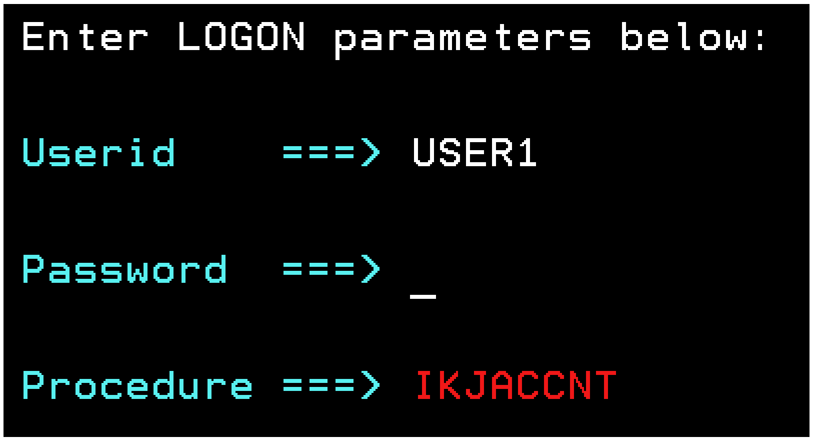
Create servers and start as z/OS Started Tasks

- □ Double click on the "WG31" icon you see on the desktop. This opens up a 3270 session.
- □ Enter `TSO USER1` and press the host enter key (which is the right-Ctrl key):



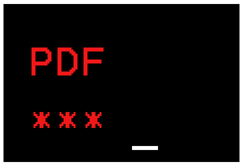
```
===>   TSO USER1_
```

- □ Provide the password for the `USER1` ID (the instructors will provide) and press host enter:



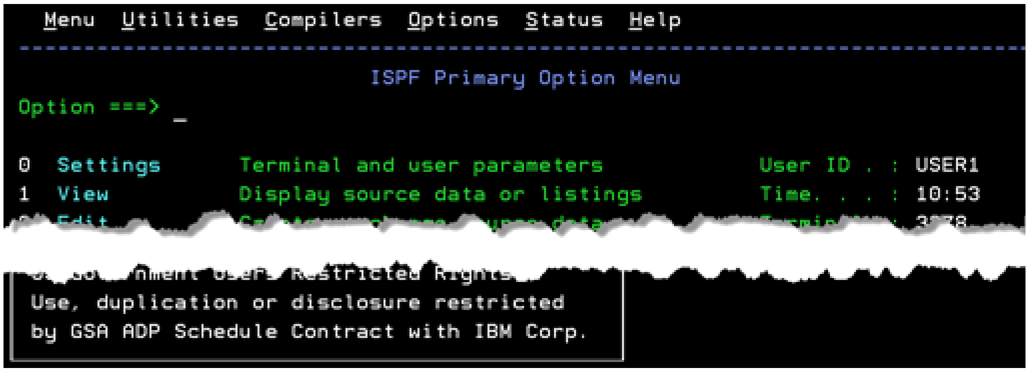
```
Enter LOGON parameters below:

Userid     ===> USER1

Password   ===> _

Procedure ===> IKJACCNT
```

- □ When you see the following, press host enter (right-Ctrl) to clear:



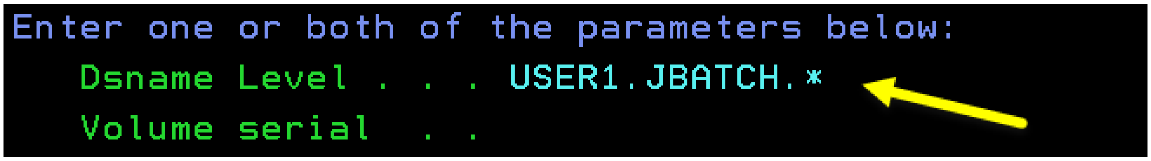
```
PDF
*** _
```

You will then see the "ISPF Primary Option Menu" screen:



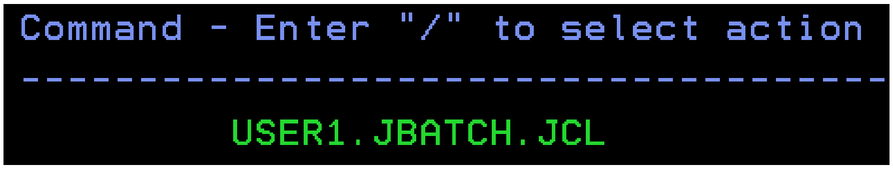
```
  Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                          ISPF Primary Option Menu
 Option ===> _

 0  Settings      Terminal and user parameters            User ID . : USER1
 1  View          Display source data or listings         Time. . . : 10:53
 2  Edit          Create or change source data            Terminal. : 3278

 U.S. Government Users Restricted Rights
 Use, duplication or disclosure restricted
 by GSA ADP Schedule Contract with IBM Corp.
```

- □ At the "Option" input field at the top, key in `=3.4` and press the host enter key.
- □ Then, at the "Data Set List Utility" panel, key in `USER1.JBATCH.*` and press host enter:



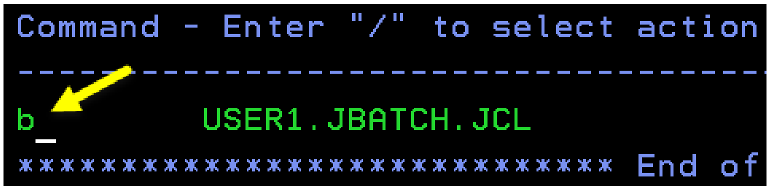
```
Enter one or both of the parameters below:
   Dsname Level . . . USER1.JBATCH.*
   Volume serial  . .
```

This will display a list of data sets that match that criteria. There should be only one:



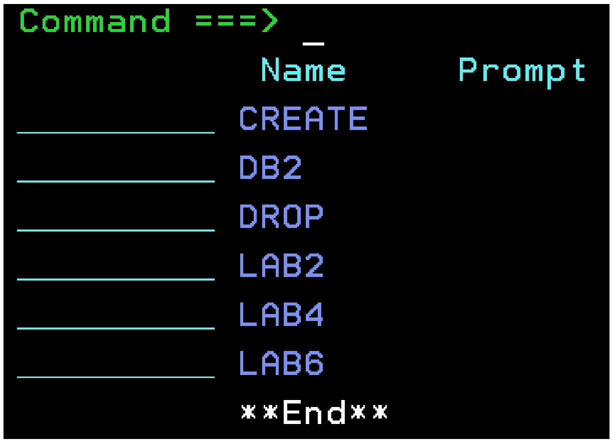
```
Command - Enter "/" to select action
---------------------------------------
          USER1.JBATCH.JCL
```

□ Place a **b** (for "browse") in the prefix area to the left of the data set, then press host enter:



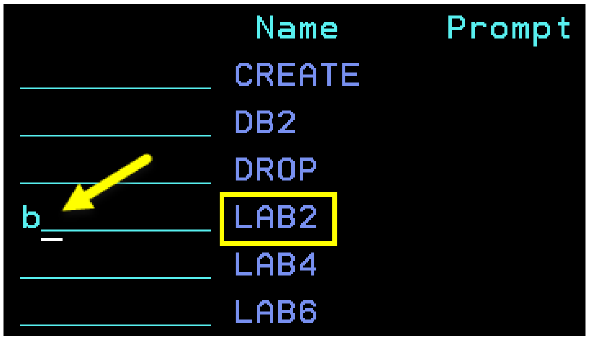
```
Command - Enter "/" to select action
-----------------------------------
b_        USER1.JBATCH.JCL
***************************** End of
```

This will display the "members" in the data set:



```
Command ===> _
           Name      Prompt
_________ CREATE
_________ DB2
_________ DROP
_________ LAB2
_________ LAB4
_________ LAB6
          **End**
```

□ Place a **b** (again, for "browse") in the prefix area to the left of the `LAB2` member:

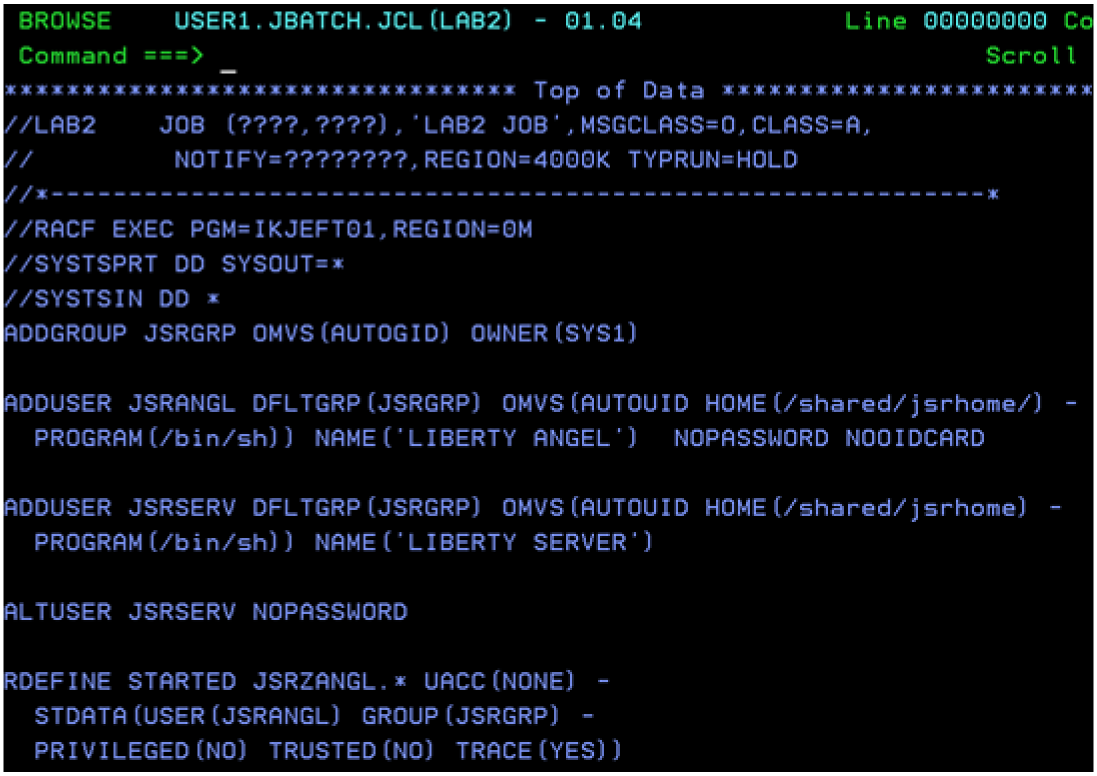


```
           Name      Prompt
_________ CREATE
_________ DB2
_________ DROP
b________ LAB2
_________ LAB4
_________ LAB6
```

Press the host enter key.

© 2017, IBM Corporation

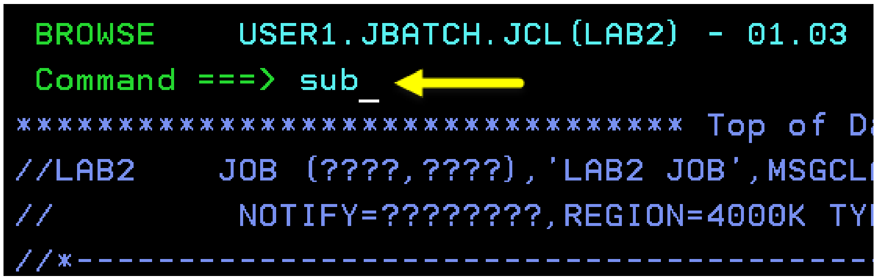
□ You will now see the contents of that member. You are in "browse" mode, so you can't modify anything. Use the scroll function -- F8 for 'down', F7 for 'up' -- to look at what this job is doing.



```
 BROWSE    USER1.JBATCH.JCL(LAB2) - 01.04              Line 00000000 Co
 Command ===> _                                                  Scroll
********************************* Top of Data ************************
//LAB2    JOB (????,????),'LAB2 JOB',MSGCLASS=O,CLASS=A,
//         NOTIFY=????????,REGION=4000K TYPRUN=HOLD
//*----------------------------------------------------------------*
//RACF EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
ADDGROUP JSRGRP OMVS(AUTOGID) OWNER(SYS1)

ADDUSER JSRANGL DFLTGRP(JSRGRP) OMVS(AUTOUID HOME(/shared/jsrhome/) -
  PROGRAM(/bin/sh)) NAME('LIBERTY ANGEL')  NOPASSWORD NOOIDCARD

ADDUSER JSRSERV DFLTGRP(JSRGRP) OMVS(AUTOUID HOME(/shared/jsrhome) -
  PROGRAM(/bin/sh)) NAME('LIBERTY SERVER')

ALTUSER JSRSERV NOPASSWORD

RDEFINE STARTED JSRZANGL.* UACC(NONE) -
  STDATA(USER(JSRANGL) GROUP(JSRGRP) -
  PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

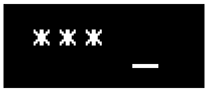
Notes: This job is creating a few RACF profiles -- a group ID, two user IDs, a set of `STARTED` profiles, and a `SURROGAT` profile. The `SURROGAT` is what allows your `USER1` ID to "switch user" to the server `JSRSERV` ID without requiring a password.

□ In the "Command" field at the top, key in **sub** and press the host enter key:



```
 BROWSE    USER1.JBATCH.JCL(LAB2) - 01.03
 Command ===> sub_
********************************* Top of D
//LAB2    JOB (????,????),'LAB2 JOB',MSGCL
//         NOTIFY=????????,REGION=4000K TY
//*---------------------------------------
```

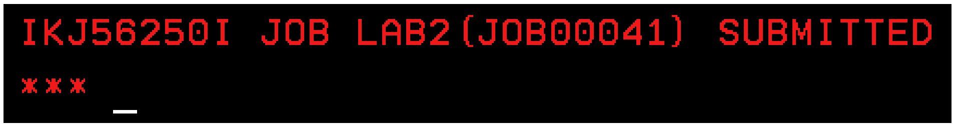
You should see three stars at the bottom of your screen:



```
 ***
     _
```

Press host enter to clear that screen hold.

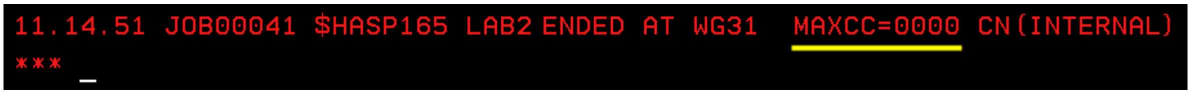
□ You should then see a message indicating the job has been submitted:



```
IKJ56250I JOB LAB2(JOB00041) SUBMITTED
***
    _
```

Press host enter to clear the screen hold indicated by the three stars.

□ The job runs fairly quickly, so you should now see the return code:

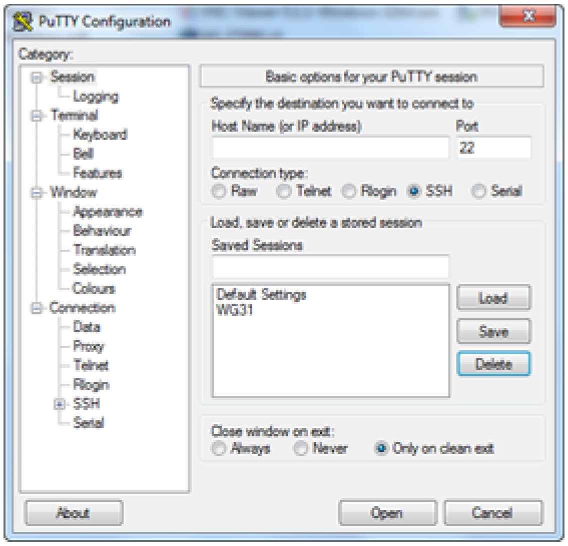


```
11.14.51 JOB00041 $HASP165 LAB2 ENDED AT WG31  MAXCC=0000 CN(INTERNAL)
*** _
```

The string `MAXCC=0000` means the job ran without errors.

If you don't see this message, it means the job is still running. Press host enter again and repeat until you do see this message. Then press host enter to clear the screen hold indicated by the three stars.

□ Locate the "PuTTY" icon on the desktop and double-click on it. It will bring up this panel:



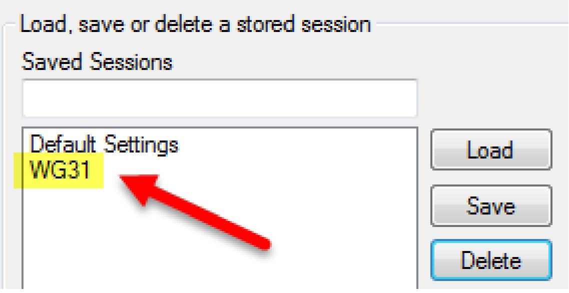
Port: 22

Connection type: SSH

Saved Sessions:
Default Settings
WG31

Close window on exit: Only on clean exit

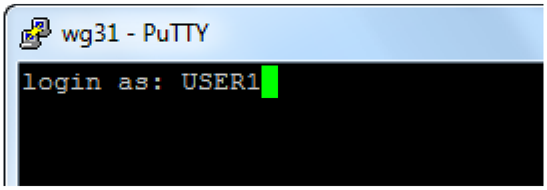
□ Double-click on "WG31" under "Save Sessions":



Default Settings
WG31

That will open up a PuTTY session to your z/OS system.

□ Log in with **USER1**:



```
login as: USER1
```

Press the *PC enter key* (not the right-Ctrl).

□ Provide the password for `USER1` (it's the same password you used for TSO).

□ Enter the following command to create the directory under which the servers will operate:

```
mkdir /shared/jsrhome
```

- ☐ Now change the ownership so the `JSRSRV` ID has ownership:

  `chown JSRSERV:JSRGRP /shared/jsrhome`

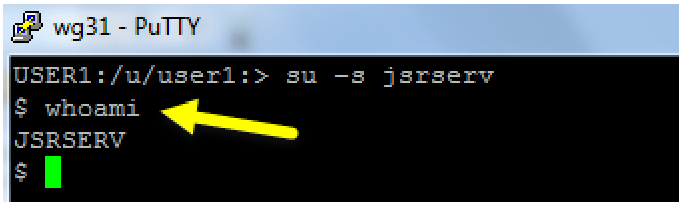
- ☐ Finally, change the permissions so "other" has no permissions:

  `chmod 750 /shared/jsrhome`

- ☐ Enter the following command to "switch user" to the `JSRSERV` ID we'll use for the Liberty z/OS servers:

  `su -s JSRSERV`

- ☐ Verify that you are indeed `JSRSERV` by entering the command **`whoami`** as shown here:

  ```
  USER1:/u/user1:> su -s jsrserv
  $ whoami
  JSRSERV
  $
  ```

  If it responds back with `JSRSERV`, then you are currently operating as that ID[1].

  Hint: At this point you may want to maximize the PuTTY terminal screen so you can see the full command after you paste it.

- ☐ Change directories to the Liberty z/OS installation `/bin` location:

  `cd /shared/zWebSphere/Liberty/V17001/bin`

- ☐ Set the JAVA_HOME variable so it points to a valid 64-bit Java:

  `export JAVA_HOME=/shared/java/J8.0_64/`

- ☐ Now set the WLP_USER_DIR location. This tells the environment where the servers will be created:

  `export WLP_USER_DIR=/shared/jsrhome`

- ☐ We're going to create all four servers right here even though we won't really use three of them until later in the workshop[2]. Enter the following four commands:

  - ○ **`./server create JSRDISP`**
    You should see: "Server JSRDISP created."

  - ○ **`./server create JSREXEC1`**
    You should see: "Server JSREXEC1 created."

  - ○ **`./server create JSREXEC2`**
    You should see: "Server JSREXEC2 created."

  - ○ **`./server create JSRMON`**
    You should see: "Server JSRMON created."

- ☐ Enter the following command to list the servers:

  `ls -al /shared/jsrhome/servers`

  You should see something like this:

  ```
  $ ls -al /shared/jsrhome/servers
  total 128
  drwxr-x--T   8 JSRSERV  JSRGRP      8192 Mar 27 14:12 .
  ```

[1] The ability to "switch user" without a password was granted with the SURROGAT profile generated in the LAB2 job.

[2] We will, however, start and verify all four in this lab.

```
drwxr-x---    3 JSRSERV  JSRGRP      8192 Mar 27 14:11 ..
drwxr-x--T    3 JSRSERV  JSRGRP      8192 Mar 27 14:11 .classCache
drwxr-x---    2 JSRSERV  JSRGRP      8192 Mar 27 14:12 .logs
drwxr-x---    5 JSRSERV  JSRGRP      8192 Mar 27 14:11 JSRDISP
drwxr-x---    5 JSRSERV  JSRGRP      8192 Mar 27 14:12 JSREXEC1
drwxr-x---    5 JSRSERV  JSRGRP      8192 Mar 27 14:12 JSREXEC2
drwxr-x---    5 JSRSERV  JSRGRP      8192 Mar 27 14:12 JSRMON
$
```

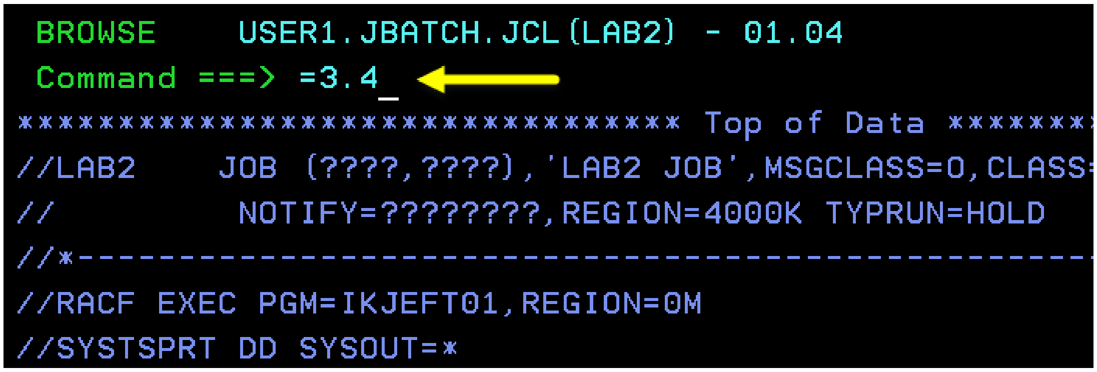
□ The default `server.xml` files contain things we don't need. We created a shell script that copies in some cleaned-up files. Issue the following command:

`/u/user1/jbatch/lab2/copy.sh`

You should see this:

```
$ ./copy.sh
Copying files for JSRDISP
Copying files for JSREXEC1
Copying files for JSREXEC2
Copying files for JSRMON
All done!
```

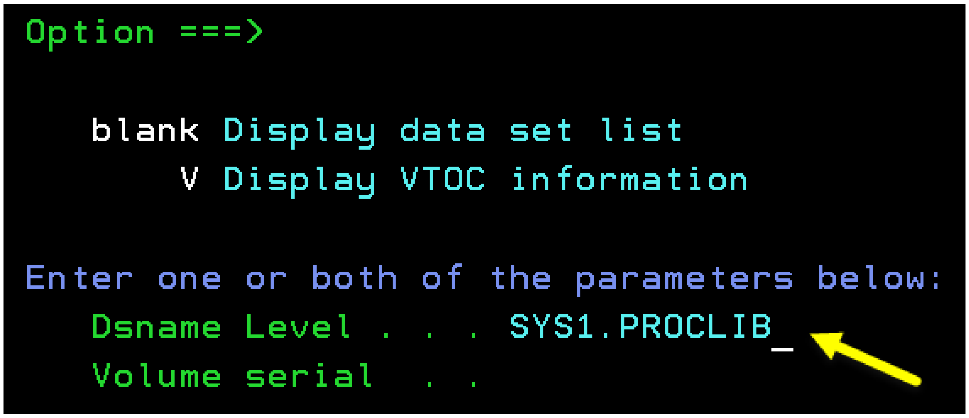
□ Go back to your 3270 session. In the command field at the top, enter **`=3.4`** and press the host enter key (right-Ctrl):



```
 BROWSE    USER1.JBATCH.JCL(LAB2) - 01.04
 Command ===> =3.4_
********************************* Top of Data ********
//LAB2    JOB (????,????),'LAB2 JOB',MSGCLASS=O,CLASS=
//          NOTIFY=????????,REGION=4000K TYPRUN=HOLD
//*----------------------------------------------------
//RACF EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
```

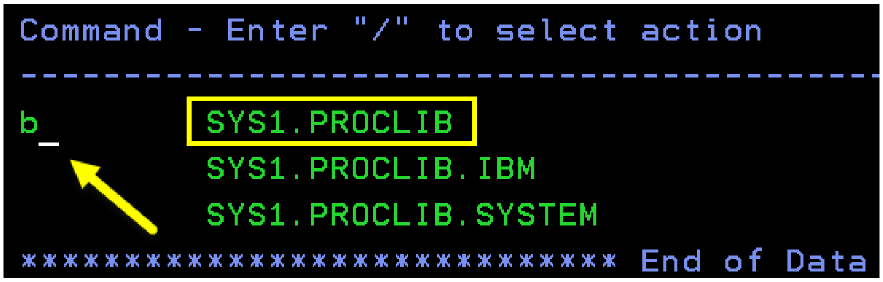
This will take you back to the "Data Set List Utility" panel.

□ On the "Dsname Level" field, enter **`SYS1.PROCLIB`** and press the host enter key:



```
Option ===>

   blank Display data set list
       V Display VTOC information

Enter one or both of the parameters below:
   Dsname Level . . . SYS1.PROCLIB_
   Volume serial  . .
```

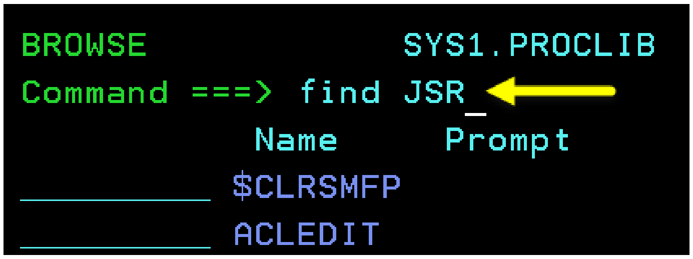
□ In the resulting list of data sets that match that pattern, but a **b** (for browse) next to the `SYS1.PROCLIB` data set:



```
Command - Enter "/" to select action
----------------------------------------------
b_          SYS1.PROCLIB
            SYS1.PROCLIB.IBM
            SYS1.PROCLIB.SYSTEM
***************************** End of Data
```

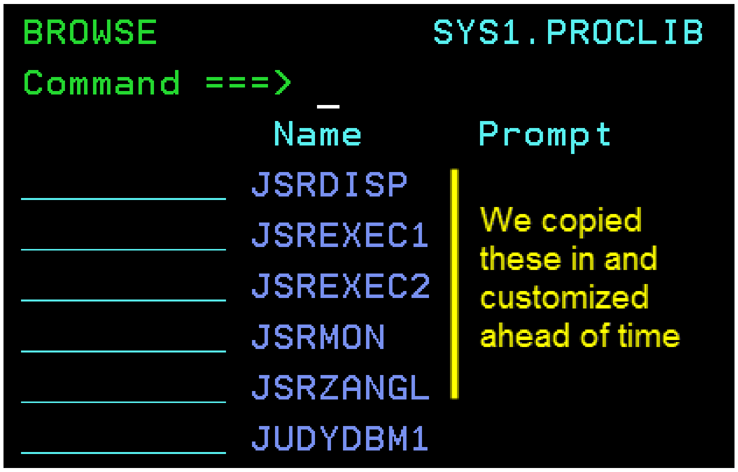
Press the host enter key.

□ In the command line, enter **`find JSR`** and press host enter:



```
BROWSE                    SYS1.PROCLIB
Command ===> find JSR_
                Name         Prompt
_____________  $CLRSMFP
_____________  ACLEDIT
```

That will find the first instance of a member that starts with "JSR". You should see five members that start with "JSR":

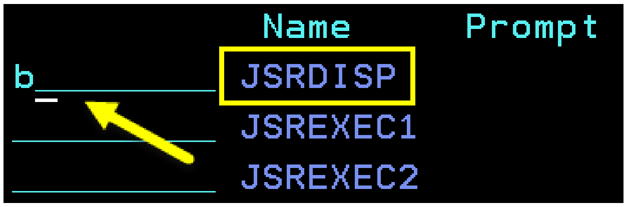


```
BROWSE                    SYS1.PROCLIB
Command ===> _
                Name         Prompt
_____________  JSRDISP
_____________  JSREXEC1
_____________  JSREXEC2
_____________  JSRMON
_____________  JSRZANGL
_____________  JUDYDBM1
```

We copied these in and customized ahead of time

Note: We copied and customized these to save a little time in the workshop. We copied them to `SYS1.PROCLIB` from the `/templates/zos/procs` location under the Liberty install path. The customization was minor ... we'll review the customization in the steps that follow.

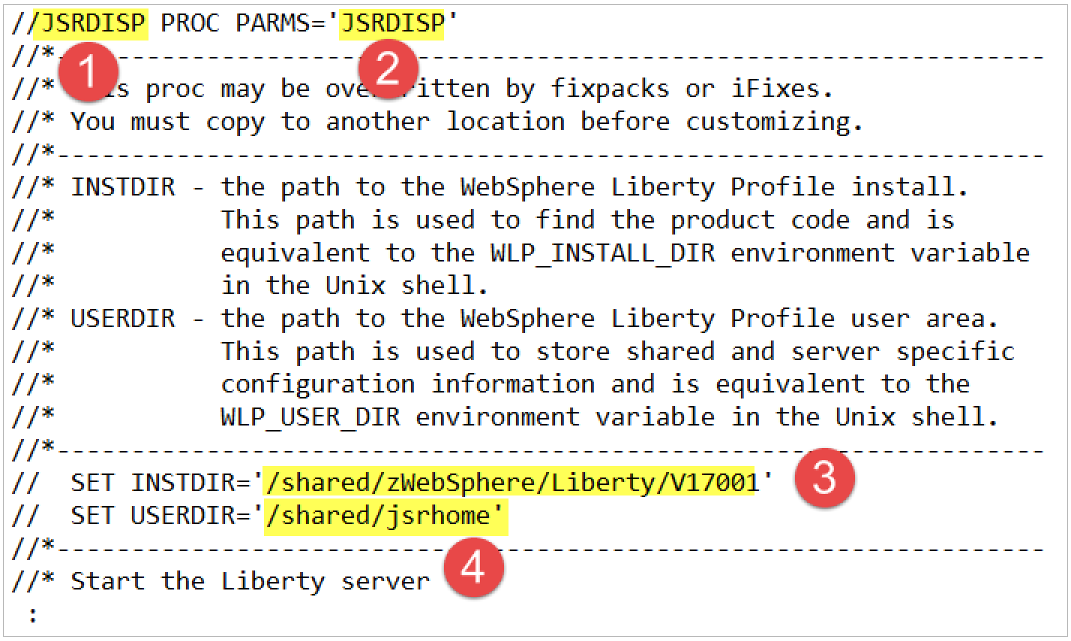
□ Place a **b** (for browse) next to the `JSRDISP` member:



```
                Name         Prompt
b_____________ JSRDISP
______________ JSREXEC1
______________ JSREXEC2
```

Then press the host enter key.

- □ The JCL start procedure will display. It will look like the following. The numbered blocks correspond to notes that follow:



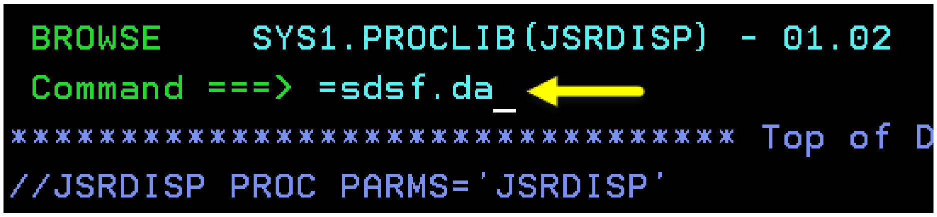
```
//JSRDISP PROC PARMS='JSRDISP'
//*-----------------------------------------------------------------
//*  s proc may be ove  itten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*-----------------------------------------------------------------
//* INSTDIR - the path to the WebSphere Liberty Profile install.
//*           This path is used to find the product code and is
//*           equivalent to the WLP_INSTALL_DIR environment variable
//*           in the Unix shell.
//* USERDIR - the path to the WebSphere Liberty Profile user area.
//*           This path is used to store shared and server specific
//*           configuration information and is equivalent to the
//*           WLP_USER_DIR environment variable in the Unix shell.
//*-----------------------------------------------------------------
//  SET INSTDIR='/shared/zWebSphere/Liberty/V17001'
//  SET USERDIR='/shared/jsrhome'
//*-----------------------------------------------------------------
//* Start the Liberty server
  :
```

Notes:

1. We changed the proc name from the default `BBGZSRV` to the name of the server, **`JSRDISP`** in this case.
2. We changed the default `PARMS='defaultServer'` to `PARMS='JSRDISP'` to explicitly name the server to be started. This avoids passing `PARMS=` in on the `START` command.
3. We set `INSTDIR=` to the location where Liberty z/OS is installed on these systems.
4. We set the `USERDIR=` value to the "user directory" of `/shared/jsrhome`.

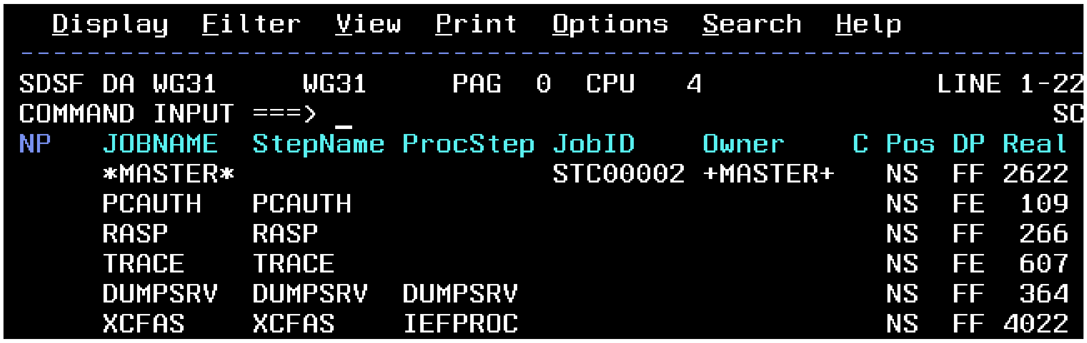
We customized the others -- `JSREXEC1`, `JSREXEC2`, `JSRMON` -- in the same way.

- □ In the command line at the top, enter **`=sdsf.da`** and press the host enter key:



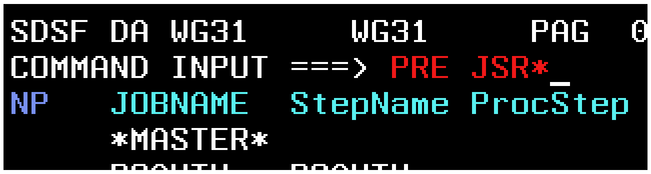
```
 BROWSE    SYS1.PROCLIB(JSRDISP) - 01.02
 Command ===> =sdsf.da_
********************************** Top of D
//JSRDISP PROC PARMS='JSRDISP'
```

You'll see something like this:



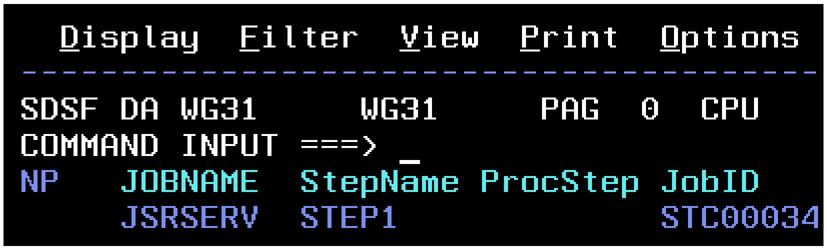
```
SDSF DA WG31     WG31     PAG  0  CPU    4          LINE 1-22
COMMAND INPUT ===> _                                        SC
NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real
     *MASTER*                   STC00002 +MASTER+   NS  FF 2622
     PCAUTH   PCAUTH                                NS  FE  109
     RASP     RASP                                  NS  FF  266
     TRACE    TRACE                                 NS  FE  607
     DUMPSRV  DUMPSRV  DUMPSRV                      NS  FF  364
     XCFAS    XCFAS    IEFPROC                      NS  FF 4022
```

□ To show *only* our jobs that start with "JSR," we'll set a prefix. Enter **PRE JSR*** in the command line and press host enter:



```
SDSF DA WG31      WG31      PAG  0
COMMAND INPUT ===> PRE JSR*_
NP    JOBNAME  StepName ProcStep
      *MASTER*
```

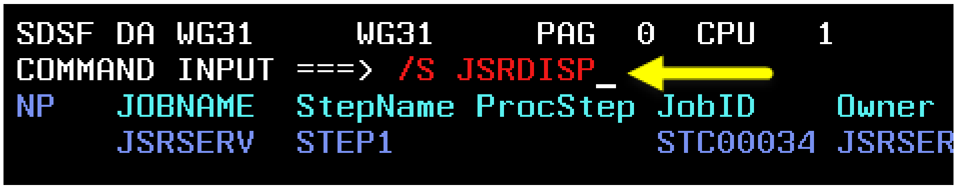
That should result in:



```
  Display  Filter  View  Print  Options
 ----------------------------------------
 SDSF DA WG31      WG31      PAG  0  CPU
 COMMAND INPUT ===> _
 NP    JOBNAME  StepName ProcStep JobID
       JSRSERV  STEP1             STC00034
```

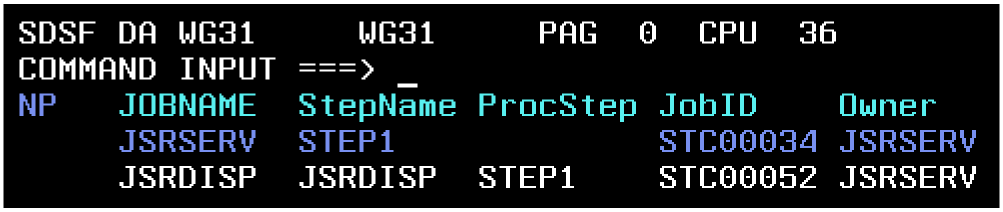
(That "JSRSERV" job is really your PuTTY session operating under the ID of JSRSERV.)

□ On the command line enter **/S JSRDISP** to start that server:



```
SDSF DA WG31      WG31      PAG  0  CPU    1
COMMAND INPUT ===> /S JSRDISP_
NP    JOBNAME  StepName ProcStep JobID    Owner
      JSRSERV  STEP1             STC00034 JSRSER
```

Press the host enter key. After a moment or two, press the host enter key again to refresh the screen. You should see:



```
SDSF DA WG31      WG31      PAG  0  CPU   36
COMMAND INPUT ===> _
NP    JOBNAME  StepName ProcStep JobID    Owner
      JSRSERV  STEP1             STC00034 JSRSERV
      JSRDISP  JSRDISP  STEP1    STC00052 JSRSERV
```

That indicates your `JSRDISP` server is up and running as a z/OS started task.

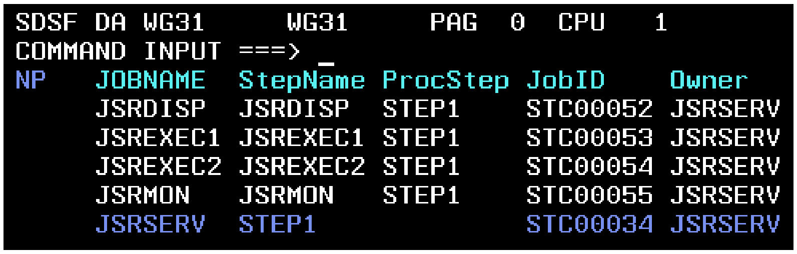
□ Now enter the other three server start commands:

○ **/S JSREXEC1**

○ **/S JSREXEC2**

○ **/S JSRMON**

You should now see all four servers running:



```
SDSF DA WG31      WG31      PAG  0  CPU    1
COMMAND INPUT ===> _
NP    JOBNAME  StepName ProcStep JobID    Owner
      JSRDISP  JSRDISP  STEP1    STC00052 JSRSERV
      JSREXEC1 JSREXEC1 STEP1    STC00053 JSRSERV
      JSREXEC2 JSREXEC2 STEP1    STC00054 JSRSERV
      JSRMON   JSRMON   STEP1    STC00055 JSRSERV
      JSRSERV  STEP1             STC00034 JSRSERV
```

© 2017, IBM Corporation

□ What TCP ports are those servers using? Let's check. Go back to your PuTTY terminal and enter the following command:

netstat | grep JSR

You should see:

```
$ netstat | grep JSR
JSRDISP   0000050C 0.0.0.0..10080        0.0.0.0..0            Listen
JSREXEC1  00000529 0.0.0.0..11080        0.0.0.0..0            Listen
JSREXEC2  00000531 0.0.0.0..12080        0.0.0.0..0            Listen
JSRMON    0000053A 0.0.0.0..13080        0.0.0.0..0            Listen
$
```

Each server's port value was set to be 1000 higher than the previous. You don't have to do it this way, it's just something we did to make sure the ports were unique.

Note: The SSL ports are 10443, 11443, 12443, and 13443, but they don't show because we've not yet started anything that requires those ports.

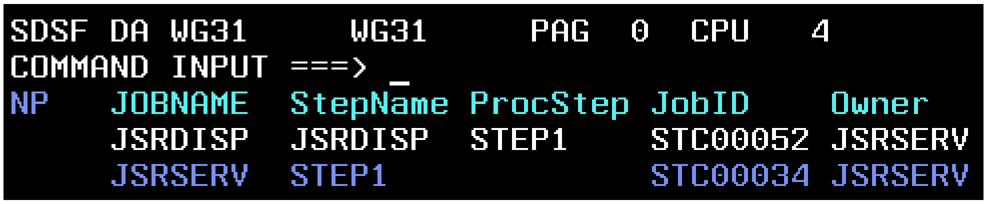
□ Go back to the 3270 session and shut down JSREXEC1, JSREXEC2, and JSRMON:

○ **/P JSREXEC1**

○ **/P JSREXEC2**

○ **/P JSRMON**

You should be left with just one active server -- JSRDISP:



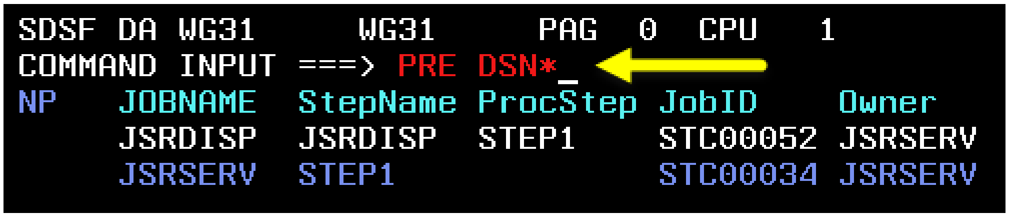
```
SDSF DA WG31      WG31      PAG  0  CPU   4
COMMAND INPUT ===> _
NP    JOBNAME  StepName ProcStep JobID    Owner
      JSRDISP  JSRDISP  STEP1    STC00052 JSRSERV
      JSRSERV  STEP1             STC00034 JSRSERV
```

Generate DDL and create Batch Persistence DB

The next thing to do is generate the DDL for the JobRepository persistence tables. To do this we need to add a few things to the server.xml so the ddlGen shell script can contact the server, and the server can communicate with DB2.

To keep this from being a typing exercise we'll have you copy in a pre-built XML. We'll show you what was added and explain why it was added.

□ Go to your 3270 session and set the prefix to **PRE DSN*** as shown here:

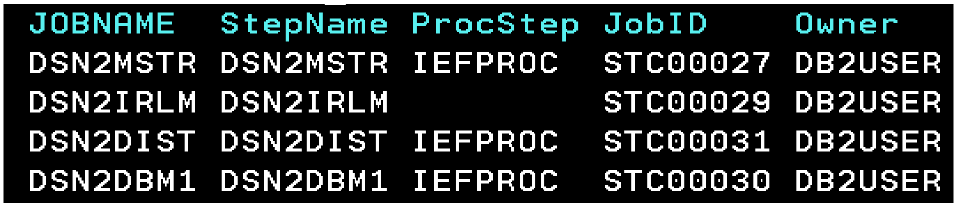


```
SDSF DA WG31      WG31      PAG  0  CPU   1
COMMAND INPUT ===> PRE DSN*_
NP    JOBNAME  StepName ProcStep JobID    Owner
      JSRDISP  JSRDISP  STEP1    STC00052 JSRSERV
      JSRSERV  STEP1             STC00034 JSRSERV
```

When you press the host enter key you should see no jobs. That's because DB2 is not yet started. You'll do that now.

□ Enter the command: **/-DSN2 START DB2**

Wait about 30 seconds, then refresh the screen with the host enter key. You should see four address spaces:



```
JOBNAME  StepName ProcStep JobID    Owner
DSN2MSTR DSN2MSTR IEFPROC  STC00027 DB2USER
DSN2IRLM DSN2IRLM          STC00029 DB2USER
DSN2DIST DSN2DIST IEFPROC  STC00031 DB2USER
DSN2DBM1 DSN2DBM1 IEFPROC  STC00030 DB2USER
```

□ Go back to your PuTTY session and enter the following command *as one line*:

```
cp /u/user1/jbatch/lab2/disp_ddl.xml
                            /shared/jsrhome/servers/JSRDISP/server.xml
```

The `server.xml` copied in looks like this. Take a few seconds and note what was **added**:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>localConnector-1.0</feature>
    </featureManager>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      createTables="false"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />
    <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

    <library id="DB2T4LibRef">
      <fileset dir="/shared/db21210/jdbc/classes/"
      includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
    </library>

    <authData id="batchAlias" user="SYSADM1" password="SYSADM1" />

    <dataSource id="batchDB"
      containerAuthDataRef="batchAlias"
      type="javax.sql.XADataSource"
      jdbcDriverRef="DB2T4">
     <properties.db2.jcc
       serverName="wg31.washington.ibm.com"
       portNumber="2446"
       databaseName="DSN2LOC"
       driverType="4" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
        host="*"
        httpPort="10080"
        httpsPort="10443" />

</server>
```

The two features enable the JSR-352 batch function as well as the JMX local connector; the rest is all batch persistence and DB2 z/OS definitions.

© 2017, IBM Corporation

□ Go back to your PuTTY session and enter the following commands:

○	`cd /shared/zWebSphere/Liberty/V17001/bin`
○	`export PATH=$PATH:/shared/java/J8.0_64/bin`

The first is just to make certain you're in the `/bin` directory of the Liberty install location.

The second is so the `java` command is on the PATH.

□ Now, generate the DDL with the following command:

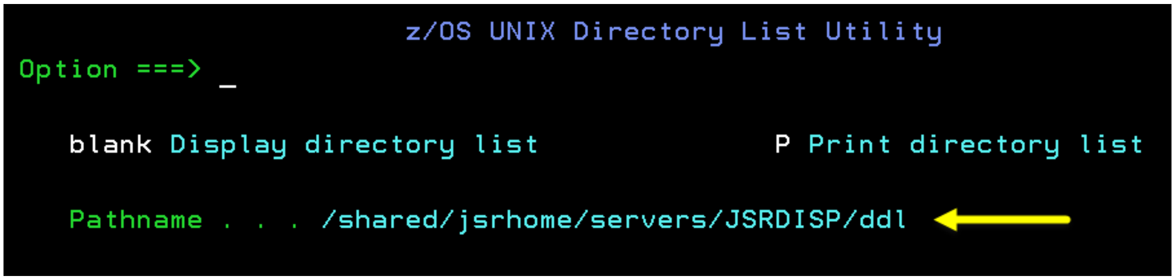
```
./ddlGen generate JSRDISP
```

You should get the following message:

```
CWWKD0107I: The requested DDL was generated in the following
directory: /shared/jsrhome/servers/JSRDISP/ddl
```

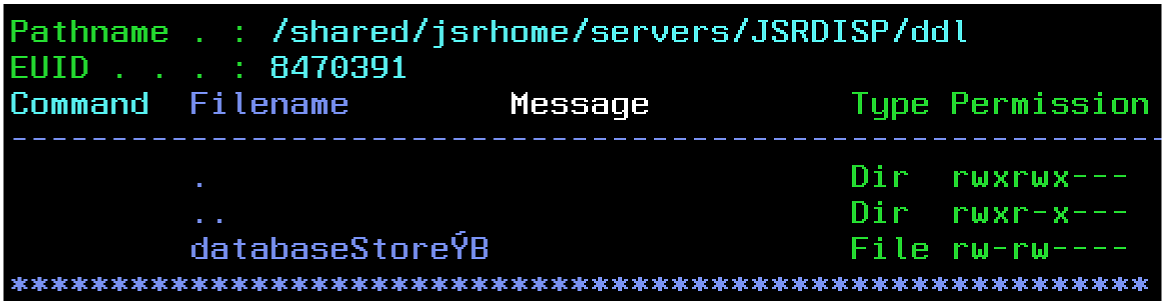
□ Go back to your 3270 session and enter **=3.17** from any command line.

□ Enter **/shared/jsrhome/servers/JSRDISP/ddl** on the "Pathname" field:



```
                     z/OS UNIX Directory List Utility
Option ===> _

   blank Display directory list              P Print directory list

   Pathname . . . /shared/jsrhome/servers/JSRDISP/ddl
```

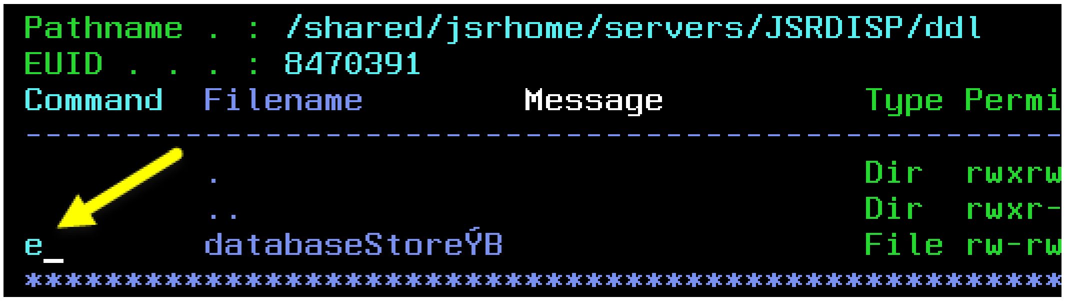
and press the host enter key. You should see:



```
Pathname . : /shared/jsrhome/servers/JSRDISP/ddl
EUID . . . : 8470391
Command  Filename         Message          Type Permission
-----------------------------------------------------------
         .                                 Dir  rwxrwx---
         ..                                Dir  rwxr-x---
         databaseStoreÝB                   File rw-rw----
***********************************************************
```

That's telling you there's one file in that directory.

□ Place an **e** (for edit) in the prefix area to the left of the file name:

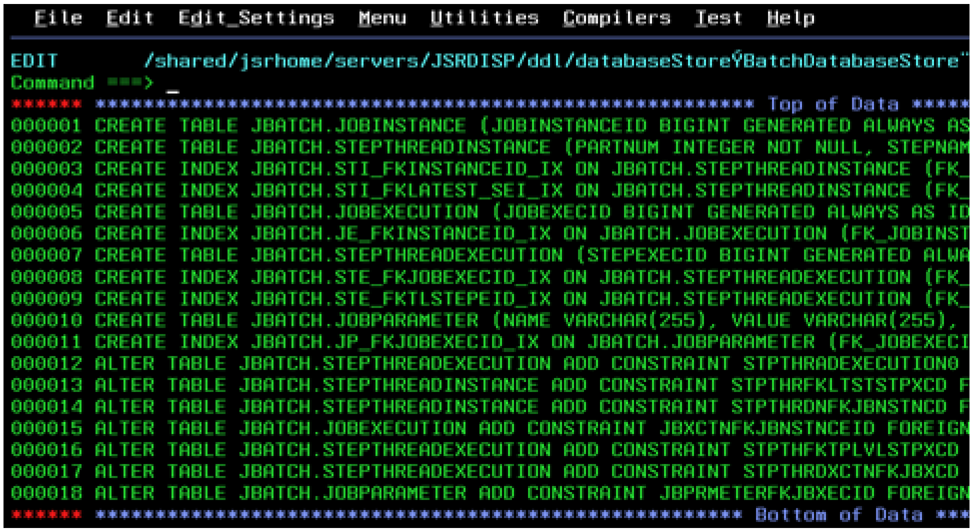


```
Pathname . : /shared/jsrhome/servers/JSRDISP/ddl
EUID . . . : 8470391
Command  Filename         Message          Type Permi
-----------------------------------------------------
         .                                 Dir  rwxrw
         ..                                Dir  rwxr-
e_       databaseStoreÝB                   File rw-rw
*****************************************************
```

Press the host enter key.

□ You'll see an "Edit Entry Panel" ... just press host enter to get past that.

□ You should now see:



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       /shared/jsrhome/servers/JSRDISP/ddl/databaseStoreÝBatchDatabaseStore"
Command ===> _
****** ******************************************************* Top of Data *****
000001 CREATE TABLE JBATCH.JOBINSTANCE (JOBINSTANCEID BIGINT GENERATED ALWAYS AS
000002 CREATE TABLE JBATCH.STEPTHREADINSTANCE (PARTNUM INTEGER NOT NULL, STEPNAM
000003 CREATE INDEX JBATCH.STI_FKINSTANCEID_IX ON JBATCH.STEPTHREADINSTANCE (FK_
000004 CREATE INDEX JBATCH.STI_FKLATEST_SEI_IX ON JBATCH.STEPTHREADINSTANCE (FK_
000005 CREATE TABLE JBATCH.JOBEXECUTION (JOBEXECID BIGINT GENERATED ALWAYS AS ID
000006 CREATE INDEX JBATCH.JE_FKINSTANCEID_IX ON JBATCH.JOBEXECUTION (FK_JOBINST
000007 CREATE TABLE JBATCH.STEPTHREADEXECUTION (STEPEXECID BIGINT GENERATED ALWA
000008 CREATE INDEX JBATCH.STE_FKJOBEXECID_IX ON JBATCH.STEPTHREADEXECUTION (FK_
000009 CREATE INDEX JBATCH.STE_FKTLSTEPEID_IX ON JBATCH.STEPTHREADEXECUTION (FK_
000010 CREATE TABLE JBATCH.JOBPARAMETER (NAME VARCHAR(255), VALUE VARCHAR(255),
000011 CREATE INDEX JBATCH.JP_FKJOBEXECID_IX ON JBATCH.JOBPARAMETER (FK_JOBEXECI
000012 ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT STPTHRADEXECUTION0
000013 ALTER TABLE JBATCH.STEPTHREADINSTANCE ADD CONSTRAINT STPTHRFKLTSTSTPXCD F
000014 ALTER TABLE JBATCH.STEPTHREADINSTANCE ADD CONSTRAINT STPTHRDNFKJBNSTNCD F
000015 ALTER TABLE JBATCH.JOBEXECUTION ADD CONSTRAINT JBXCTNFKJBNSTNCEID FOREIGN
000016 ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT STPTHFKTPLVLSTPXCD
000017 ALTER TABLE JBATCH.STEPTHREADEXECUTION ADD CONSTRAINT STPTHRDXCTNFKJBXCD
000018 ALTER TABLE JBATCH.JOBPARAMETER ADD CONSTRAINT JBPRMETERFKJBXECID FOREIGN
****** ****************************************************** Bottom of Data ***
```

That is the DB2 z/OS Data Definition Language (DDL) statements to create the JobRepository table for IBM Java Batch. Each line represents a command. The commands go *way* past the right side of the 3270 screen.

Notes: This file is in "tagged ASCII," which means the editor will display in readable text if you have the `_BPXK_AUTOCVT=ON` environment variable set.

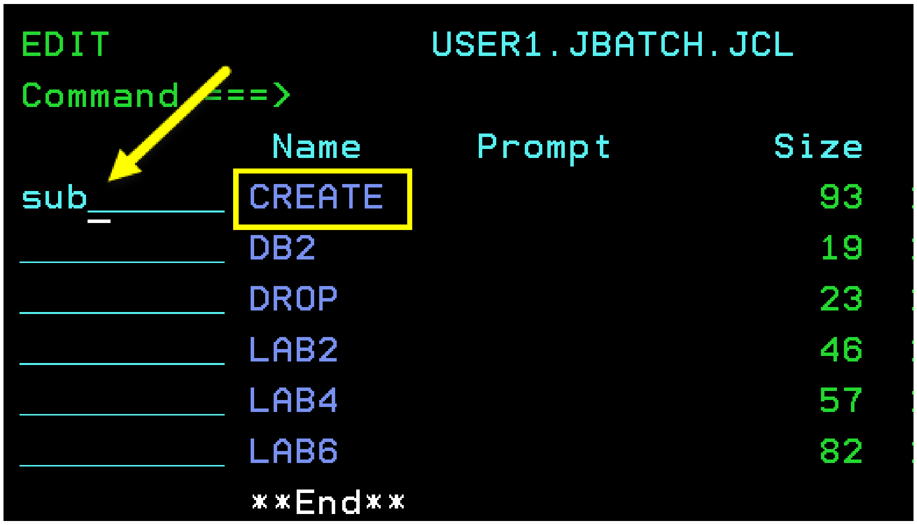
We ran this command ahead of the workshop, downloaded the output to our PC, reformatted all the lines so they'd fit within a 72-column JCL limit, then uploaded back to the mainframe so it would hit as EBCDIC. We wrappered it with JCL to create the tables. That's what you'll submit next.

□ In 3270, enter **=3.4**

□ Enter **USER1.JBATCH.JCL** and press host enter to display that dataset.

□ Place a **b** (for browse) next to that data set.

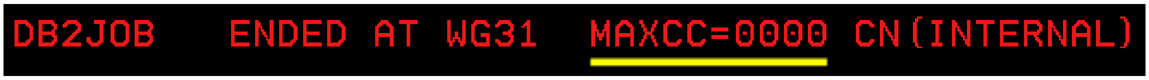
□ Submit the `CREATE` member[3] by entering **sub** next to the member and pressing the host enter key:



```
EDIT                  USER1.JBATCH.JCL
Command ===>
             Name       Prompt        Size
sub_______   CREATE                     93
__________   DB2                        19
__________   DROP                       23
__________   LAB2                       46
__________   LAB4                       57
__________   LAB6                       82
             **End**
```

[3] This is where we placed the DDL we generated. We broke the lines to fit within 72 columns, and we wrappered it with JCL so the tables could be created as a JCL batch job. Feel free to browse that member if you'd like.

□ This job takes about 10 to 20 seconds to complete, so hit the host enter key occasionally to refresh the screen. Eventually you should see the following:



```
DB2JOB    ENDED AT WG31   MAXCC=0000 CN(INTERNAL)
```

That means the tables were created.

□ We're almost there. Go back to your PuTTY session and enter the following command *as one line*:

```
cp /u/user1/jbatch/lab2/disp_app.xml
                              /shared/jsrhome/servers/JSRDISP/server.xml
```

The `server.xml` copied in looks like this. Take a few moments to note what's been ~~**removed**~~, and what's been **added**:

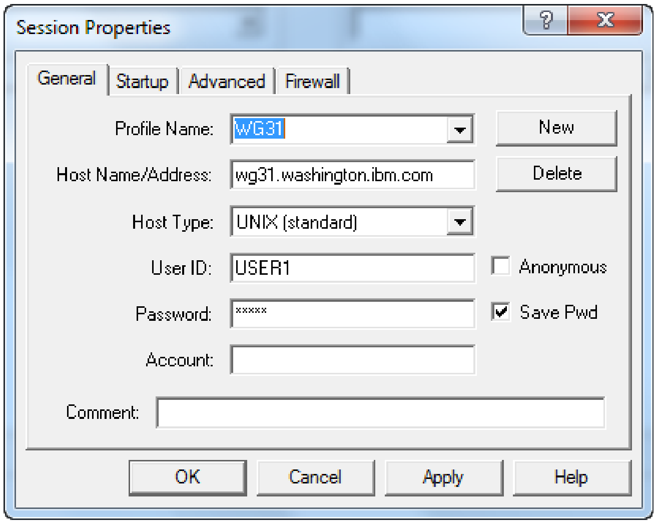
```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        ~~<feature>localConnector-1.0</feature>~~
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <basicRegistry id="basic1" realm="jbatch">
      <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="com.ibm.ws.batch">
      <security-role name="batchAdmin">
        <special-subject type="EVERYONE"/>
        <user name="Fred" />
      </security-role>
    </authorization-roles>

    <batchPersistence jobStoreRef="BatchDatabaseStore" />

    <databaseStore id="BatchDatabaseStore"
      createTables="false"
      dataSourceRef="batchDB" schema="JBATCH" tablePrefix="" />
    <jdbcDriver id="DB2T4" libraryRef="DB2T4LibRef" />

    <library id="DB2T4LibRef">
      <fileset dir="/shared/db21210/jdbc/classes/"
      includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
    </library>

    <authData id="batchAlias" user="SYSADM1" password="SYSADM1" />

    <dataSource id="batchDB"
      containerAuthDataRef="batchAlias"
      type="javax.sql.XADataSource"
      jdbcDriverRef="DB2T4">
     <properties.db2.jcc
       serverName="wg31.washington.ibm.com"
       portNumber="2446"
       databaseName="DSN2LOC"
       driverType="4" />
    </dataSource>

    <httpEndpoint id="defaultHttpEndpoint"
        host="*"
        httpPort="10080"
        httpsPort="10443" />

</server>
```

The `localConnect-1.0` feature is no longer needed, so that's removed. We add `batchManagement-1.0` and `appSecurity-2.0` to enable the IBM operational enhancements. The other red XML is in support of security. This is "basic" security.

- □ This would be a good time to look at the `messages.log` file and see if the new XML was loaded properly, or if there were problems. Locate the WS-FTP icon on the desktop and double click on that.

Note: The messages.log file is a "tagged ascii" file. It can be viewed with the OEDIT function in a 3270 session. We find it easier to use an FTP client to view the file. It ends up downloading the file and opening it in an editor on the PC.

- □ You should see the launch window for the tool, pre-loaded with the host and user information:



Profile Name: WG31

Host Name/Address: wg31.washington.ibm.com

Host Type: UNIX (standard)

User ID: USER1

□ Anonymous

Password: *****

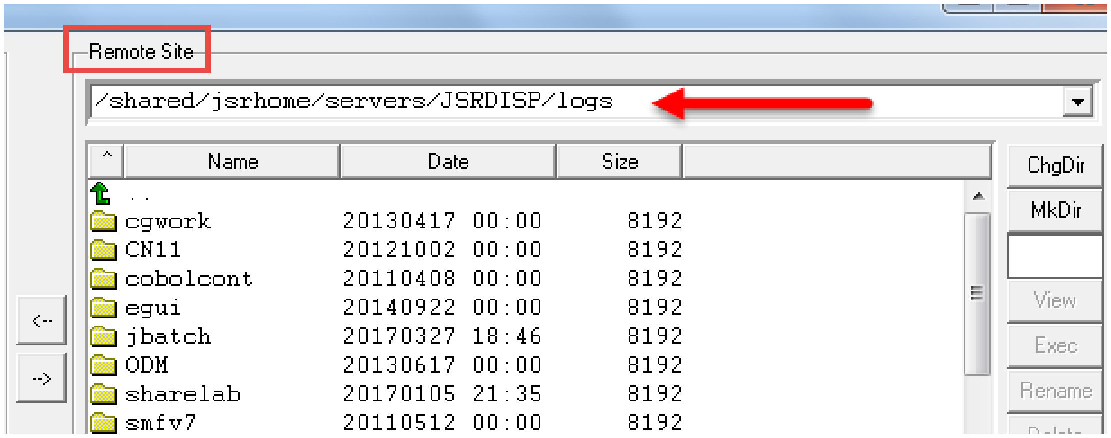
☑ Save Pwd

Account:

Comment:

Click "OK" to launch the tool.

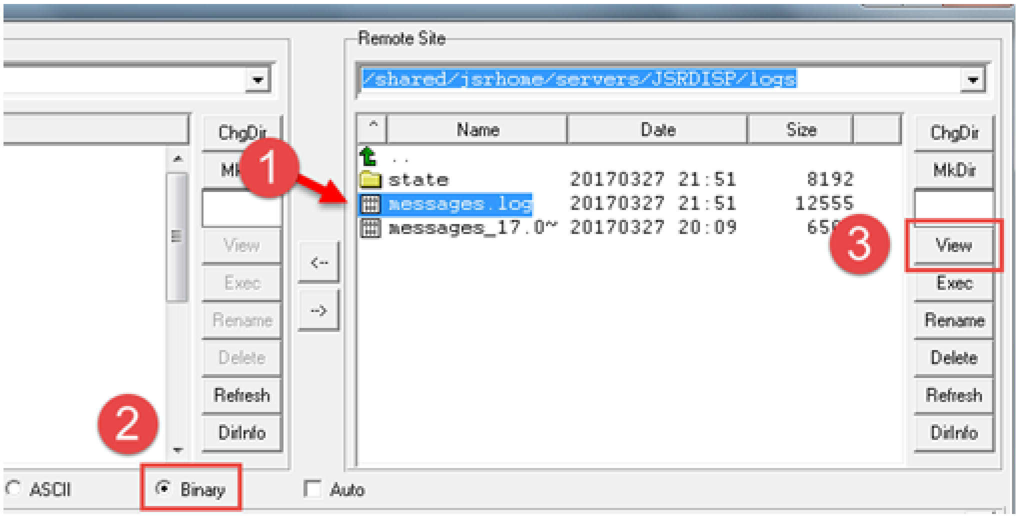
□ On the right side of the tool is the "remote site," which in this case is the z/OS system. Enter **`/shared/jsrhome/servers/JSRDISP/logs`** into the field, then press the PC enter key:



```
/shared/jsrhome/servers/JSRDISP/logs

Name           Date              Size
..
cgwork         20130417 00:00    8192
CN11           20121002 00:00    8192
cobolcont      20110408 00:00    8192
egui           20140922 00:00    8192
jbatch         20170327 18:46    8192
ODM            20130617 00:00    8192
sharelab       20170105 21:35    8192
smfv7          20110512 00:00    8192
```

That will result in a list of files in that directory being displayed.

□ Do the following:



```
/shared/jsrhome/servers/JSRDISP/logs

Name              Date              Size
..
state             20170327 21:51    8192
messages.log      20170327 21:51    12555
messages_17.0~    20170327 20:09    65
```

1 Highlight the `messages.log` file with a single click of the mouse.

2 Make sure "Binary" is selected

3 Click the "View" button

The result will the be the `messages.log` file displayed in a text editor on your PC.

□ If you scroll down and to the right you'll see the messages. Somewhere in the bottom 10 or 15 messages you should see:

```
CWWKF0012I: The server installed the following features: [ssl-1.0,
batchManagement-1.0, json-1.0, distributedMap-1.0, appSecurity-2.0].
```

and:

```
CWWKO0219I: TCP Channel defaultHttpEndpoint-ssl has been started and
is now listening for requests on host *  (IPv4) port 10443.
```

Those are good signs.

□ Close the text editor that is displaying the `messages.log` file.

Validate Java Batch using SleepyBatchlet Sample

We are now ready to test our first batch application and verify the environment.

□ In your PuTTY session, enter the following command *as one line*:

```
cp /u/user1/jbatch/apps/SleepyBatchletSample-1.0.war
  /shared/jsrhome/servers/JSRDISP/dropins/SleepyBatchletSample-1.0.war
```

That copies the "Sleepy Batchlet" sample application to the server's "dropins" directory where it will be detected and dynamically loaded.

□ Go to the WS-FTP client and view the `messages.log` file again. You should see the following message near the bottom:

```
CWWKZ0001I: Application SleepyBatchletSample-1.0 started in 0.089 seconds.
```

Close the file editor displaying the `messages.log` file.

□ Go to the PuTTY session and enter the following command *as one line*:

```
./batchManager submit --batchManager=localhost:10443 --user=Fred
                        --password=fredpwd --applicationName=SleepyBatchletSample-1.0
                      --jobXMLName=sleepy-batchlet.xml --trustSslCertificates --wait
```

You should see something like this:

```
[2017/03/27 18:33:44.129 -0400] CWWKY0101I: Job sleepy-batchlet with
instance ID 1 has been submitted.

[2017/03/27 18:33:44.133 -0400] CWWKY0106I: JobInstance:{"jobName":"sleepy-
batchlet","instanceId":1,"appName":"SleepyBatchletSample-
1.0#SleepyBatchletSample-
1.0.war","submitter":"Fred","batchStatus":"STARTING","jobXMLName":"sleepy-
batchlet.xml","instanceState":"SUBMITTED","lastUpdatedTime":"2017/03/27
22:33:40.534 +0000"}
```

Then after 15 seconds (the default "sleep" period) you should see:

```
[2017/03/27 18:34:14.322 -0400] CWWKY0105I: Job sleepy-batchlet with
instance ID 1 has finished. Batch status: COMPLETED. Exit status: COMPLETED

[2017/03/27 18:34:14.322 -0400] CWWKY0107I: JobExecution:{"jobName":"sleepy-
batchlet","executionId":1,"instanceId":1,"batchStatus":"COMPLETED","exitStat
us":"COMPLETED","createTime":"2017/03/27 22:33:42.994
+0000","endTime":"2017/03/27 22:33:59.469
+0000","lastUpdatedTime":"2017/03/27 22:33:59.469
+0000","startTime":"2017/03/27 22:33:44.170 +0000","jobParameters":
{},"restUrl":"https://192.168.17.219:10443/ibm/api/batch/","serverId":"local
host//shared/jsrhome/JSRDISP","logpath":"/shared/jsrhome/servers/JSRDISP/log
s/joblogs/sleepy-batchlet/2017-03-
27/instance.1/execution.1/","stepExecutions":
[{"stepExecutionId":1,"stepName":"step1","batchStatus":"COMPLETED","exitStat
us":"SleepyBatchlet:i=15;stopRequested=false","stepExecution":"https://local
host:10443/ibm/api/batch/jobexecutions/1/stepexecutions/step1"}]}
```

□ View the messages.log file again. You should see println() output for every second it "slept."

Congratulations! You've setup and run your first batch job for this workshop.

End of Unit 2 Lab

© 2017, IBM Corporation

Unit 3 - JSR-352 Concepts

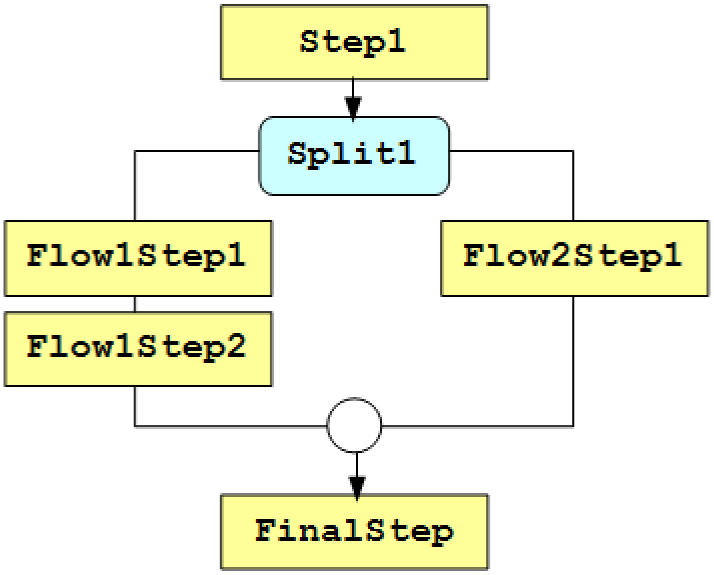
In this lab we'll take a closer look at how the JSR-352 specification works. Specifically: JSL flow control, and JSR "chunk" processing.

JSL orchestration: splits and flows

□ In your PuTTY session, copy the following application to the server's `/dropins` directory:

```
cp /u/user1/jbatch/apps/JSR352Concepts2ProjectWAR.war
       /shared/jsrhome/servers/JSRDISP/dropins/JSR352Concepts2ProjectWAR.war
```

This is a job that has a split and two flows and looks like this:



□ Take a look below at the Job Specification Language (JSL) for the job and compare to the picture above, noting the steps and split IDs and the "next=" labels:

```
<job ...>
   <step id="Step1" next="Split1">
      <batchlet ref="com.ibm.sample.LoggingBatchlet" />
   </step>
   <split id="Split1" next="FinalStep">
      <flow id="Flow1">
         <step id="Flow1Step1" next="Flow1Step2">
            <batchlet ref="com.ibm.sample.LoggingBatchlet" />
         </step>
         <step id="Flow1Step2">
            <batchlet ref="com.ibm.sample.LoggingBatchlet" />
         </step>
      </flow>
      <flow id="Flow2">
         <step id="Flow2Step1">
            <batchlet ref="com.ibm.sample.LoggingBatchlet" />
         </step>
      </flow>
   </split>
   <step id="FinalStep">
      <batchlet ref="com.ibm.sample.LoggingBatchlet" />
   </step>
</job>
```

© 2017, IBM Corporation

□ Note how every step has the same `ref=` value, which means each step runs the exact same Java class. All it does[4] is this:

```
log.log(Level.INFO, "Batchlet in control in step "+stepContext.getStepName());
```

That simply puts out a message indicating which step the processing is in.

□ In the PuTTY session, change directories to the Liberty /bin directory:

```
cd /shared/zWebSphere/Liberty/V17001/bin
```

□ While you're at it, make sure the JAVA_HOME is set:

```
export JAVA_HOME=/shared/java/J8.0_64/
```

□ Enter the following command *as one line* to run the job:

```
./batchManager submit --batchManager=localhost:10443
                                       --user=Fred --password=fredpwd
                           --applicationName=JSR352Concepts2ProjectWAR
                  --jobXMLName=SplitExample.xml --pollingInterval_s=5
                                        --trustSslCertificates --wait
```

It should come back relatively quickly with a status of 'COMPLETED'.

□ In the WS-FTP client, make sure the "Remote Site" location is:

```
/shared/jsrhome/servers/JSRDISP/logs
```

then highlight the `messages.log` file and click the "View" button. At the very bottom you should see:

```
I Batchlet in control in step Step1
I Batchlet in control in step Flow1Step1
I Batchlet in control in step Flow2Step1
I Batchlet in control in step Flow1Step2
I Batchlet in control in step FinalStep
```

Note: With a split, the flows operate on separate JVM threads "in parallel." So the sequence of messages you see may be slightly different from what's shown here.

***Lesson:** Our lesson here was **very** basic and **very** simple -- the Job Specification Language (JSL) file can control the "flow" through a multi-step job.*

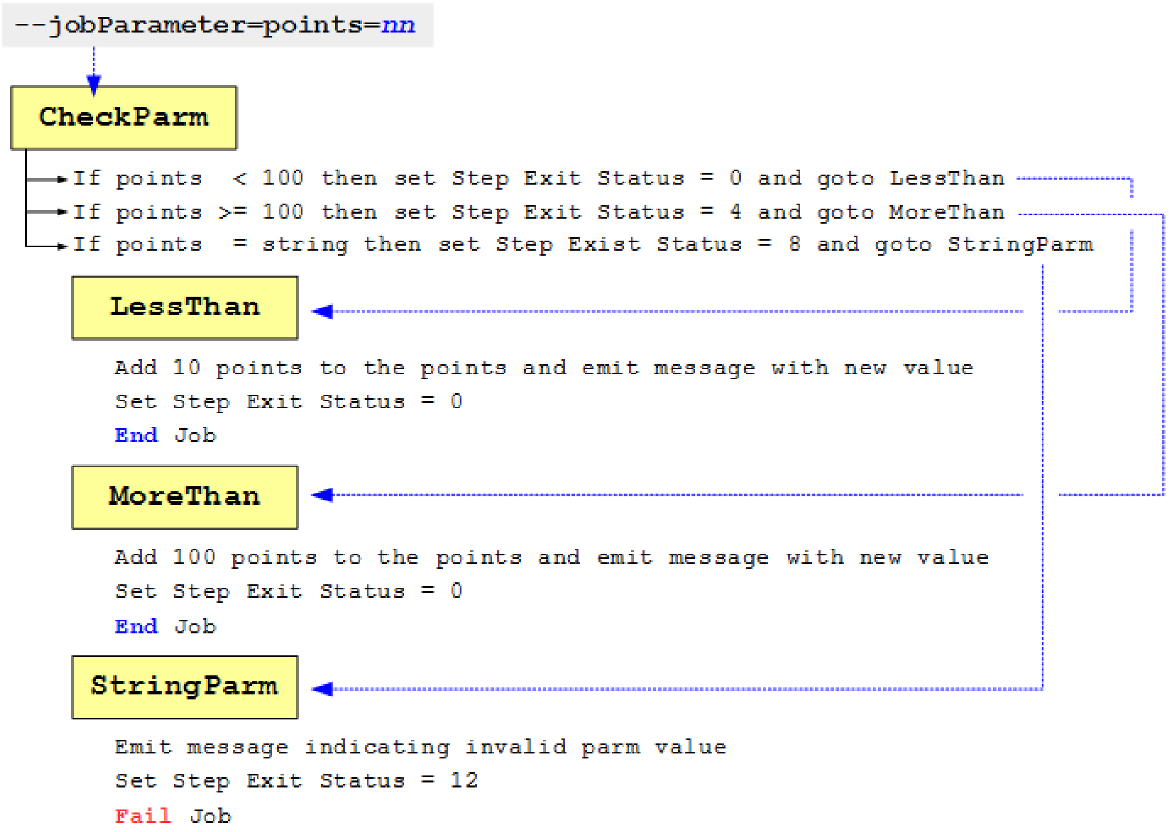
JSL orchestration: conditional flow control

□ In your PuTTY session, copy the following application to the server's `/dropins` directory:

```
cp /u/user1/jbatch/apps/JSR352ConceptsProjectWAR.war
        /shared/jsrhome/servers/JSRDISP/dropins/JSR352ConceptsProjectWAR.war
```

This is a job that has four steps, and it uses "conditional flow control" to determine what to do, based on the step exit status set by the first step. See next for an illustration:

[4] The Java source has a bit more than just that one line, of course.



```
--jobParameter=points=nn

CheckParm
  If points  < 100 then set Step Exit Status = 0 and goto LessThan
  If points >= 100 then set Step Exit Status = 4 and goto MoreThan
  If points  = string then set Step Exist Status = 8 and goto StringParm

LessThan
    Add 10 points to the points and emit message with new value
    Set Step Exit Status = 0
    End Job

MoreThan
    Add 100 points to the points and emit message with new value
    Set Step Exit Status = 0
    End Job

StringParm
    Emit message indicating invalid parm value
    Set Step Exit Status = 12
    Fail Job
```

- □ Take a look below at the Job Specification Language (JSL) for the job and compare to the picture above. Note in particular the "next on=" that follows first step:

```
<job ..>
   <step id="CheckParm">
      <batchlet ref="com.ibm.concepts.CheckParm">
         <properties >
            <property name="points" value="#{jobParameters['points']}?:0;"/>
         </properties>
      </batchlet>
      <next on="0" to="LessThan" />
      <next on="4" to="MoreThan" />
      <next on="8" to="StringParm" />
   </step>
   <step id="LessThan">
      <batchlet ref="com.ibm.concepts.LessThan" >
         <properties >
            <property name="points" value="#{jobParameters['points']}?:0;"/>
         </properties>
      </batchlet>
      <end on="*" exit-status="0"/>
   </step>
   <step id="MoreThan">
      <batchlet ref="com.ibm.concepts.MoreThan" >
         <properties >
            <property name="points" value="#{jobParameters['points']}?:0;"/>
         </properties>
      </batchlet>
      <end on="*" exit-status="0"/>
   </step>
   <step id="StringParm">
      <batchlet ref="com.ibm.concepts.StringParm" />
      <fail on="*" exit-status="12"/>
   </step>
</job>
```

□ Now take a look next at the relevant code in the CheckParm step:

```
      try {
            int i = (new Integer(points)).intValue();
            if (i<100) { rc = new String("0");}
            if (i>=100) { rc = new String("4");}
      }
      catch(NumberFormatException nfe) {
            rc = new String("8");
      }

      stepContext.setExitStatus(rc);
```

In plain language -- "Take the input string, turn it into an integer, then check for value. If less than 100, then set the step exit status to 0; if greater than or equal to 100 then set exit status to 4; if not an integer, then catch exception and set exit status to 8."

□ Enter the following command *as one line* to run the job with a "points" value of less than 100:

```
./batchManager submit --batchManager=localhost:10443
                                                          --user=Fred --password=fredpwd
                                              --applicationName=JSR352ConceptsProjectWAR
                               --jobXMLName=JSR352ConceptsJob.xml --trustSslCertificates
                                    --pollingInterval_s=5 --wait --jobParameter=points=50
```

In your PuTTY session output you should see the following for the job:

```
Batch status: COMPLETED. Exit status: 0
```

□ In the WS-FTP client, highlight the `messages.log` file and click the "View" button. At the very bottom you should see:

```
I Batchlet in control in step CheckParm
I Input parameter points=50
I Batchlet in control in step LessThan
I Input parameter points = 50
I Your point value gives you Gold status
I We gave you a bonus.  Your new point level is 60
```

□ Next, enter the following command *as one line* to run the job with a "points" value of more than 100:

```
./batchManager submit --batchManager=localhost:10443
                                                          --user=Fred --password=fredpwd
                                              --applicationName=JSR352ConceptsProjectWAR
                               --jobXMLName=JSR352ConceptsJob.xml --trustSslCertificates
                                --pollingInterval_s=5 --wait --jobParameter=points=5000
```

In your PuTTY session output you should see the following for the job:

```
Batch status: COMPLETED. Exit status: 0
```

□ In the WS-FTP client, highlight the `messages.log` file and click the "View" button. At the very bottom you should see:

```
I Batchlet in control in step CheckParm
I Input parameter points=5000
I Batchlet in control in step MoreThan
I Input parameter points = 5000
I Your point value gives you Platinum status
I We gave you a bonus.  Your new point level is 5100
```

□ Finally, try a command where the job parameter value is not an integer:

```
./batchManager submit --batchManager=localhost:10443
                                                  --user=Fred --password=fredpwd
                                      --applicationName=JSR352ConceptsProjectWAR
                   --jobXMLName=JSR352ConceptsJob.xml --trustSslCertificates
                      --pollingInterval_s=5 --wait --jobParameter=points=xyz
```

In your PuTTY session output you should see:

```
Batch status: FAILED . Exit status: 12
```

Note: Recall the JSL related to the "StringParm" step:
`<fail on="*" exit-status="12"/>`
That tells the container to fail the job and assign a job exit status of 12.

□ Go take a look at the `messages.log` file. You should see something like this:

```
I Batchlet in control in step CheckParm
I Input parameter points=xyz
I Batchlet in control in step StringParm
I Parameter value supplied is not an integer value.  This is not valid.
W CWWKY0011W: Job JSR352ConceptsJob failed with batch status FAILED and exit
status 12 for job instance nn and job execution nn.
```

***Lesson:** Our lesson here was the following:*

(a) You can set step exit status values

(b) You can control the flow based on conditional processing of step exit status

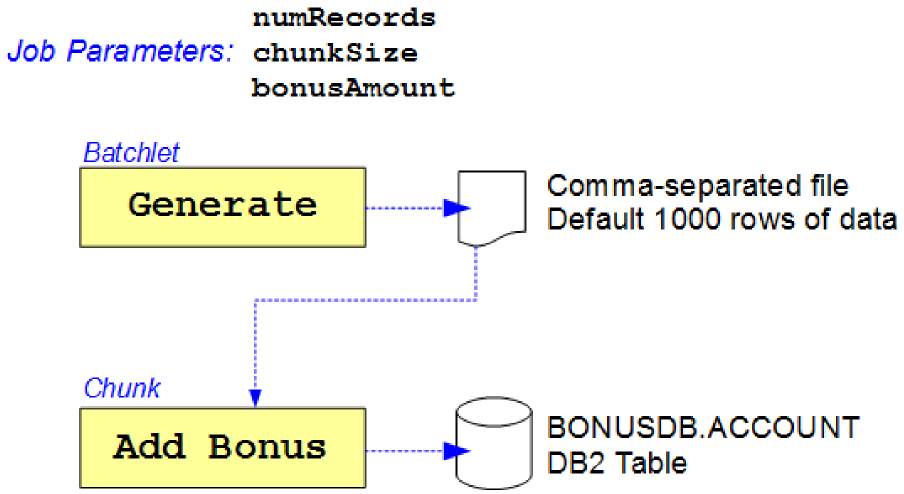
Chunk Processing: the modified BonusPayout sample

Note: We modified the BonusPayout sample available on Github by making the logging show the chunk activity more clearly. The sample available on Github is a bit silent by default. We changed it so more messages are put out, and the chunk activity can be seen more clearly.

□ In your PuTTY session, copy the following application to the server's `/dropins` directory:

```
cp /u/user1/jbatch/apps/JSR352Concepts3WAR.war
                   /shared/jsrhome/servers/JSRDISP/dropins/JSR352Concepts3WAR.war
```

This is a two-step job: the first is a batchlet which generates a file of numbers; the second is a chunk step that reads from the file, adds a "bonus" value to each record, and then writes each row to a DB2 table:



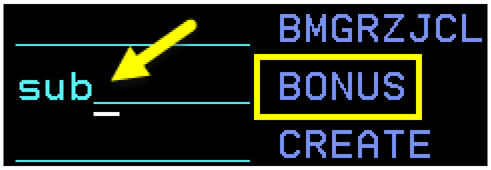
The Job Specification Language (JSL) file is a bit more involved than the others we showed, so we won't look at that here.

- □ Next, copy in a new `server.xml` file with the following command:

```
cp /u/user1/jbatch/lab3/jsrdisp.xml
                              /shared/jsrhome/servers/JSRDISP/server.xml
```

(That brings in some JDBC definitions so the `BONUSDB.ACCOUNT` table can be accessed.)

- □ In your 3270 session, go to **=3.4**, provide the data set **USER1.JBATCH.JCL** in the list field. Place a **b** (for browse) next to the data set when listed. Then type **sub** next to the **BONUS** member and press host enter:



This will create the `BONUSDB.ACCOUNT` table in DB2.

Press host enter to clear any three-star (***) screen holds and the job completion message.

- □ In the PuTTY session, enter the following command to submit the BonusPayout job:

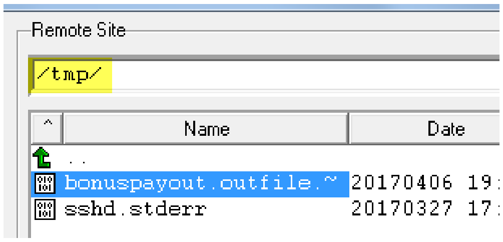
```
./batchManager submit --batchManager=localhost:10443
          --user=Fred --password=fredpwd --applicationName=JSR352Concepts3WAR
   --jobXMLName=LoggingSimpleBonusPayout.xml --jobParameter=dsJNDI=jdbc/bonus
      --jobParameter=tableName=BONUSDB.ACCOUNT --jobParameter=logging=true
                 --jobParameter=numRecords=10 --jobParameter=chunkSize=2
                   --pollingInterval_s=5 --trustSslCertificates --wait
```

Note `numRecords` and `chunkSize`. That's controlling the size of the generated data and the "chunk" (checkpoint) interval. This will "chunk" five times[5].

You should see: `Batch status: COMPLETED. Exit status: COMPLETED`

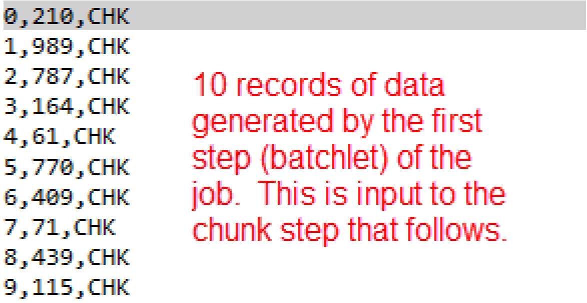
[5] Six, really -- it'll do five chunks based on the chunksize of 2, then it'll reach end of file and do a final chunk to close.

- ☐ In the WS-FTP client, type in **/tmp/** for the "Remote Site" value and press PC enter. You should see something like this:



```
..
bonuspayout.outfile.~   20170406 19:
sshd.stderr             20170327 17:
```

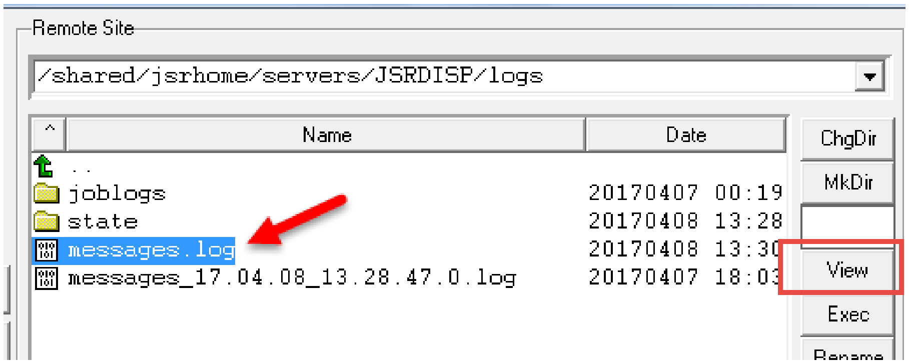
- ☐ Highlight the `bonuspayout.outfile.`*`nn`*`.csv` file[6] you see and click on the "View" button. That should open the text editor, and you should see *something* like this:



```
0,210,CHK
1,989,CHK
2,787,CHK
3,164,CHK
4,61,CHK
5,770,CHK
6,409,CHK
7,71,CHK
8,439,CHK
9,115,CHK
```

10 records of data generated by the first step (batchlet) of the job. This is input to the chunk step that follows.

- ☐ Go back to the WS-FTP client and put **/shared/jsrhome/servers/JSRDISP/logs** in the "remote site" field and press PC enter. Then highlight the messages.log file and click the "View" button:



```
..
joblogs                              20170407 00:19
state                                20170408 13:28
messages.log                         20170408 13:30
messages_17.04.08_13.28.47.0.log     20170407 18:03
```

- ☐ Scroll to the <u>bottom</u> of the `messages.log` file and take a moment to review what's there. It should look something like this. See the notes that follow this example output.

```
I Read Account 0
I Processing 0 Old Balance = 210 New Balance = 310
I Read Account 1
I Processing 1 Old Balance = 989 New Balance = 1089
I DSRA8203I: Database product name : DB2
I DSRA8204I: Database product version : DSN10015
I DSRA8205I: JDBC driver name  : IBM Data Server Driver for JDBC and SQLJ
I DSRA8206I: JDBC driver version  : 4.14.119
I Adding: 2 items to table name: BONUSDB.ACCOUNT via batch update
I Chunk Listener afterChunk
I Chunk Listener beforeChunk
I Read Account 2
I Processing 2 Old Balance = 787 New Balance = 887
I Read Account 3
```

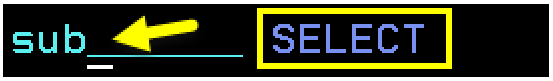
[6] The *nn* value will be the "Instance ID" of your batch job.

© 2017, IBM Corporation

```
I Processing 3 Old Balance = 164 New Balance = 264
I Adding: 2 items to table name: BONUSDB.ACCOUNT via batch update
I Chunk Listener afterChunk
I Chunk Listener beforeChunk
I Read Account 4
I Processing 4 Old Balance = 61 New Balance = 161
I Read Account 5
I Processing 5 Old Balance = 770 New Balance = 870
I Adding: 2 items to table name: BONUSDB.ACCOUNT via batch update
I Chunk Listener afterChunk
I Chunk Listener beforeChunk
I Read Account 6
I Processing 6 Old Balance = 409 New Balance = 509
I Read Account 7
I Processing 7 Old Balance = 71 New Balance = 171
I Adding: 2 items to table name: BONUSDB.ACCOUNT via batch update
I Chunk Listener afterChunk
I Chunk Listener beforeChunk
I Read Account 8
I Processing 8 Old Balance = 439 New Balance = 539
I Read Account 9
I Processing 9 Old Balance = 115 New Balance = 215
I Adding: 2 items to table name: BONUSDB.ACCOUNT via batch update
I Chunk Listener afterChunk
I Chunk Listener beforeChunk
I End of stream reached in class com.ibm.sample.GeneratedCSVReader
I Chunk Listener afterChunk
```

Note: The text highlighted in red represents a "chunk interval" processing. It read in the account record 4, and then it added 100 to the balance. Then it read in account record 5 and added 100 to the balance. Then it reached its "chunkSize" value and commited the transaction to DB2. It did this over and over until the ItemReader returned null (end of file) and it ended.

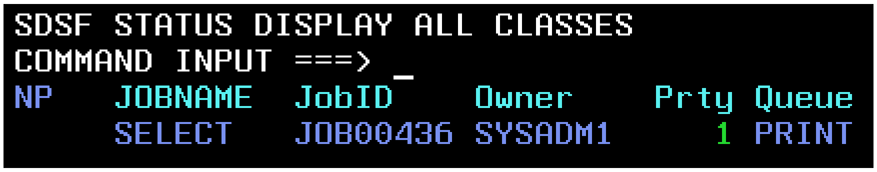
□ Go back to your 3270 session, go to **=3.4**, provide the data set **USER1.JBATCH.JCL** in the list field. Place a **b** (for browse) next to the data set when listed. Then type **sub** next to the **SELECT** member[7] and press host enter:



```
sub_          SELECT
```

Press host enter to clear any three-star (***) screen holds and the job completion message.

□ Let's go look at the output from that. Go to **=sdsf.st** and then set the prefix to **PRE SEL***. You should see:

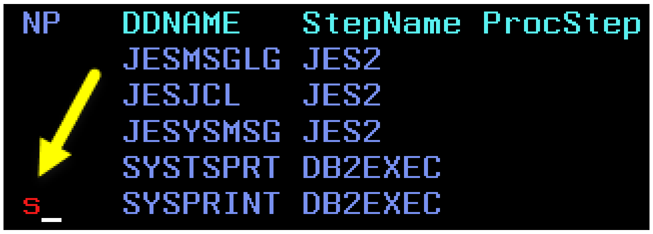


```
SDSF STATUS DISPLAY ALL CLASSES
COMMAND INPUT ===> _
NP   JOBNAME  JobID    Owner    Prty Queue
     SELECT   JOB00436 SYSADM1     1 PRINT
```

□ Place a question mark (**?**) next to the job and press host enter.

[7] This is simply JCL that issues a SELECT * FROM BONUSDB.ACCOUNT to list the contents of that table.

□ Now place an **s** next to the `SYSPRINT` member and press host enter:



```
NP     DDNAME   StepName ProcStep
       JESMSGLG JES2
       JESJCL   JES2
       JESYSMSG JES2
       SYSTSPRT DB2EXEC
s_     SYSPRINT DB2EXEC
```

You should see something like this:

```
   +--------------------------------------------------------+
   |  ACCTNUM   |  BALANCE   |    INSTANCEID     |  ACCTCODE  |
   +--------------------------------------------------------+
1_|          0 |        310 |                nn | CHK        |
2_|          1 |       1089 |                nn | CHK        |
3_|          2 |        887 |                nn | CHK        |
4_|          3 |        264 |                nn | CHK        |
5_|          4 |        161 |                nn | CHK        |
6_|          5 |        870 |                nn | CHK        |
7_|          6 |        509 |                nn | CHK        |
8_|          7 |        171 |                nn | CHK        |
9_|          8 |        539 |                nn | CHK        |
10_|          9 |        215 |                nn | CHK        |
   +--------------------------------------------------------+
```

Where *nn* is the instance number of the batch job that ran.

This is showing the contents fo the `BONUSDB.ACCOUNT` table, which is the DB2 table into which the chunk step of that job wrote its output. In this case it did a COMMIT every two records because that's what you asked for with `chunkSize=2` on the batchManager command.

□ Go back to your PuTTY session.

□ Let's run the job again, this time with 1000 record, a chunk size of 100, and logging turned off so the output is not so chatty:

```
./batchManager submit --batchManager=localhost:10443
  --user=Fred --password=fredpwd --applicationName=JSR352Concepts3WAR
                                  --jobXMLName=LoggingSimpleBonusPayout.xml
                                           --jobParameter=dsJNDI=jdbc/bonus
 --jobParameter=tableName=BONUSDB.ACCOUNT --jobParameter=logging=false
           --jobParameter=numRecords=1000 --jobParameter=chunkSize=100
                        --pollingInterval_s=5 --trustSslCertificates --wait
```

Note the parameters highlighted in red: we are turning logging off, we set the number of records to 1000, and we set the chunk interval to 100.

You should see: `Batch status: COMPLETED. Exit status: COMPLETED`

□ In the WS-FTP client, go to **/tmp/** and click the "Refresh" button. You should now see two "csv" files. Highlight the second "csv" file you see there. Click the "View" button. This time you'll see many more records ... 1000 to be precise. Scroll to the bottom and the last record should be 999.

□ In 3270, go to the **USER1.JBATCH.JCL** data set and submit the **SELECT** member again.

- □ Go to **=sdsf.st** with the prefix of **PRE SEL*** and you should see two SELECT jobs. Place a question mark (?) next to the one with the higher JobID value and press host enter. Then place an s next to the SYSPRINT member and press host enter.

  At the top you should see the records created by the first running of the job. But if you scroll down (F8) you should see the records for the second running (keep an eye on the INSTANCEID column).

- □ Clean up the applications you used in this lab. In the PuTTY session, enter the following command:

```
rm /shared/jsrhome/servers/JSRDISP/dropins/JSR352*
```

- □ Verify that only SleepyBatchlet remains:

```
ls /shared/jsrhome/servers/JSRDISP/dropins
```

Lesson: *Our lesson here was the following:*

(a) Experience a "chunk" application

(b) See evidence of the "chunk" process taking place

End of Unit 3 Lab

© 2017, IBM Corporation

Unit 4 - Job Submission and Control

batchManager

You used this briefly back in Lab 2 when you validated your environment. We'll explore a bit more of batchManager here.

□ Go to your PuTTY session[8] and verify a few things:

○ Enter command **`whoami`**

It should respond with: `JSRSERV`

If it returns `USER1`, then enter command **`su -s JSRSERV`**

○ Enter command **`pwd`**

You should be in `/shared/zWebSphere/Liberty/V17001/bin`

If not, then enter **`cd /shared/zWebSphere/Liberty/V17001/bin`**

○ Enter command **`echo $WLP_USER_DIR`**

It should respond with `/shared/jsrhome`

If not, then enter command **`export WLP_USER_DIR=/shared/jsrhome`**

○ Finally, enter command **`echo $JAVA_HOME`**

It should respond with export `/shared/java/J8.0_64/`

If not, then enter command **`export JAVA_HOME=/shared/java/J8.0_64/`**

□ Submit the SleepyBatchlet sample again with the following command entered as *one line*:

```
./batchManager submit --batchManager=localhost:10443 --user=Fred
                      --password=fredpwd --applicationName=SleepyBatchletSample-1.0
                     --jobXMLName=sleepy-batchlet.xml --trustSslCertificates --wait
```

This should result in the job being submitted and running to a 'COMPLETED' state. If you look closely at the output, you'll see the "instance ID" and "execution ID" both incremented by one from the previous execution.

□ The default "sleep time" of that application is 15 seconds, but that can be altered by passing in a job parameter. Enter the following as one line:

```
./batchManager submit --batchManager=localhost:10443 --user=Fred
                      --password=fredpwd --applicationName=SleepyBatchletSample-1.0
                --jobXMLName=sleepy-batchlet.xml --jobParameter=sleep.time.seconds=5
                                                 --trustSslCertificates --wait
```

Notice the extra value: `--jobParameter=sleep.time.seconds=5`. That tells the application to "sleep" for only 5 seconds.

You should see the job status as 'COMPLETED'.

□ Let's verify the passed-in job parameter actually took effect. Using the WS-FTP client, navigate to the `/shared/jsrhome/servers/JSRDISP/logs` directory and view the `messages.log` file. You should see:

```
SleepyBatchlet: process: entry
SleepyBatchlet: process: sleep for: 5
SleepyBatchlet: process: [0] sleeping for a second...
SleepyBatchlet: process: [1] sleeping for a second...
```

[8] If it's closed, then double click on the "PuTTY" icon and load the WG31 profile. Log in as "USER1."

```
SleepyBatchlet: process: [2] sleeping for a second...
SleepyBatchlet: process: [3] sleeping for a second...
SleepyBatchlet: process: [4] sleeping for a second...
SleepyBatchlet: process: exit. exitStatus: SleepyBatchlet:i=5;stopRequested=false
```

That verifies the passed-in value took effect.

□ Next you'll add one more parameter: `--getJobLog`. That will result in the job log coming back to your PuTTY session. Enter the following as one line:

```
./batchManager submit --batchManager=localhost:10443 --user=Fred
               --password=fredpwd --applicationName=SleepyBatchletSample-1.0
       --jobXMLName=sleepy-batchlet.xml --jobParameter=sleep.time.seconds=5
                                  --getJobLog --trustSslCertificates --wait
```

When that completes you will see the job log come back to the PuTTY session. It will look *something*[9] like this:

```
xxxxx Begin file: instance.4/execution.4/part.1.log xxxxxxxxxxxxxxxxxx
[3/30/17 14:33:36:349 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Started invoking execution for a job
 JobInstance id = 4
 JobExecution id = 4
 Job Name = sleepy-batchlet
 Job Parameters = {sleep.time.seconds=5}
==========================================================

[3/30/17 14:33:36:368 GMT] com.ibm.ws.batch.JobLogger
[3/30/17 14:33:36:547 GMT] com.ibm.ws.batch.JobLogger
==========================================================
For step name = step1
 New top-level step execution id = 4
==========================================================

[3/30/17 14:33:36:787 GMT] com.ibm.ws.batch.JobLogger
[3/30/17 14:33:42:011 GMT] com.ibm.ws.batch.JobLogger
[3/30/17 14:33:42:026 GMT] com.ibm.ws.batch.JobLogger
[3/30/17 14:33:42:026 GMT] com.ibm.ws.batch.JobLogger
==========================================================
Completed invoking execution for a job
 JobInstance id = 4
 JobExecution id = 4
 Job Name = sleepy-batchlet
 Job Parameters = {sleep.time.seconds=5}
 Job Batch Status = COMPLETED, Job Exit Status = COMPLETED
==========================================================



xxxxx End file: instance.4/execution.4/part.1.log xxxxxxxxxxxxxxxxxx
```

□ Now let's run that exact same command again, but this time do it from a shell script.

Why? Because this helps illlustrate how you might programmatically invoke a batch job on z/OS from a scheduler function running *off-platform*. Or from another z/OS LPAR.

In your PuTTY session, enter command **`cd /u/user1/jbatch/lab4`**

[9] There's a few lines of text to the right we're omitting here. The lines are long and we wanted to avoid wrapping text here.

□ Let's see what's in the shell script we developed for this workshop. Enter the command: **`cat subjob.sh`**

> **Note:** That shell script has four sections:
>
> - Set variables used to construct the command
> - Set JAVA_HOME so batchManager can operate
> - Construct and submit the command
> - Capture the batchManager return code and format into something more compatible with the shift BPXBATCH will perform[10].

□ Submit the job with this command: **`./subjob.sh`**

You should see:

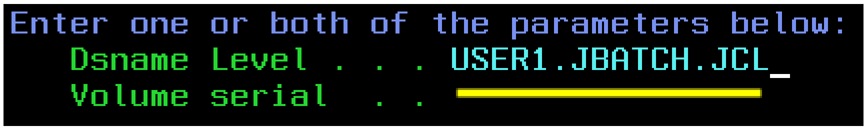
```
$ ./subjob.sh
Setting Variables
Set Java Home
Submitting Command
batchManager exit code= 35 Status=Good: submitted and completed.
Will set shell script exit code to 0
If BPXBATCH, then JES return code should be RC=0000
```

> **Note:** Return code 35 is what batchManager issues when the job has successfully been submitted and has completed. So in this context, view RC=35 as good news. The shell script sees the batchManager 35 and sets the shell script exit code to 0.

□ Check the output files:

STDOUT	**`cat out.txt`** You should see the joblog.
STDERR	**`cat err.txt`** You should nothing back as a successful execution results in nothing to STDERR.

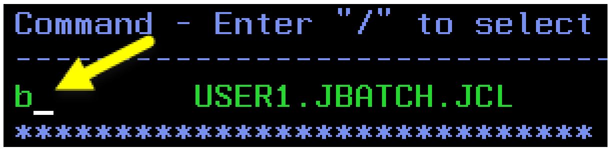
□ Let's now run that shell script using JCL and BPXBATCH. In your 3270 session, go to **`=3.4`** and set the data set name to **`USER1.JBATCH.JCL`**:



```
Enter one or both of the parameters below:
   Dsname Level . . . USER1.JBATCH.JCL_
   Volume serial  . .
```

Press the host enter key.

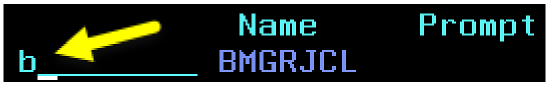
□ Place an **b** (for browse) next to the data set:



```
Command - Enter "/" to select
-------------------------------
b_        USER1.JBATCH.JCL
******************************
```

Press host enter.

□ Place **b** next to the `BMGRJCL` member and press host enter:



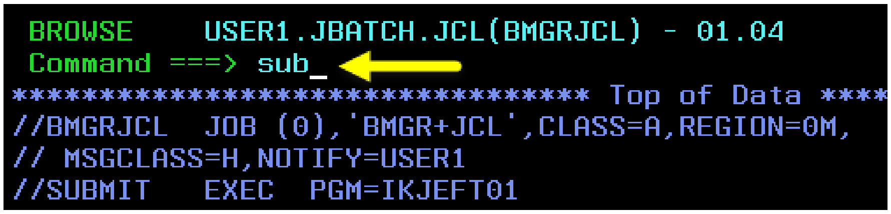
```
              Name      Prompt
b_           BMGRJCL
```

[10] The objective is to set the shell script exit code to something under 16. batchManager exit codes are above 16. Numbers above 16 are shifted by BPXBATCH, then JES will capture a *portion* of that. It gets confusing. But for numbers under 16 then it's more understandable and deterministic.

You should see a relatively simple JCL that uses `BPXBATCH` to execute the shell script:

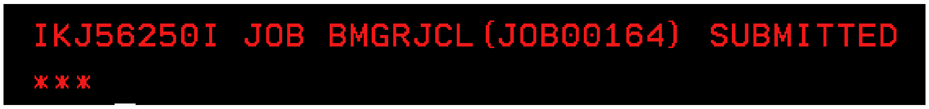
```
//BMGRJCL  JOB (0),'BMGR+JCL',CLASS=A,REGION=0M,
// MSGCLASS=H,NOTIFY=USER1
//SUBMIT   EXEC  PGM=IKJEFT01
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSTSIN  DD  *
 BPXBATCH  SH +
 /u/user1/jbatch/lab4/subjob.sh
/*
//*
```

□ Submit that job by typing **sub** in the command field and press host enter:



```
 BROWSE    USER1.JBATCH.JCL(BMGRJCL) - 01.04
 Command ===> sub_
********************************** Top of Data ****
//BMGRJCL  JOB (0),'BMGR+JCL',CLASS=A,REGION=0M,
// MSGCLASS=H,NOTIFY=USER1
//SUBMIT   EXEC  PGM=IKJEFT01
```

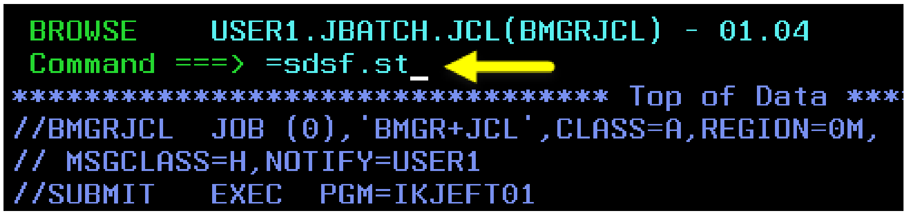
□ You will get three asterisks on your screen (***). Press host enter to clear that. You should then see:



```
 IKJ56250I JOB BMGRJCL(JOB00164) SUBMITTED
 ***
```

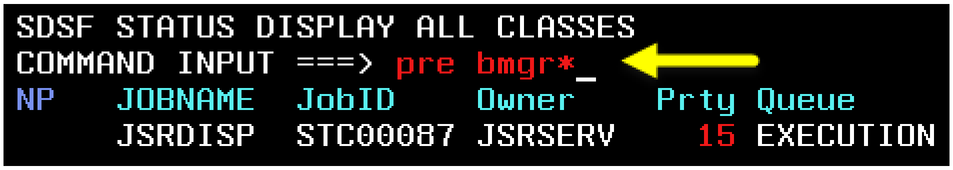
Again, press host enter to clear that.

□ In the command field, enter **=sdsf.st** to go to the job status display:



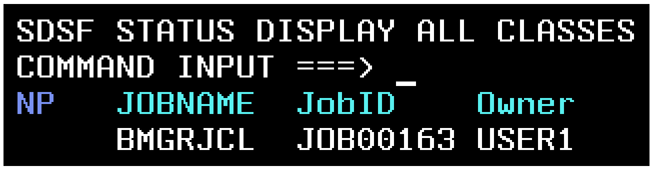
```
 BROWSE    USER1.JBATCH.JCL(BMGRJCL) - 01.04
 Command ===> =sdsf.st_
********************************** Top of Data ***
//BMGRJCL  JOB (0),'BMGR+JCL',CLASS=A,REGION=0M,
// MSGCLASS=H,NOTIFY=USER1
//SUBMIT   EXEC  PGM=IKJEFT01
```

□ The "JSR*" prefix from earlier is still set, so you won't see your BMGRJCL job until you change the prefix. Enter **pre bmgr*** in the command field:



```
SDSF STATUS DISPLAY ALL CLASSES
COMMAND INPUT ===> pre bmgr*_
NP   JOBNAME  JobID    Owner    Prty Queue
     JSRDISP  STC00087 JSRSERV    15 EXECUTION
```

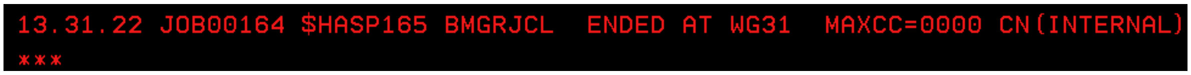
You should then see your job running:



```
SDSF STATUS DISPLAY ALL CLASSES
COMMAND INPUT ===> _
NP   JOBNAME  JobID    Owner
     BMGRJCL  JOB00163 USER1
```

□ Wait 10 or 15 seconds, then press host enter again. You should see[11]:

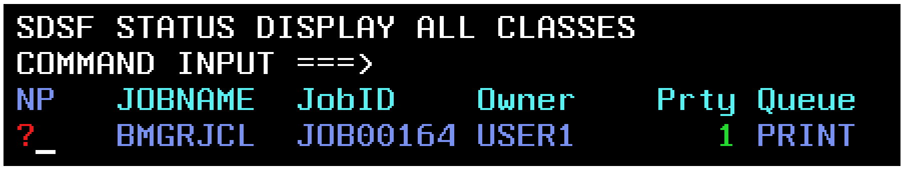
[11] If you don't see the completion message, wait a bit more and press host enter again.



```
13.31.22 JOB00164 $HASP165 BMGRJCL  ENDED AT WG31  MAXCC=0000 CN(INTERNAL)
***
```

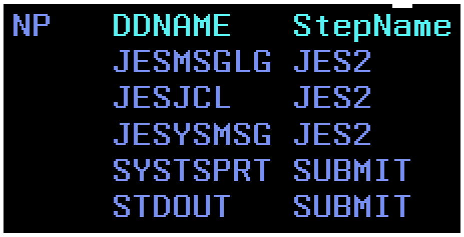
Note the "MAXCC" value. That's your return code. Press host enter to clear this.

□ Now put a question mark (**?**) next to the completed job and press host enter:



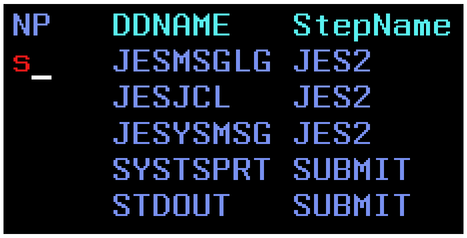
```
SDSF STATUS DISPLAY ALL CLASSES
COMMAND INPUT ===>
NP   JOBNAME  JobID    Owner    Prty Queue
?_   BMGRJCL  JOB00164 USER1       1 PRINT
```

That should bring up all the DDNAME members for the job:



```
NP   DDNAME   StepName
     JESMSGLG JES2
     JESJCL   JES2
     JESYSMSG JES2
     SYSTSPRT SUBMIT
     STDOUT   SUBMIT
```

□ Place an **s** (for "select") next to the `JESMSGLG` member and press host enter:



```
NP   DDNAME   StepName
s_   JESMSGLG JES2
     JESJCL   JES2
     JESYSMSG JES2
     SYSTSPRT SUBMIT
     STDOUT   SUBMIT
```

You should see something like the following. Note the RC value. That RC is based on the exit code the shell script set. When you're through with this, press F3.

```
  :
13.30.47 JOB00164  IRR010I  USERID USER1    IS ASSIGNED TO THIS JOB.
13.30.47 JOB00164  ICH70001I USER1    LAST ACCESS AT 13:29:08 ON FRIDAY,
MARCH 31, 2017
13.30.47 JOB00164  $HASP373 BMGRJCL  STARTED - INIT 1 - CLASS A - SYS WG31
13.31.22 JOB00164  Jobname  Procstep Stepname  CPU Time      EXCPs     RC
13.31.22 JOB00164  BMGRJCL  --None-- SUBMIT    00:00:00         51     00
13.31.22 JOB00164  $HASP395 BMGRJCL  ENDED
  :
```

□ Place an **s** next to the `STDOUT` member and press host enter. You should see something like this:

```
Setting Variables
Set Java Home
Submitting Command
batchManager exit code= 35 Status=Good: submitted and completed.
Will set shell script exit code to 0
If BPXBATCH, then JES return code should be RC=0000
```

We saw the JES return code equal 0 (it was listed as `00`, but that's the same thing).

But what about the Java Batch job log? Why isn't that here?

That's because the shell script *was redirecting that to a file*:

```
$LOCATION $ACTION $SERVER $USER $PW $APP $JOBXML $PARM1 $PARM2
                   $PARM3 $PARM4 > $OUTLOC/out.txt 2> $OUTLOC/err.txt
```

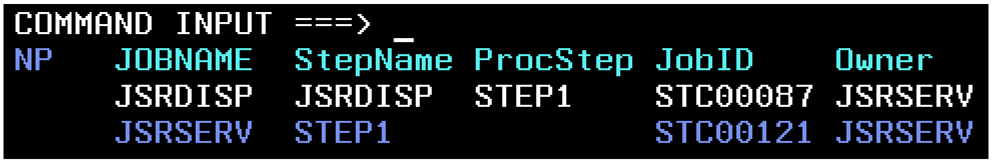
If you wanted the job log to go to JES with the `--getJobLog` parameter specified, you would simply remove that redirect to file from the shell script. It would then go to `STDOUT DD`.

We'll do that with batchManagerZos coming up next.

batchManagerZos

The batchManagerZos command line utility is very similar to batchManager, but it uses WOLA to communicate with Liberty z/OS rather than REST. Using WOLA requires some setup work.

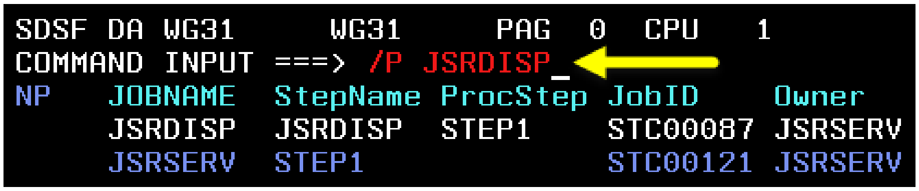
□ In your 3270 session, go to **=sdsf.da**, set the prefix to **pre JSR***. You should see your server running:



```
COMMAND INPUT ===> _
NP    JOBNAME   StepName ProcStep JobID    Owner
      JSRDISP   JSRDISP  STEP1    STC00087 JSRSERV
      JSRSERV   STEP1             STC00121 JSRSERV
```

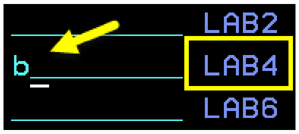
(Note: the other "JSRSERV" is your PuTTY session.)

□ Stop your server with the following command: **/P JSRDISP**



```
SDSF DA WG31      WG31      PAG  0  CPU   1
COMMAND INPUT ===> /P JSRDISP_
NP    JOBNAME   StepName ProcStep JobID    Owner
      JSRDISP   JSRDISP  STEP1    STC00087 JSRSERV
      JSRSERV   STEP1             STC00121 JSRSERV
```

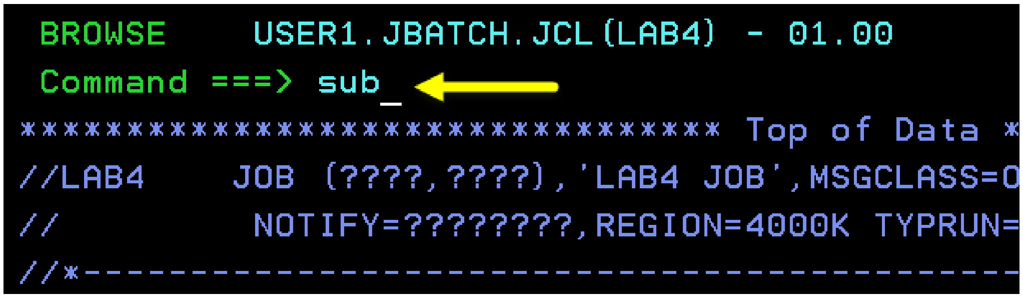
□ Go to **=3.4**, list the data set **USER1.JBATCH.JCL**, then place a **b** next to the `LAB4` member:



```
                    LAB2
b_                  LAB4
                    LAB6
```

That job creates a set of `SERVER` and `CBIND` profiles that allow WOLA to work for the server. We'll cover this in a bit more detail in Unit 6 on security.

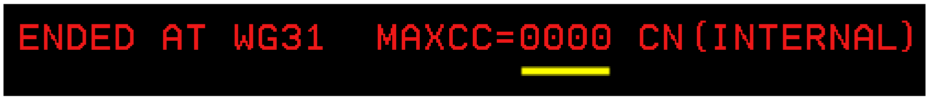
□ Submit that job by typing **sub** in the command line for the member and pressing host enter:



```
 BROWSE    USER1.JBATCH.JCL(LAB4) - 01.00
 Command ===> sub_
********************************* Top of Data *
//LAB4    JOB (????,????),'LAB4 JOB',MSGCLASS=0
//         NOTIFY=????????,REGION=4000K TYPRUN=
//*---------------------------------------------
```

If you get the "three star" screen hold, press host enter to clear that.

□ You should fairly quickly see:



```
ENDED AT WG31  MAXCC=0000 CN(INTERNAL)
```

□ In your PuTTY session, copy over a new `server.xml` so the WOLA support is present. Use the following command:

cp /u/user1/jbatch/lab4/wola.xml /shared/jsrhome/servers/JSRDISP/server.xml

That added the following XML elements to the previous `server.xml`:

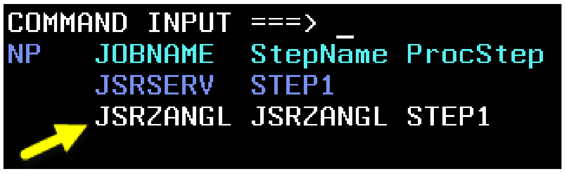
```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>zosLocalAdapters-1.0</feature>
    </featureManager>

    <zosLocalAdapters wolaGroup="LIBERTY"
        wolaName2="BATCH"
        wolaName3="MANAGER"/>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <basicRegistry id="basic1" realm="jbatch">
      <user name="Fred" password="fredpwd" />
    </basicRegistry>
    (The rest of the server.xml omitted to save space in document)
```

□ Now go to your 3270 session. Enter **=sdsf.da** and make sure the prefix is **PRE JSR*** (it most likely is still that value from before).

□ Now you're going to start the Liberty Angel, which allows access to authorized servers. Issue the following command:

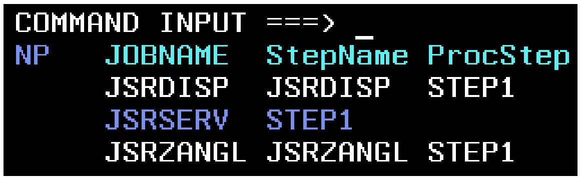
/S JSRZANGL

We copied that JCL to proclib and customized prior to the workshop. You sould see the following:



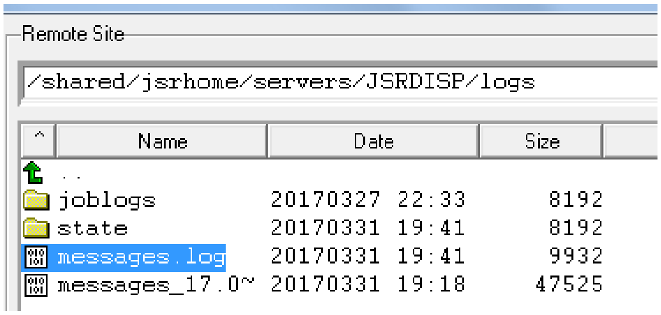
```
COMMAND INPUT ===> _
NP    JOBNAME   StepName ProcStep
      JSRSERV   STEP1
      JSRZANGL JSRZANGL STEP1
```

□ Now start your server again with **/S JSRDISP**. You should now see:



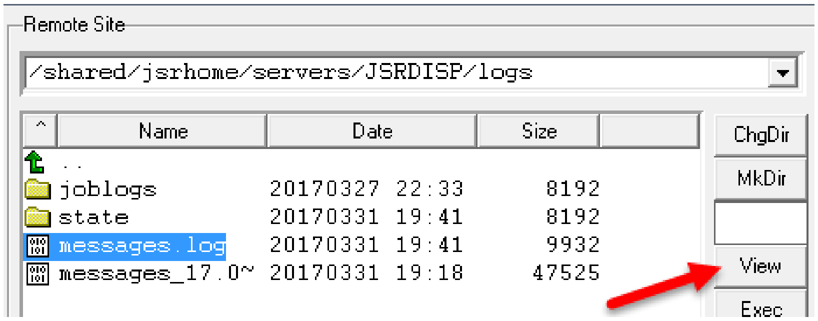
```
COMMAND INPUT ===> _
NP    JOBNAME   StepName ProcStep
      JSRDISP   JSRDISP   STEP1
      JSRSERV   STEP1
      JSRZANGL JSRZANGL STEP1
```

□ Go to the WS-FTP client and navigate to the "remote" directory where the `messages.log` file is located: **/shared/jsrhome/servers/JSRDISP/logs**. You should see:



```
Remote Site
/shared/jsrhome/servers/JSRDISP/logs

   Name              Date             Size
 ..
 joblogs           20170327 22:33     8192
 state             20170331 19:41     8192
 messages.log      20170331 19:41     9932
 messages_17.0~    20170331 19:18    47525
```

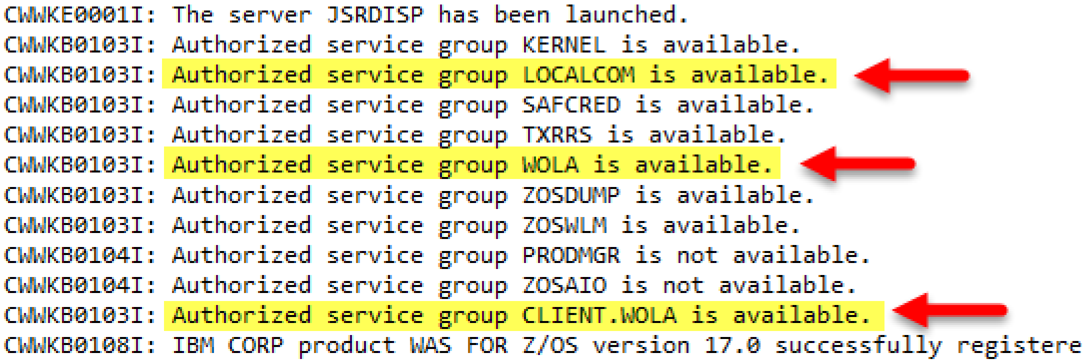
□ Highlight the `messages.log` file and click the "View" button:



```
/shared/jsrhome/servers/JSRDISP/logs

Name            Date             Size
..
joblogs         20170327 22:33    8192
state           20170331 19:41    8192
messages.log    20170331 19:41    9932
messages_17.0~  20170331 19:18   47525
```

This will open up a text editor and display the log file.

□ Look for the key "is available" messages at the top of the log file:



```
CWWKE0001I: The server JSRDISP has been launched.
CWWKB0103I: Authorized service group KERNEL is available.
CWWKB0103I: Authorized service group LOCALCOM is available.
CWWKB0103I: Authorized service group SAFCRED is available.
CWWKB0103I: Authorized service group TXRRS is available.
CWWKB0103I: Authorized service group WOLA is available.
CWWKB0103I: Authorized service group ZOSDUMP is available.
CWWKB0103I: Authorized service group ZOSWLM is available.
CWWKB0104I: Authorized service group PRODMGR is not available.
CWWKB0104I: Authorized service group ZOSAIO is not available.
CWWKB0103I: Authorized service group CLIENT.WOLA is available.
CWWKB0108I: IBM CORP product WAS FOR Z/OS version 17.0 successfully registere
```

Those indicate: (a) The Angel is up, and (b) the server ID has access to the required SAF SERVER profiles to allow access to the WOLA authorized service.

□ About half-way down the log file you should see this message:

```
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered
with the Liberty profile server using the following name: LIBERTY
BATCH MANAGER
```

That indicates the WOLA "three part name" is in effect for this Liberty z/OS server.

□ Go to your PuTTY session and issue the command **`whoami`**. It should tell you the ID in effect is `JSRSERV`. That ID was *not* granted `READ` to the `CBIND`, so that ID can't use batchManagerZos. But the `USER1` ID was granted `READ`, so it can use it.

□ Enter the command **`exit`**. This will take you *out* of `JSRSERV` and return you to the **`USER1`** ID. You can verify this **`whoami`**.

□ In your PuTTY session, change directories using this command:

```
cd /shared/zWebSphere/Liberty/V17001/lib/native/zos/s390x
```

□ Let's start with the simple "ping" command:

```
./batchManagerZos ping --batchManager=LIBERTY+BATCH+MANAGER
```

That returns *nothing* if everything is working well. If something is not right, you'll see a bunch of error messages[12].

[12] Assuming you saw the "is available" messages, then the most common problem here is either (a) the ID you're using does not have READ to the CBIND, or (b) the three-part name is not *exactly* matched. That name value is case-sensitive.

© 2017, IBM Corporation

□ Submit a SleepyBatchlet job with the following command *as one line*:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                            --applicationName=SleepyBatchletSample-1.0
                                    --jobXMLName=sleepy-batchlet.xml --wait
```

You will see the same output on your PuTTY session screen as you did with batchManager.

□ You can run batchManagerZos from JCL using BPXBATCH just like you did batchManager. Go to **=3.4**, list **USER1.JBATCH.JCL**, then browse the **BMGRZJCL** member. You should see:

```
//BMGRZJCL JOB (0),'BMGRZ+JCL',CLASS=A,REGION=0M,
// MSGCLASS=H,NOTIFY=USER1
//SUBMIT   EXEC  PGM=IKJEFT01
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSTSIN  DD  *
 BPXBATCH  SH +
 /u/user1/jbatch/lab4/subzjob.sh
/*
//*
```

Two differences from before: (1) the job name is different; (2) the shell script it calls is different.

Note: The shell script is very similar to the one used with batchManager, with the following differences: (a) the location in the file system for batchManagerZos is different from batchManager; (b) batchManagerZos does not need host/port and ID/PW, but it does need the three-part name.

□ Submit that job by entering **sub** in the command field.

□ Go to **=sdsf.st** and set **PRE BMGR***. You should see your job executing.

□ When that completes, place an **S** next to the job and press host enter. You should see near the top:

```
Jobname  Procstep Stepname  CPU Time       EXCPs     RC
BMGRZJCL --None-- SUBMIT    00:00:00          53     00
$HASP395 BMGRZJCL ENDED
```

and at the bottom:

```
Setting Variables
Submitting Command
batchManagerZos exit code= 35 Status=Good: submitted and completed.
Will set shell script exit code to 0
If BPXBATCH, then JES return code should be RC=0000
```

Job Log? There is a `--getJobLog` parameter on batchManagerZos, but we did not use it here. Why not? Because batchManagerZos requires "batch events" to be configured to retrieve the job log, and we've not yet configured that.

How did batchManager do it? By calling back to the server with the REST interface to retrieve the job log. It can do that easily because batchManager is a Java program, and Java can fairly easily format an HTTP request and parse the JSON that comes back. A native z/OS program like batchManagerZos would struggle with HTTP and JSON parsing.

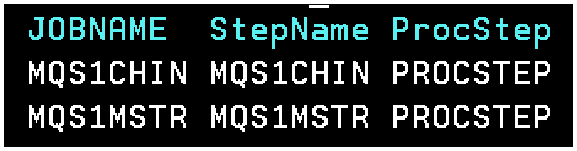
We'll see retrieval of the job log when we get to batch events.

Batch Events

In this lab we will make use of the "batch events" feature to allow batchManagerZos to monitor for job completion and retrieve the job log. We will also use the JSRMON server to host another application that will monitor the batch events tree for events that are published.

□ In your 3270 session, go to **`=sdsf.da`** and set the prefix with **`PRE MQS1*`**.

□ Start MQ with **`/-MQS1 START QMGR`**. Press host enter to refresh the screen. You should soon see:



```
JOBNAME  StepName ProcStep
MQS1CHIN MQS1CHIN PROCSTEP
MQS1MSTR MQS1MSTR PROCSTEP
```

□ Next, copy the MQ JMS resource adapter to the JSRDISP directory. In your PuTTY session, use this command *as one line*:

```
cp /shared/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar
                        /shared/jsrhome/servers/JSRDISP/wmq.jmsra.rar
```

□ Next, make the MQ "SCSQUATH" data set[13] available on the `STEPLIB` variable to your PuTTY session: **`export STEPLIB='MQ901.SCSQAUTH'`**

□ Copy the XML updates so your JSRDISP server is enabled to emit batch events:

```
cp /u/user1/jbatch/lab4/events.xml
                           /shared/jsrhome/servers/JSRDISP/server.xml
```

Here's what was added:

```
<!-- Enable features -->
<featureManager>
    <feature>servlet-3.1</feature>
    <feature>batch-1.0</feature>
    <feature>batchManagement-1.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>wmqJmsClient-2.0</feature>
</featureManager>

<batchJmsEvents connectionFactoryRef="batchConnectionFactory" />

<jmsConnectionFactory id="batchConnectionFactory"
  jndiName="jms/batch/connectionFactory">
  <properties.wmqJms
    hostName="wg31.washington.ibm.com"
    transportType="CLIENT"
    channel="SYSTEM.DEF.SVRCONN"
    port="1414"
    queueManager="MQS1">
  </properties.wmqJms>
</jmsConnectionFactory>

<variable name="wmqJmsClient.rar.location"
  value="${server.config.dir}/wmq.jmsra.rar" />

<wmqJmsClient startupRetryCount="999"
```

[13] The batchManagerZos client will use MQ BINDINGS mode to monitor the pub/sub topic tree.

```
      startupRetryInterval="1000ms"
      reconnectionRetryCount="10"
      reconnectionRetryInterval="5m">
    </wmqJmsClient>

    <zosLocalAdapters wolaGroup="LIBERTY"
        wolaName2="BATCH"
        wolaName3="MANAGER"/>
```

(That's the XML necessary to enable batch events and make the server able to connect to the MQ Queue Manager.)

□ The server will have dynamically updated its configuration with those changes, so now you're ready to submit a batchManagerZos command. First, let's make certain your'e in the right directory:

```
cd /shared/zWebSphere/Liberty/V17001/lib/native/zos/s390x
```

□ Now, enter the following command:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                              --applicationName=SleepyBatchletSample-1.0
      --jobXMLName=sleepy-batchlet.xml --wait --queueManagerName=MQS1
```

Notice that last parameter, which names the MQ queue manager to connect to and listen on the default topic root of `batch` for the job completion event.

You should see this message:

```
INFO: CWWKY0118I: Subscribing to batch events at JMS topic root:
batch/jobs/instance/+.
```

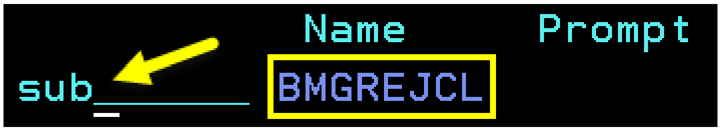
Then you should see the usual messages indicating the job has been submitted and has gone to completion. You will find the job seems to complete more quickly. In reality what's happening is batchManagerZos is aware of the job completing the moment the completion event is published. It did not have to wait for the next polling cycle to find out.

□ Let's get the job log via the batch events topic tree as well:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
 --applicationName=SleepyBatchletSample-1.0 --jobXMLName=sleepy-batchlet.xml
                              --getJobLog --wait --queueManagerName=MQS1
```

As part of the output to the PuTTY terminal you should see the job log.

□ Finally, let's invoke that via JCL and a shell script. In your 3270 session, go to the **USER1.JBATCH.JCL** data set and submit the **BMGREJCL** ("e" for "events") member:

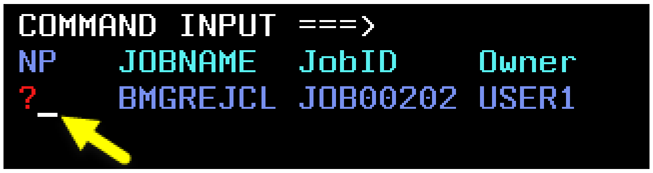


```
            Name      Prompt
sub_      BMGREJCL
```

That runs the `subzjob_events.sh` shell script. That shell script calls batchManagerZos, and the command includes `--getJobLog`, `--wait`, and `--queueManagerName`.

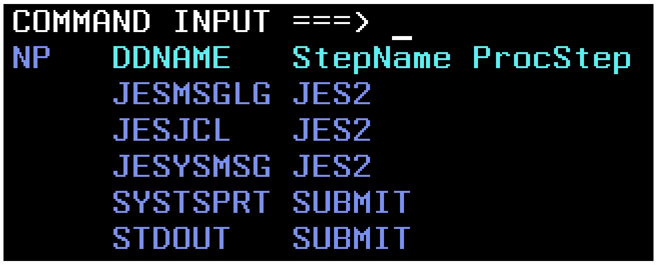
□ Go to **=sdsf.st** to view your job. The prefix should still be set to **PRE BMGR***, so your job should appear.

□ When your job completes, place a question mark (**?**) next to the job name and press host enter:



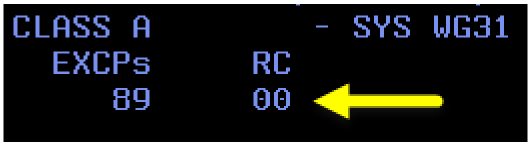
```
COMMAND INPUT ===>
NP    JOBNAME  JobID    Owner
?_    BMGREJCL JOB00202 USER1
```

That will display the `DDNAME` members:



```
COMMAND INPUT ===> _
NP    DDNAME   StepName ProcStep
      JESMSGLG JES2
      JESJCL   JES2
      JESYSMSG JES2
      SYSTSPRT SUBMIT
      STDOUT   SUBMIT
```

□ Place an **s** next to the `JESMSGLG` member and press host enter. You should see the return code:



```
CLASS A            - SYS WG31
  EXCPs       RC
     89       00
```

□ Press **F3** to return to the list of `DDNAME` members, then place an **s** next to the `STDOUT` member and press host enter. You should see the job log (which came back to the JCL job via the batch events mechanism).

At the *bottom* you should see the messages that came out of the shell script indicating the return code processing:

```
batchManagerZos exit code= 35 Status=Good: submitted and completed.
Will set shell script exit code to 0
If BPXBATCH, then JES return code should be RC=0000
```

□ Let's do one more thing with batch events -- enable the JSRMON server to watch the topic for job log parts and copy them to a directory when they're published.

Start by copying the monitor application over to JSRMON's /dropins directory:

```
cp /u/user1/jbatch/apps/JobLogEventsDirCreator-1.0.war
       /shared/jsrhome/servers/JSRMON/dropins/JobLogEventsDirCreator-1.0.war
```

□ Next, copy the MQ JCA resource adapter file to the server's directory:

```
cp /shared/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar
                          /shared/jsrhome/servers/JSRMON/wmq.jmsra.rar
```

□ Then, copy in the `server.xml` we prepared ahead of time:

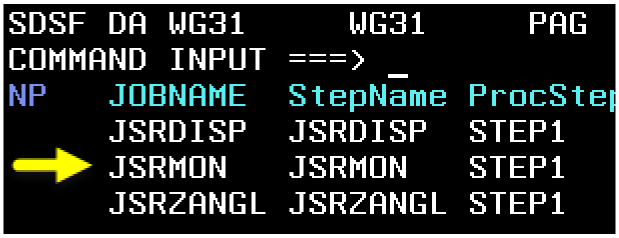
```
cp /u/user1/jbatch/lab4/monitor.xml
                             /shared/jsrhome/servers/JSRMON/server.xml
```

That XML looks like what follows. It has the MQ JMS definitions. *But it has no batch definitions at all.* This is to make the point that **any** process capable of subscribing to the topic can monitor for batch events.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">
```

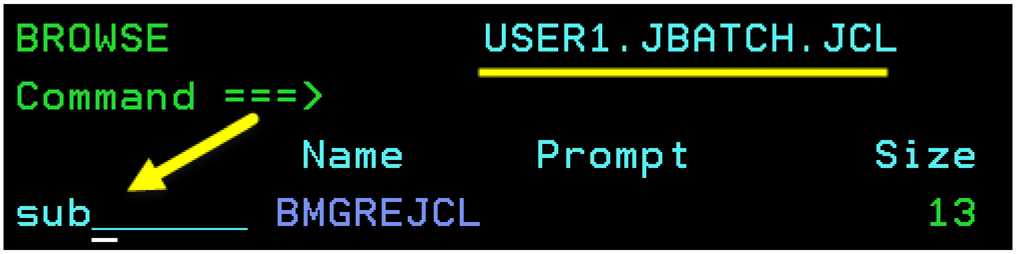
```
    <!-- Enable features -->
    <featureManager>
        <feature>mdb-3.2</feature>
        <feature>jndi-1.0</feature>
        <feature>jca-1.7</feature>
        <feature>servlet-3.1</feature>
        <feature>jsonp-1.0</feature>
        <feature>wmqJmsClient-2.0</feature>
    </featureManager>

    <variable name="wmqJmsClient.rar.location"
      value="${server.config.dir}/wmq.jmsra.rar" />

    <wmqJmsClient startupRetryCount="999"
      startupRetryInterval="1000ms"
      reconnectionRetryCount="10"
      reconnectionRetryInterval="5m">
    </wmqJmsClient>

    <jmsTopic id="JobLogEventTopic"
      jndiName="jms/batch/batchJobTopic">
      <properties.wmqJms
      baseTopicName="batch/jobs/execution/jobLogPart" />
    </jmsTopic>

    <jmsActivationSpec id="JobLogEventsDirCreator-1.0/JobLogEventsSubscriber">
      <properties.wmqJms
       destinationRef="JobLogEventTopic"
       destinationType="javax.jms.Topic"
       transportType="CLIENT"
       channel="SYSTEM.DEF.SVRCONN"
       queueManager="MQS1"
       hostName="wg31.washington.ibm.com"
       port="1414" />
    </jmsActivationSpec>

    <httpEndpoint id="defaultHttpEndpoint"
        host="*"
        httpPort="13080"
        httpsPort="13443" />

</server>
```

□ In the 3270 session, go to **=sdsf.da**, set the prefix to **PRE JSR***, and then issue the start command for the JSRMON server: **/S JSRMON**. You should see the server come up:



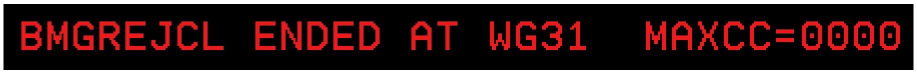
```
SDSF DA WG31      WG31      PAG
COMMAND INPUT ===> _
NP     JOBNAME   StepName  ProcStep
       JSRDISP   JSRDISP   STEP1
       JSRMON    JSRMON    STEP1
       JSRZANGL  JSRZANGL  STEP1
```

□ Go to `USER1.JBATCH.JCL` and **sub** the `BMGREJCL` job again:



```
BROWSE            USER1.JBATCH.JCL
Command ===>
              Name       Prompt        Size
sub_______  BMGREJCL                     13
```

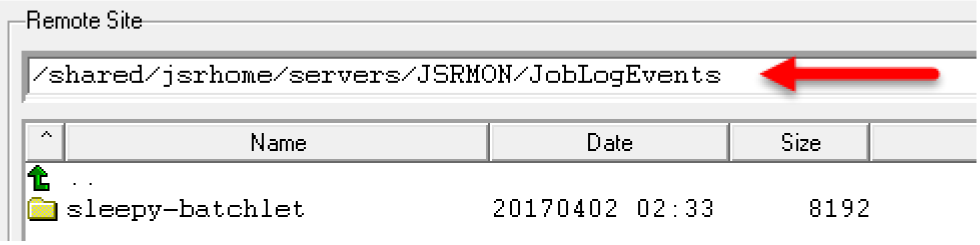
□ Wait about 15 seconds, then hit the host enter key again. You should get the completion message:



```
BMGREJCL ENDED AT WG31  MAXCC=0000
```

□ Using the WS-FTP client, go to this directory on the "remote site":

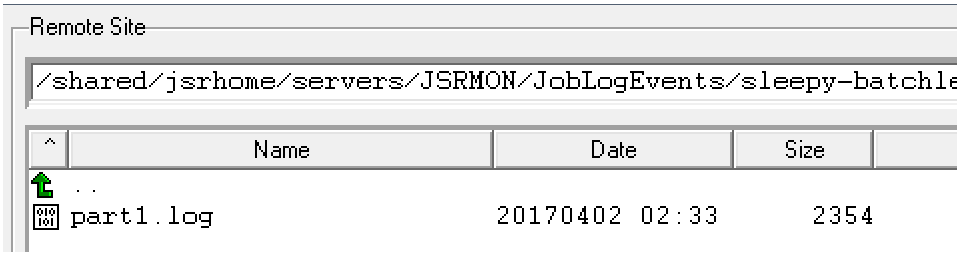
`/shared/jsrhome/servers/JSRMON/JobLogEvents`



```
/shared/jsrhome/servers/JSRMON/JobLogEvents

        Name                Date           Size
..
sleepy-batchlet        20170402 02:33      8192
```

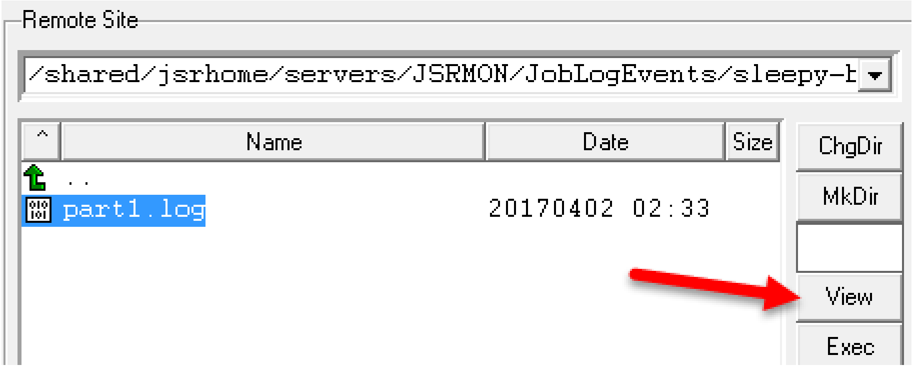
Then double-click on the `sleepy-batchlet` directory. That will take you to the next directory down, which will be today's date.

□ Repeat that process of double-clicking on the directories to go further down the tree. Eventually you will come to the directory where `part1.log` is found:



```
/shared/jsrhome/servers/JSRMON/JobLogEvents/sleepy-batchle

        Name                Date           Size
..
part1.log              20170402 02:33      2354
```

□ Highlight that file and click the "View" button:



```
/shared/jsrhome/servers/JSRMON/JobLogEvents/sleepy-b

        Name                Date          Size
..
part1.log              20170402 02:33
```

That will open a text editor where you'll see the job log.

That job log was retrieved from the MQ topic. The server where it ran -- JSRDISP -- published the job log as an event to the topic, and the MDB application in this server retrieved it and placed it down this path.

- □ Close the text editor showing the job log.
- □ Go to **=sdsf.da**, set the prefix to **PRE JSR***, then enter the command **/P JSRMON** to stop the server. We don't need this server any more this workshop.

Key Points: We've seen two processes take advantage of the batch events:

(1) batchManagerZos, which subscribed to the topic so it could determine when the submitted job was complete

(2) The application in JSRMON, which subscribed to the topic and watched for job log parts to be published.

Any process that can subscribe to a topic can monitor the batch processing. That includes any topic in the topic tree, not just job status or job log part.

AdminCenter

The AdminCenter provides a way to view the jobs through a graphical interface. You can't submit jobs through this interface, but you can view the status of jobs, stop and restart jobs[14], and view job logs.

- □ Start by copying in a new `server.xml` file with the updates needed for the Admin Center:

```
cp /u/user1/jbatch/lab4/adminCenter.xml
                                    /shared/jsrhome/servers/JSRDISP/server.xml
```

That brought in a new feature:

```
<featureManager>
    <feature>servlet-3.1</feature>
    <feature>batch-1.0</feature>
    <feature>batchManagement-1.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>wmqJmsClient-2.0</feature>
    <feature>adminCenter-1.0</feature>
</featureManager>
```

And a little further down in the XML a new role to allow "Fred" to access the AdminCenter:

```
<authorization-roles id="com.ibm.ws.batch">
  <security-role name="batchAdmin">
    <special-subject type="EVERYONE"/>
    <user name="Fred" />
  </security-role>
</authorization-roles>

<administrator-role>
  <user>Fred</user>
</administrator-role>

<batchPersistence jobStoreRef="BatchDatabaseStore" />
```

- □ Start an instance of the Firefox browser.

[14] Starting with 17.0.0.1.

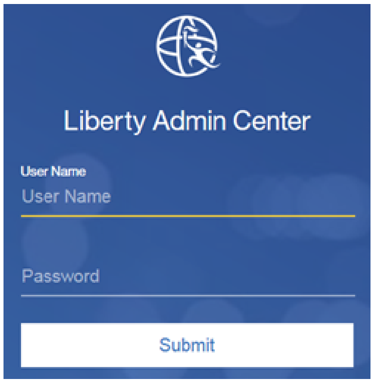
- ☐ Supply this URL into Firefox:

  `http://wg31.washington.ibm.com:10080/adminCenter`

  That is going to throw a "Your Connection is Not Secure" warning.

  > **Why?** Because the self-generated and self-signed certificate created with Liberty "basic" security is not well-known. Your browser does not recognize it.

- ☐ In Firefox, "Add an Exception" and accept the security challenge to allow access to this website even though Firefox is warning you to be careful.
- ☐ You should now see this:



Liberty Admin Center

User Name

User Name

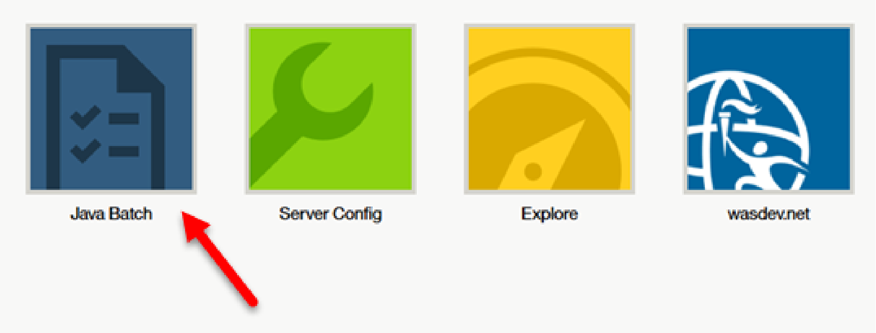
Password

Supply the userid and password (note: *these are case-sensitive*):

User Name	`Fred`
Password	`fredpwd`

Click "Submit".

- ☐ You should now see:



Click on the "Java Batch" tool icon.

- ☐ You should now see a list of the jobs you have run to this point in the workshop.
- ☐ Take a few minutes and explore the interface. See if you can:
  - ○ View a job log (hint: the little page icon)
  - ○ See what server the job ran in (hint: hover mouse near left side of a job listing and you should see a little down arrow appear. Click on that and a second line should appear with the server name.)
  - ○ Modify the columns that display (hint: the little gear symbol)
- ☐ Close the browser.

End of Unit 4 Lab

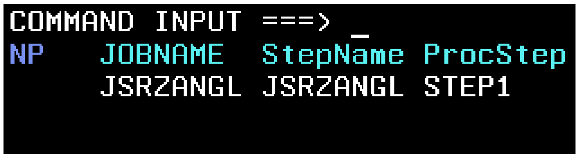
© 2017, IBM Corporation

Unit 5 - Multi-JVM (or "Multi-Server")

Now we're going to separate the job *submission* role and the job *execution* role. The JSRDISP server will act as the "dispatcher," and it'll queue job requests to an MQ queue where an "executor" server (JSREXEC1 or JSREXEC2) will pick up the message and run the job.

Configure JSRDISP and JSREXEC1 for Multi-JVM

□ Start by stopping the JSRDISP server. We do this so we have a nice, clean `messages.log` file to look at. Go to **`=sdsf.da`**, set the prefix to **`PRE JSR*`** and then issue the command **`/P JSRDISP`** to stop that server. Refresh the screen with the host enter key. You should end up with just the Angel:



```
COMMAND INPUT ===> _
NP    JOBNAME  StepName ProcStep
      JSRZANGL JSRZANGL STEP1
```

□ Go to your PuTTY session and change directories to the `/dropins` directory of the JSRDISP server:

```
cd /shared/jsrhome/servers/JSRDISP/dropins
```

□ List the files in that directory with the command **`ls`**. You should see just one file:

```
SleepyBatchletSample-1.0.war
```

□ Delete that file:

```
rm SleepyBatchletSample-1.0.war
```

The "dispatcher" won't run the application; that will take place back in one of the "executor" servers.

□ In the PuTTY session. copy in the new server.xml file:

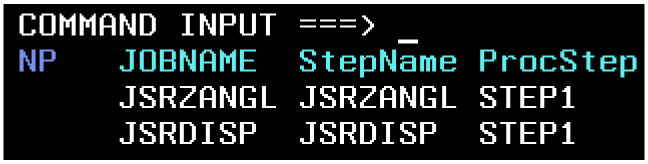
```
cp /u/user1/jbatch/lab5/jsrdisp.xml
                            /shared/jsrhome/servers/JSRDISP/server.xml
```

That brought in the following new things:

```
    <featureManager>
       (no additions to the feature list)
    </featureManager>

    <batchJmsDispatcher
      connectionFactoryRef="batchConnectionFactory"
      queueRef="batchJobSubmissionQueue" />

    <jmsQueue id="batchJobSubmissionQueue"
      jndiName="jms/batch/jobSubmissionQueue">
      <properties.wmqJms baseQueueName="JAVA.BATCH.QUEUE"
        priority="QDEF"
        baseQueueManagerName="MQS1">
      </properties.wmqJms>
    </jmsQueue>

    (no other changes elsewhere in the server.xml)
```

The tag `<batchJmsDispatcher>` enables this server to be a "dispatcher," and the `<jmsQueue>` element defines the queue on which to PUT job submission messages

□ In your 3270 session, go to **=sdsf.da** and start the server with **/S JSRDISP**. You should now see the Angel and the JSRDISP server:



```
COMMAND INPUT ===> _
NP   JOBNAME  StepName ProcStep
     JSRZANGL JSRZANGL STEP1
     JSRDISP  JSRDISP  STEP1
```

□ Let's turn our attention to the first "executor" server. We'll start by copying the SleepyBatchlet sample application to that server's /dropins directory:

```
cp /u/user1/jbatch/apps/SleepyBatchletSample-1.0.war
       /shared/jsrhome/servers/JSREXEC1/dropins/SleepyBatchletSample-1.0.war
```

□ We'll copy the MQ JMS resource adapter to the server's directory:

```
cp /shared/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar
                 /shared/jsrhome/servers/JSREXEC1/wmq.jmsra.rar
```

□ Next, we'll copy in the updated server.xml for this executor server:

```
cp /u/user1/jbatch/lab5/jsrexec1.xml
                    /shared/jsrhome/servers/JSREXEC1/server.xml
```

That server.xml has a lot of JMS and MQ definitions in it, which looks a lot like the JSRDISP server's XML. Rather than show it all[15], we'll highlight three key things in the JSREXEC1 server's server.xml file:

```
<batchJmsExecutor activationSpecRef="batchActivationSpec1"
  queueRef="batchJobSubmissionQueue"/>
```

That enabled this server to act as an "executor." And:

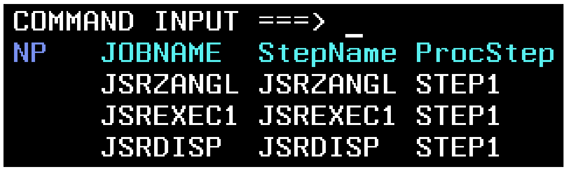
```
 <jmsActivationSpec id="batchActivationSpec1" >
   <properties.wmqJms
     destinationRef="batchJobSubmissionQueue"
messageSelector="com_ibm_ws_batch_applicationName = 'SleepyBatchletSample-1.0'"
     maxPoolDepth="1"
      :
```

The messageSelector tells the JMS activation specification what message to watch for and pick up. In this case it's indicating the SleepyBatchlet application. And finally:

```
<jmsQueue id="batchJobSubmissionQueue"
  jndiName="jms/batch/jobSubmissionQueue">
  <properties.wmqJms baseQueueName="JAVA.BATCH.QUEUE"
     baseQueueManagerName="MQS1">
  </properties.wmqJms>
</jmsQueue>
```

That tells the server what MQ queue to listen on. That's the same queue the JSRDISP dispatcher server was defined to *put* messages on.

□ In 3270, go to **=sdsf.da** and start the server with **/S JSREXEC1**. You should now see:



```
COMMAND INPUT ===> _
NP   JOBNAME  StepName ProcStep
     JSRZANGL JSRZANGL STEP1
     JSREXEC1 JSREXEC1 STEP1
     JSRDISP  JSRDISP  STEP1
```

[15] Feel free to look at the file if you wish. Either OEDIT in z/OS, or use the WS-FTP client to "view" the file.

□ Now you're ready to submit a job to the JSRDISP server, and have it place the job submission message on the queue to be picked up by the JSREXEC1 server listening on that queue.

We'll do *both* batchManager and batchManagerZos:

batchManager

○

```
cd /shared/zWebSphere/Liberty/V17001/bin
```

○

```
./batchManager submit --batchManager=localhost:10443 --user=Fred
                      --password=fredpwd --applicationName=SleepyBatchletSample-1.0
                     --jobXMLName=sleepy-batchlet.xml --trustSslCertificates --wait
```

Note: port 10443 is the port of the JSRDISP server, **not** the executor server.

○ Give it 20 or 30 seconds and you should see `"batchStatus":"COMPLETED"`

○ Look for `"serverId":"localhost//shared/jsrhome/JSREXEC1"` in the output to the PuTTY screen. That tells you that executed in the JSREXEC1 server.

○ Using the WS-FTP client, go to the **`/shared/jsrhome/servers/JSREXEC1/logs`** directory. Highlight the **`messages.log`** file and click on the "View" button. At the bottom you should see:

```
SleepyBatchlet: process: entry
SleepyBatchlet: process: sleep for: 15
SleepyBatchlet: process: [0] sleeping for a second...
SleepyBatchlet: process: [1] sleeping for a second...
SleepyBatchlet: process: [2] sleeping for a second...
 :
SleepyBatchlet: process: [14] sleeping for a second...
SleepyBatchlet: process: exit. exitStatus: SleepyBatchlet:i=15;stopRequested=false
```

That's *more* proof it ran in the `JSREXEC1` server and **not** the `JSRDISP` you submitted to.

batchManagerZos

○

```
cd /shared/zWebSphere/Liberty/V17001/lib/native/zos/s390x
```

○

```
export STEPLIB='MQ901.SCSQAUTH'
```

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                                  --applicationName=SleepyBatchletSample-1.0
                   --jobXMLName=sleepy-batchlet.xml --queueManagerName=MQS1 --wait
```

○ Give it 20 or so seconds and you should see `"batchStatus":"COMPLETED"`

○ Look for `"serverId":"localhost//shared/jsrhome/JSREXEC1"` in the output to the PuTTY screen. That tells you that executed in the JSREXEC1 server.

○ Again, use the WS-FTP client to look at the JSREXEC1 server's messages.log file. You'll see a *second* set of "SleepyBatchlet" output messages. That proves the job ran in that server and not the JSRDISP you submitted against using batchManagerZos.

Key Point: the multi-JVM support works with either batchManager or batchManagerZos in the same way -- the job submission message is placed on the queue, and is then picked up by whatever executor server is listening on that queue with a message selector that matches.

Use message selector to direct job execution

We're going to start two "executor" servers; one with SleepyBatchlet, and the other with a different application -- BonusPayout, which is a "chunk" application. We're going to show how each server will only pick up the application it is defined to pick up based on the message selector in the `server.xml` file.

□ First, we'll copy the BonusPayout application to the `/dropins` directory of the JSREXEC2 server:

```
cp /u/user1/jbatch/apps/BonusPayout-1.0.war
                                /shared/jsrhome/servers/JSREXEC2/dropins/BonusPayout-1.0.war
```

□ Next, we'll copy the MQ JMS resource adapter to the JSREXEC2 directory:

```
cp /shared/mqm/V9R0M1/java/lib/jca/wmq.jmsra.rar
                                        /shared/jsrhome/servers/JSREXEC2/wmq.jmsra.rar
```

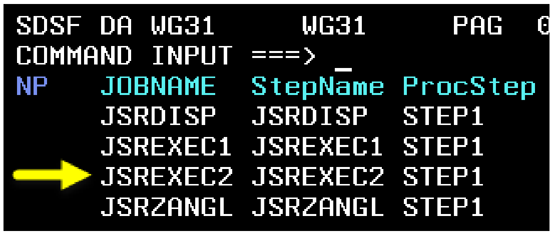
□ Then we'll copy the pre-built `server.xml` to the JSREXEC2 server:

```
cp /u/user1/jbatch/lab5/jsrexec2.xml
                                            /shared/jsrhome/servers/JSREXEC2/server.xml
```

That is *almost* identical to the JSREXEC1 `server.xml` except for HTTP ports and the message selector[16]:

```
    <jmsActivationSpec id="batchActivationSpec2" >
      <properties.wmqJms
        destinationRef="batchJobSubmissionQueue"
  messageSelector="com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'"
        transportType="CLIENT"
```

□ In your 3270 session, go to **=sdsf.da**, set the prefix to **PRE JSR***, and start the server with the command **/S JSREXEC2**. You should see:



```
SDSF DA WG31     WG31     PAG  0
COMMAND INPUT ===> _
NP    JOBNAME  StepName ProcStep
      JSRDISP  JSRDISP  STEP1
      JSREXEC1 JSREXEC1 STEP1
      JSREXEC2 JSREXEC2 STEP1
      JSRZANGL JSRZANGL STEP1
```

□ Using batchManagerZos, submit the BonusPayout job:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
     --applicationName=BonusPayout-1.0 --jobXMLName=SimpleBonusPayoutJob.xml
   --jobParameter=dsJNDI=jdbc/bonus --jobParameter=tableName=BONUSDB.ACCOUNT
                                              --queueManagerName=MQS1 --wait
```

This job completes very quickly. You should see a result in just a few seconds.

□ Inspect the output that comes to your PuTTY terminal. In particular, look for batch status of `COMPLETED` and exit status of `COMPLETED`, and the server ID, which should be `JSREXEC2`:

```
INFO: CWWKY0101I: Job  with instance ID xx has been submitted.

INFO: CWWKY0106I: JobInstance:
{"jobName":"","instanceId":xx,"appName":"BonusPayout-1.0#BonusPayout-
1.0.war","submitter":"","batchStatus":"STARTING","jobXMLName":"BonusPayoutJo
b.xml","instanceState":"JMS_CONSUMED","lastUpdatedTime":"2017/04/03
00:16:49.682 +0000"}

INFO: CWWKY0105I: Job  with instance ID xx has finished. Batch status:
COMPLETED. Exit status: COMPLETED

INFO: CWWKY0107I: JobExecution:
{"jobName":"BonusPayoutJob","executionId":xx,"instanceId":xx,"batchStatus":"
COMPLETED","exitStatus":"COMPLETED","createTime":"2017/04/03 00:16:49.636
```

[16] It also has additional JDBC definitions to access the BONUSDB.ACCOUNT table in DB2.

```
+0000","endTime":"2017/04/03 00:16:51.606
+0000","lastUpdatedTime":"2017/04/03 00:16:51.606
+0000","startTime":"2017/04/03 00:16:49.810 +0000","jobParameters":
{"tableName":"BONUSDB.ACCOUNT","com.ibm.ws.batch.submitter.jobId":"STC00243"
,"com.ibm.ws.batch.submitter.jobName":"USER16","dsJNDI":"jdbc/bonus"},"restU
rl":"https://192.168.17.219:12443/ibm/api/batch","serverId":"localhost//shar
ed/jsrhome/JSREXEC2","logpath":"/shared/jsrhome/servers/JSREXEC2/logs/joblog
s/BonusPayoutJob/2017-04-03/instance.xx /execution.xx /","stepExecutions":
[{"stepExecutionId":xx,"stepName":"generate","batchStatus":"COMPLETED","exit
Status":"COMPLETED"},
{"stepExecutionId":xx,"stepName":"addBonus","batchStatus":"COMPLETED","exitS
tatus":"COMPLETED"},
{"stepExecutionId":xx,"stepName":"validation","batchStatus":"COMPLETED","exi
tStatus":"1000"}]}
```

Use Case: You've just seen one use case for the Multi-JVM design: the ability to submit jobs through a common Dispatcher and have it run in the specific server where the batch job is hosted.

Next we're going to see another use-case: directing job submission to a specific server based on a job parameter you supply.

Use compound message selector

□ We're going to have JSREXEC2 host *both* SleepyBatchlet and BonusPayout. So issue the following command to copy SleepyBatchlet to that server's `/dropins` directory:

```
cp /u/user1/jbatch/apps/SleepyBatchletSample-1.0.war
       /shared/jsrhome/servers/JSREXEC2/dropins/SleepyBatchletSample-1.0.war
```

□ Copy in a new `server.xml` to the JSREXEC1 server:

```
cp /u/user1/jbatch/lab5/jsrexec1_2.xml
                    /shared/jsrhome/servers/JSREXEC1/server.xml
```

The message selector is now:

```
messageSelector="com_ibm_ws_batch_applicationName =
          'SleepyBatchletSample-1.0' AND jobPriority = '1'"
```

This will only pick up messages for SleepyBatchlet where a job parameter of `jobPriority` is set to the value '1'.

□ Copy in a new `server.xml` to the JSREXEC2 server:

```
cp /u/user1/jbatch/lab5/jsrexec2_2.xml
                    /shared/jsrhome/servers/JSREXEC2/server.xml
```

The message selector is now:

```
messageSelector="(com_ibm_ws_batch_applicationName =
         'SleepyBatchletSample-1.0' AND NOT jobPriority = '1')
          OR com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'"
```

This will pick up a BonusPayout job submission or any SleepBatchlet submission where `jobPriority` is *other than* 1 (but it can't be omitted: if not present, the message won't be picked up by either server).

□ Enter the following job submission command:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
 --applicationName=SleepyBatchletSample-1.0 --jobXMLName=sleepy-batchlet.xml
                --queueManagerName=MQS1 --jobParameter=jobPriority=1 --wait
```

Inspect the output and look for the server ID. You'll see that went to JSREXEC1. That's because the jobParameter of `jobPriority=1` was specified.

□ Now enter the following job submission command:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
  --applicationName=SleepyBatchletSample-1.0 --jobXMLName=sleepy-batchlet.xml
                              --queueManagerName=MQS1 --jobParameter=jobPriority=2 --wait
```

You'll find that goes to JSREXEC2. That's because the jobPriority value is *not* 1, but rather it is `jobPriority=2`.

□ Finally, enter the following job submission command:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                    --applicationName=BonusPayout-1.0 --jobXMLName=BonusPayoutJob.xml
      --jobParameter=dsJNDI=jdbc/bonus --jobParameter=tableName=BONUSDB.ACCOUNT
                                                      --queueManagerName=MQS1 --wait
```

That goes to JSREXEC2 because the job is BonusPayout, and that's only being selected for in the `server.xml` of that server.

End of Unit 5 Lab

Unit 6 - Security

Return the environment to the essentials

To keep the spotlight on the essentials, we're going to return the environment to just servers JSRDISP and JSREXEC1. The message selector in JSREXEC1 will be simple: selecting just on the application name for SleepyBatchlet.

□ In your 3270 session, go to **`=sdsf.da`**, set the prefix to **`PRE JSR*`**, and then stop all three servers:

- ○ **`/P JSRDISP`**
- ○ **`/P JSREXEC1`**
- ○ **`/P JSREXEC2`**

□ In your PuTTY session, copy `server.xml` files over to the two servers:

```
cp /u/user1/jbatch/lab6/jsrdisp.xml
                                /shared/jsrhome/servers/JSRDISP/server.xml
```

and

```
cp /u/user1/jbatch/lab6/jsrexec1.xml
                               /shared/jsrhome/servers/JSREXEC1/server.xml
```

□ Go back to your 3270 session and start both servers:

- ○ **`/S JSRDISP`**
- ○ **`/S JSREXEC1`**

□ Now check to make sure things still work as it did before:

Admin Console

- ○ Open a browser and go to **`http://wg31.washington.ibm.com:10080/adminCenter/`**
- ○ Accept the security challenge caused by the self-signed certificate
- ○ Log on with ID of **`Fred`** and **`password`** of fredpwd (case sensitive values)
- ○ Log off (upper right corner "person" icon) and close the browser.

batchManager

- ○ In the PuTTY session: **`cd /shared/zWebSphere/Liberty/V17001/bin`**
- ○ Also in PuTTY: **`export JAVA_HOME=/shared/java/J8.0_64/`**
- ○ And then in PuTTY:

```
./batchManager submit --batchManager=localhost:10443
                                        --user=Fred --password=fredpwd
                            --applicationName=SleepyBatchletSample-1.0
        --jobXMLName=sleepy-batchlet.xml --trustSslCertificates --wait
```

- ○ Wait 30 or so seconds, then look for

```
Batch status: COMPLETED. Exit status: COMPLETED
```

batchManagerZos

- ○ In PuTTY: **`export STEPLIB='MQ901.SCSQAUTH'`**
- ○ In PuTTY: **`cd /shared/zWebSphere/Liberty/V17001/lib/native/zos/s390x`**

- ○ In PuTTY:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
                         --applicationName=SleepyBatchletSample-1.0
      --jobXMLName=sleepy-batchlet.xml --wait --queueManagerName=MQS1
```

- ○ Wait 20 or so seconds, then look for

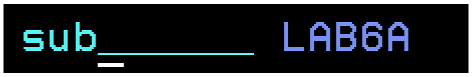
```
Batch status: COMPLETED. Exit status: COMPLETED
```

You have verified the essentially workings of a Multi-JVM topology with basic security.

batchManagerZos: SAF authentication and authorization

We're going to start with this because this is the easiest: it does not involve SSL and certificates.

- □ Go to **=3.4**, list the data set **USER1.JBATCH.JCL**, then place a **b** (for browse) next to the data set to browse.
- □ Then submit the **LAB6A** member:



```
sub_______ LAB6A
```

That (a) creates a SAF ID of `FRED` with password `FREDSAF`, (b) creates the Java batch `EJBROLE` profiles and grants `FRED` to `batchAdmin`; (c) grants `FRED` read to the `CBIND` so that ID can use WOLA.

- □ Copy an updated `server.xml` to the JSRDISP server:

```
cp /u/user1/jbatch/lab6/jsrdisp_2.xml
                         /shared/jsrhome/servers/JSRDISP/server.xml
```

That updated included the following added and ~~removed~~:

```
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>zosLocalAdapters-1.0</feature>
        <feature>wmqJmsClient-2.0</feature>
        <feature>adminCenter-1.0</feature>
        <feature>zosSecurity-1.0</fature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <safRegistry id="saf" />
    <safAuthorization racRouteLog="ASIS" />
    <safCredentials profilePrefix="BBGZDFLT" />

    ~~<basicRegistry id="basic1" realm="jbatch">~~
      ~~<user name="Fred" password="fredpwd" />~~
    ~~</basicRegistry>~~

    ~~<authorization-roles id="com.ibm.ws.batch">~~
      ~~<security-role name="batchAdmin">~~
        ~~<special-subject type="EVERYONE"/>~~
        ~~<user name="Fred" />~~
      ~~</security-role>~~
    ~~</authorization-roles>~~
```

~~<administrator-role>~~

~~<user>Fred</user>~~

~~</administrator-role>~~

□ Copy in the updated server.xml for the JSREXEC1 server:

```
cp /u/user1/jbatch/lab6/jsrexec1_2.xml
                                    /shared/jsrhome/servers/JSREXEC1/server.xml
```

The changes there are essentially the same as for the other XML you copied in.

□ Open up a *second* PuTTY session and log on as **FRED** with a password of **FREDSAF**.

□ Enter the following two commands to get ready to use batchManagerZos:

○ `cd /shared/zWebSphere/Liberty/V17001/lib/native/zos/s390x`

○ `export STEPLIB='MQ901.SCSQAUTH'`

□ Enter the following command to test FRED's ability to use WOLA[17]:

```
./batchManagerZos ping --batchManager=LIBERTY+BATCH+MANAGER
```

If *successful*, that should respond with nothing ... just the prompt.

□ You're ready to submit a batchManagerZos command under the ID of FRED:

```
./batchManagerZos submit --batchManager=LIBERTY+BATCH+MANAGER
 --applicationName=SleepyBatchletSample-1.0 --jobXMLName=sleepy-batchlet.xml
                                              --wait --queueManagerName=MQS1
```

If you wait the usual 20 or so seconds for SleepyBatchlet to complete, you should see the "COMPLETED" message.

□ Enter the command **exit** into the PuTTY session you started with the FRED ID. That will close the session.

Validation? How do we know it was actually FRED that submitted that job? You can see in the output:

```
"submitter":"FRED"
```

(When using WOLA, the ID used is the ID under which the batchManagerZos client is operating. In this case you logged into PuTTY with FRED, so FRED was the ID used.)

How do we know we went to SAF for authorization? The server.xml has no knowledge of FRED any more, nor any knowledge of what FRED is authorized to do. We removed all that from the server.xml. The XML now points to SAF. It went to SAF to check that FRED was a valid ID.

Finally, we know FRED has access to the EJBROLE because if that ID did not have access you would have received an error trying to submit the command:

```
CWWKY0304W: User FRED is not authorized to start batch jobs
```

You didn't see that, so FRED has access to the EJBROLE.

[17] This is really a test of FRED's access to the CBIND profile that protects against unauthorized use of WOLA against a server, based on the WOLA "three-part name" for that server.

AdminCenter + batchManager: SAF authentication and authorization

When it comes to SAF authentication and authorization, the AdminCenter and batchManager are very similar to one another. We're leaving SSL off the table for now. We'll pick that up to end this lab.

- □ Go to the **USER1.JBATCH.JCL** data set and submit the **LAB6B** member.

> **Notes:** That job creates the unauthenticated group and guest, the APPL profile, and two EJBROLE profiles. Feel free to browse the member if you'd like.
>
> That job assumes the LAB6A job was run. Notably: the FRED user ID, and the Java Batch authorization EJBROLE profiles.

- □ Try the AdminCenter:

○	Open a browser and go to: **http://wg31.washington.ibm.com:10080/adminCenter/**
○	Accept any security challenges you receive. (We have not yet changed out the SSL certificate, so for now the browser is complaining just as it was before.)
○	Try to log in with the previous ID/PW value of **Fred** and **fredpwd**. You should get an error: *"Login error: unrecognized user name and/or password."* That's because the password "fredpwd" is what we used when it was "basic security." Fred now has a SAF password of FREDSAF.
○	Now log in with **FRED** and **FREDSAF**. That should succeed. That ID/PW combination is defined in SAF, and it has access to the "Administrator" role required to access the AdminCenter.
○	Log off the AdminCenter and close the browser.

- □ Go to your remaining PuTTY session. Enter the command **whoami** to confirm that is the one logged in with the `USER1` ID.
- □ Enter the command: **cd /shared/zWebSphere/Liberty/V17001/bin**
- □ Now enter the job submission command:

```
./batchManager submit --batchManager=localhost:10443
    --user=FRED --password=FREDSAF --applicationName=SleepyBatchletSample-1.0
                   --jobXMLName=sleepy-batchlet.xml --trustSslCertificates --wait
```

Note the ID and password ... this is the new `FRED`/`FREDSAF` combination.

That should result in: `Batch status: COMPLETED. Exit status: COMPLETED`

AdminCenter + batchManager: SAF Keyrings and SSL

We're going to layer on the final piece of this -- pushing the SSL certificate down into SAF keyrings. The "certificate authority" we are going to use is RACF-generated, so it *still* won't be recognized by a browser. In the "real world" you would sign your server certificate with a well-known certificate authority.

- □ In your 3270 session, go to **USER1.JBATCH.JCL** and submit the job **LAB6C**.

  That job (a) generates a CA certificate and a server certificate; (b) signs the server cert with the CA cert; (c) creates a server keyring and a client keyring; (d) connects the certs to the keyrings; and (d) gives the server ID and client ID the ability to access the certificates.

- □ In your PuTTY session, copy the new `server.xml` over to the JSRDISP server:

```
cp /u/user1/jbatch/lab6/jsrdisp_3.xml
                              /shared/jsrhome/servers/JSRDISP/server.xml
```

The following were added and ~~removed~~ by that copy operation:

```
    <featureManager>
        <feature>servlet-3.1</feature>
        <feature>batch-1.0</feature>
        <feature>batchManagement-1.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>zosLocalAdapters-1.0</feature>
        <feature>wmqJmsClient-2.0</feature>
        <feature>adminCenter-1.0</feature>
        <feature>zosSecurity-1.0</feature>
        <feature>ssl-1.0</feature>
    </featureManager>

    ~~<keyStore id="defaultKeyStore" password="Liberty"/>~~

    <sslDefault sslRef="DefaultSSLSettings" />
    <ssl id="DefaultSSLSettings"
      keyStoreRef="CellDefaultKeyStore"
      trustStoreRef="CellDefaultTrustStore" />
    <keyStore id="CellDefaultKeyStore"
      location="safkeyring:///Keyring.SERVER"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />
    <keyStore id="CellDefaultTrustStore"
      location="safkeyring:///Keyring.SERVER"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />

    <safRegistry id="saf" />
    <safAuthorization racRouteLog="ASIS" />
    <safCredentials profilePrefix="BBGZDFLT" />
```

□ Copy the new server.xml for the JSREXEC1 server as well:

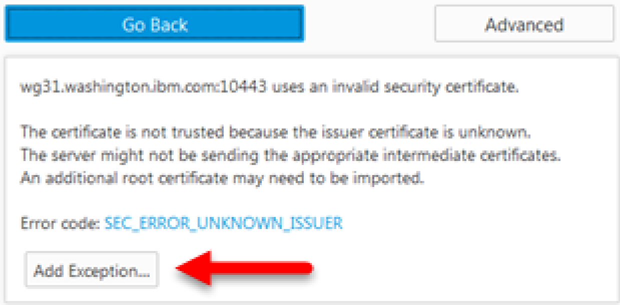
```
cp /u/user1/jbatch/lab6/jsrexec1_3.xml
                            /shared/jsrhome/servers/JSREXEC1/server.xml
```

The updates for that are the same as we saw for JSRDISP.

□ Open a Firefox[18] browser and go to:

```
http://wg31.washington.ibm.com:10080/adminCenter/
```

You will be challenged because the authority that signed the server certificate is not a well-known authority. We expected that. Click on the "Add Exception" button:



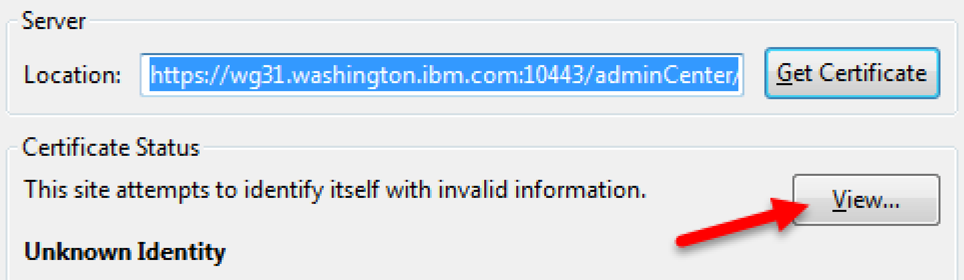
wg31.washington.ibm.com:10443 uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is unknown.
The server might not be sending the appropriate intermediate certificates.
An additional root certificate may need to be imported.

Error code: SEC_ERROR_UNKNOWN_ISSUER

[18] Firefox makes it easier to inspect the certificate. Chrome makes it more challenging.

□ Click on the "View" button to view the certificate:



Server

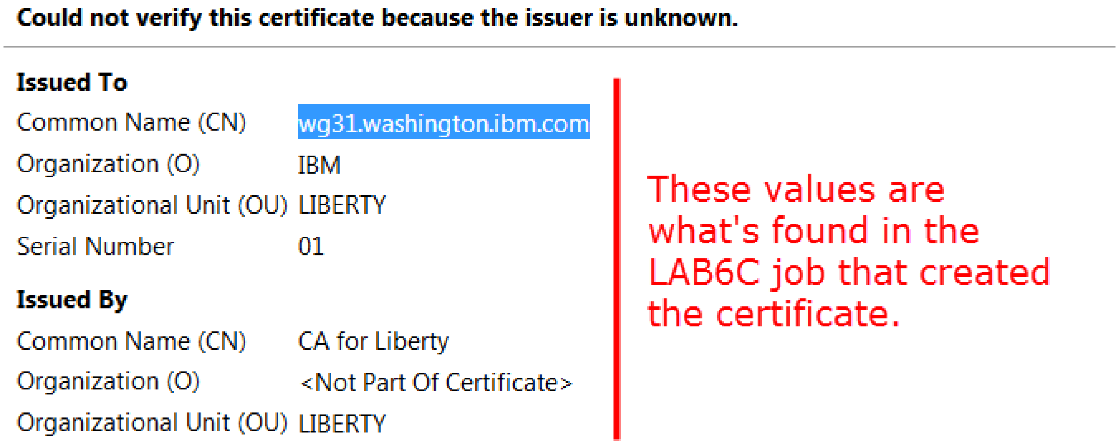
Location: https://wg31.washington.ibm.com:10443/adminCenter/

Certificate Status

This site attempts to identify itself with invalid information.

Unknown Identity

You should see:



Could not verify this certificate because the issuer is unknown.

Issued To

Common Name (CN)	wg31.washington.ibm.com
Organization (O)	IBM
Organizational Unit (OU)	LIBERTY
Serial Number	01

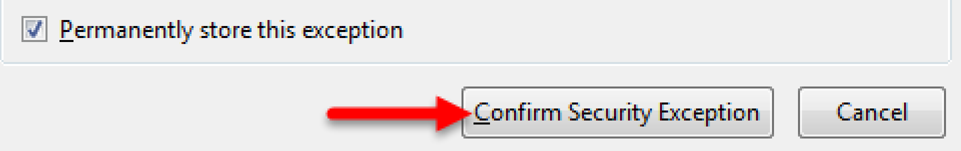
Issued By

Common Name (CN)	CA for Liberty
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	LIBERTY

These values are what's found in the LAB6C job that created the certificate.

That verifies the SAF-certificate is being used and *not* the previous Liberty-generated one.

□ Click the "Close" button on the certificate view, then click "Confirm Security Exception" button:



☑ Permanently store this exception

□ Then click the "Add Exception" button. You should be taken to the AdminCenter login panel.

□ Log in with **FRED** and **FREDSAF**. You should gain entry, just as you did before.

□ Close the browser.

□ Go to your PuTTY session. Enter **whoami** to confirm the active ID is USER1. That's important because the keyring that holds the CA certificate is connected to the USER1 ID.

Note: If the ID that returns from whoami is FRED, then close the PuTTY session.

If the ID that returns from whoami is JSRSERV, then it means you're still operating under JSRSERV from when you "switched users" to that ID earlier. Enter **exit** to return to the original USER1 ID.

□ For batchManager to understand that it must go to a SAF keyring to get the CA certificate so it can trust the server certificate, you must enter a very long JVM_ARGS string.

***Note:** get this string from the Lab Commands file. Copy and paste this command.*

```
export JVM_ARGS="-Djavax.net.ssl.trustStore=safkeyring://USER1/Keyring.CLIENT
                                        -Djavax.net.ssl.trustStoreType=JCERACFKS
                      -Djavax.net.ssl.keyStore=safkeyring://USER1/Keyring.CLIENT
                                          -Djavax.net.ssl.keyStoreType=JCERACFKS
                                           -Dcom.ibm.ssl.keyStoreFileBased=false
                                         -Dcom.ibm.ssl.trustStoreFileBased=false
                            -Djava.protocol.handler.pkgs=com.ibm.crypto.provider
                                      -Djavax.net.ssl.keyStorePassword=password"
```

□ Now you can enter the batchManager command:

```
./batchManager submit --batchManager=wg31.washington.ibm.com:10443
   --user=FRED --password=FREDSAF --applicationName=SleepyBatchletSample-1.0
                                     --jobXMLName=sleepy-batchlet.xml --wait
```

Note two things:

1. The host name is no longer "localhost" but rather than host name of the z/OS system. That's because that's the host name that's part of the server certificate.
2. The `--trustSslCertificates` parameter is no longer present. That's what we used before to get by the untrusted nature of the Liberty-generated self-signed certificate. With that removed, we're telling the client to validate the certificate. It can do that if it knows about the SAF keyring where the CA certificate is stored.

You should see the job get submitted and run to completion.

End of Unit 6 Lab

And End of Document