

SHARE Session #8369: z/OS Batch Java Applications

Introduction

Welcome! We hope that this lab session will be not only instructional (and challenging), but also fun! Please don't hesitate to ask questions during the lab, and please fill out an evaluation. We appreciate your comments as they will help us to improve future offerings.

Objectives

1. Deploy Java applications from a workstation IDE (Eclipse) to z/OS.
2. Become familiar with how to run z/OS batch jobs under the JZOS Batch Java launcher.
3. Write a Java jobstep to interact with MVS datasets
4. Write a Java jobstep to perform XML Processing
5. Write a Java jobstep to interact with a SOAP web service

Prerequisites

Lab participants will need to be comfortable with using z/OS, including ISPF, SDSF, JCL, and a very basic familiarity with z/OS Unix System Services. If you don't have these skills, ***please team up with someone*** who does; see an instructor and we'll help find you a partner. It will also be helpful if you have a basic understanding of Java, but this is not strictly required.

Some Details

- This lab will use the IBM SHARE LPAR, with is running z/OS 1.7. If you have IBM contacts, please thank them for making this invaluable resource available to SHARE.
- We will be using IBM JAVA SDK 5.0 (31-bit), although everything presented will also run under SDK 1.4.2.
- You will be using JZOS 1.2.4 (the IBM alphaWorks version). As mentioned in the previous session (#8368), IBM has announced that the JZOS batch launcher will be integrated into an upcoming level of the z/OS Java SDK.
- The lab instructors will help you to get connected to the IBM SHARE z/OS LPAR using the 3270 emulator on your workstation.
- If you were not able to attend the previous session: #8368 “Java: z/OS Stand-Alone Applications”, please see if any extra handouts are available, since they contain some helpful background for this lab.
- For more information on using Java on z/OS, see <http://dovetail.com>. Materials used in this lab are available at this site for download. A forum is available for your questions on running Java applications on z/OS.

Setting up Eclipse to Deploy Java Batch Applications to z/OS

This lab uses Eclipse and an Apache Ant task to allow editing of Java source (and JCL) on the workstation, and transfer these files to z/OS.

Your TSO ID for this lab will be **sharenn**, where nn is your assigned unique number. This lab does **NOT** use the ids *SHAREAnn* or *SHAREBnn*. If you try to use one of these ids, your labs will not work properly!

1. Using a 3270 emulator, use ISPF option 3.2 to allocate a one cylinder PDS named 'SHAREnn.JBATCH.JCL' with RECFM=FB, LRECL=80, and Directory Blocks=5. If you don't know how to do this, ask an instructor for help.
2. Start Eclipse on your workstation using the "Eclipse jzosLabs" icon on your desktop.
3. Locate the BatchSample project in the Package Explorer and expand it.
4. Make a copy of the file zos.properties.template

- Left click on the file **zos.properties.template** to select
- Right click + Copy
- Left click on the project **BatchSample** to select
- Right click + Paste
- When prompted, type the name **zos.properties**

5. Edit the file **zos.properties** and customize for your specific SHARE id:

Property	Value
server	mvsl.centers.ihost.com
userid	SHAREnn (your sharenn id)
password	this will be supplied by the instructors
jcl.dsn	'SHAREnn.JBATCH.JCL'
loadlib.dsn	'KIRK.JZOS.LOADLIB'
jzos.version	50
sdk.home	/usr/lpp/java15/J5.0
app.home	/sharelab/sharenn/jbatch

Save your changes: File+Save or (Ctrl-S).

6. The file **deploy.xml** contains the complete Ant project, which will use the customized properties you set in the previous step to deploy your batch java applications. If you are familiar with Ant, feel free to spend some time examining this file, then execute it as follows:

- Left click on the Ant script **deploy.xml** to select
- Right click + Run As+Ant Build...
- In the dialog, locate and select the "JRE" tab
- Choose "Run in the same JRE as the workspace"
- Press "Apply"
- Press "Run"

After running the script, your Eclipse console should look something like this:

```
Buildfile: /home/goetze/workspace/BatchSample/deploy.xml
compile:
buildJar:
  [mkdir] Created dir: /home/goetze/workspace/BatchSample/deploy
  [jar] Building jar:
/home/goetze/workspace/BatchSample/deploy/application.jar
deployJar:
  [echo] Copying files to
mvs1.centers.ihost.com:/shareuser/goetze/jbatch...
  [ftp] sending files
  [ftp] transferring
/home/goetze/workspace/BatchSample/deploy/application.jar
  [ftp] 1 files sent
deployJcl:
  [echo] Copying JCL members to
mvs1.centers.ihost.com:'goetze.jbatch.jcl'...
  [ftp] sending files
  [ftp] transferring /home/goetze/workspace/BatchSample/jcl/EXJZ0SVM
  [ftp] transferring /home/goetze/workspace/BatchSample/jcl/SAMPLE50
  [ftp] 2 files sent
all:
BUILD SUCCESSFUL
Total time: 4 seconds
```

7. At this point, whenever you make changes to java/JCL files in Eclipse and re-run **deploy.xml**, the changed files will be uploaded to z/OS.

Note: The labs that follow require changes to the JCL that you just uploaded. You may either edit the JCL on z/OS using ISPF, or edit it with the Eclipse editor and deploy your changes to z/OS. Please note that changes you make with ISPF will be overwritten (lost) if you subsequently use Eclipse to make changes.

Hello World

This lab section demonstrates how to run a simple Java application on z/OS under the JZOS batch toolkit. When complete, you will have verified that JZOS and Java are installed correctly and that you can run Java programs from a z/OS batch environment.

1. Review the PDS member **<HLQ>.JBATCH.JCL(HELLO)**:

- Verify that `JAVACLS='com.dovetail.jzos.jbatch.HelloWorld'`.
- In the STDENV DD locate the comment:
Add application home directory and jars to CLASSPATH
and observe how all of the .jar files found in the directory APPL_HOME are added to the classpath. This will ensure that the application.jar that you are FTPing to z/OS from Eclipse will be added to the classpath and make your classes loadable.

2. Submit the job and switch to SDSF to view the job output.

Tips:

- SDSF is option “S” from the SHARE ISPF primary options menu.
- Consider splitting your ISPF screen so that you can tailor and submit from one screen and view output from the other.
- So that SDSF filters only your jobs, issue the command:

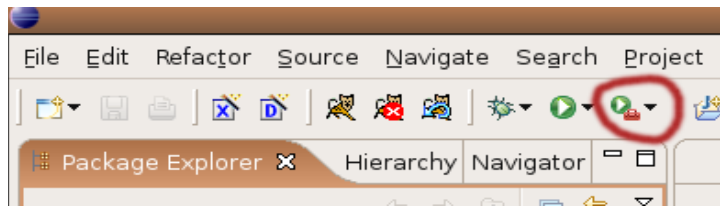
```
PREFIX SHAREnn*
```

3. Locate your job in the list, and bring up the individual DDNAME list via the “?” command. Browse the each of the following DDs and confirm that they exist and contain the expected output.

<i>DDNAME</i>	<i>Description</i>
SYSOUT	Contains the output from the JZOS batch launcher. Confirm that the launcher finished with an exit code of 0.
STDOUT	Contains the redirected Java standard output stream (System.out). This DD should contain the text: Hello Java Batch!

4. Now, back in Eclipse, make a change to the class **com.dovetail.jzos.jbatch.HelloWorld** by changing the output string to “Hello SHARE!”.
5. Save the file (CTRL-S).

6. Run the Ant script again. A one-click shortcut is to click the Eclipse external tools “run” icon:



7. Submit the job again (using your 3270 emulator) and check the job output to see the result of your changed java class.

This edit+deploy+submit cycle is quite efficient and makes it possible to use a rich programming environment like Eclipse in tandem with the traditional z/OS batch job facilities.

Interacting with Jobstep Datasets

This lab uses a Java class to read the records from the INPUT jobstep DD and write the results to the OUTPUT jobstep DD. This example illustrates how easy it can be to interact with datasets using Java in Batch, and should serve as a template for doing other similar tasks.

1. Locate the class `com.dovetail.jzos.jbatch.ZFileCopy` in Eclipse using the Package Explorer and study the `main()` method. Note that this class expects the DD names INPUT and OUTPUT to be present, and that the `com.dovetail.jzos.ZFile` class is used to open both DDs in record mode.
2. Examine the while loop that copies the records from the INPUT DD to the OUTPUT DD. What is returned from `ZFile.read()` when there are no more records in the dataset? _____
3. Review the PDS member **<HLQ>.JBATCH.JCL(DSCOPY)**. Verify that:
 - `JAVACLS='com.dovetail.jzos.jbatch.ZFileCopy'`.
 - `INPUT=SYS1.MACLIB(ACB),DISP=SHR`
 - `OUTPUT DD SYSOUT=*`
4. Submit and run your JOB. Verify that the ACB member of SYS1.MACLIB was written to SYSOUT.
5. (Optional) The JZOS class `com.dovetail.jzos.ZFile` class has many useful methods, one of which `getRecfm()`, can be used to retrieve the record format of the opened dataset.
 - Using Eclipse to edit `com.dovetail.jzos.jbatch.ZFileCopy`, write a new line to STDOUT (using `System.out.println()`) to display the RECFM of the INPUT DD.
 - Save your changes, deploy to z/OS, submit the JCL and verify that your output contains your changes.
6. (Optional) Experiment with different INPUT and/or OUTPUT DD statements.

XML processing with Java batch

Java is used extensively throughout the industry for XML processing. The lab that follows illustrates how to integrate this technology into a traditional MVS batch job stream. This lab reads a tab delimited file (U.S. Congressional member data) from the Internet and formats it as an XML document. Then in a subsequent step, the XML data is parsed and converted to columnar data. This lab illustrates several useful Java/batch job techniques:

- Using Java and several open source libraries to access the Internet and parse XML documents.
- Using Java return codes for conditional jobstep processing.
- Using a common JZOS configuration script to control multiple Java jobsteps.

In the first step below, we pre-allocate the sequential datasets to hold the XML and the columnar data. In a traditional jobstream the XML data would be allocated as a temporary dataset and deleted upon job completion, but we will want to examine the intermediate data once the job has executed.

1. Using a 3270 emulator, use ISPF option 3.2 to allocate two sequential datasets:
 - **<SHAREnn>.CONGRESS.XML**
 - **<SHAREnn>.CONGRESS.DATA**

For both datasets, use the following allocation parameters:

DCB= (RECFM=VB, LRECL=200, BLKSIZE=20000) and SPACE= (CYL, (1, 1)). Be sure to specify 0 (zero) directory blocks to get sequential datasets. If you don't know how to do this, ask an instructor for help.

2. Locate the JCL member **XMLSAMP** and change all 3 occurrences of **<HLQ>** with your SHAREnn id. You may do this either from ISPF or an Eclipse editor.
3. Locate the two steps in the JCL, **FTPXML** and **PARSEXML**. Note that the STDENV DD looks different than it did for the previous lab steps. Since this job runs multiple job steps, the JZOS environment setup is done via a shared setup script: **<APP_HOME>/jzos_config.sh**. This script accepts user specific environment variable exports (e.g. **<APP_HOME>**) and does all of the STDENV setup necessary to run JZOS.
4. What java class is executed by the step **FTPXML**? **com.dovetail.jzos.jbatch._____**
5. In step **PARSEXML** examine the COND statement. This step will only execute if the previous step (**FTPXML**) produced a condition code of 0. When EXJZOSVM runs, it returns a non-zero exit code if the requested Java class could not be executed or failed during execution (e.g. throws an uncaught exception). Batch job streams can make use of the feature to ensure that job step processing stops or skips if there was a problem with a Java step.
6. If you used the Eclipse editor to make the JCL changes, re-deploy the application to z/OS.
7. Submit the JCL and examine the joblog and confirm that the two datasets were populated properly.
8. (optional) Locate the class **com.dovetail.jzos.jbatch.FtpDownloadToXml** in Eclipse and answer the following questions:
 - What class is used to retrieve the data from the Internet? _____
 - Which method produces the **<member>** element for each member of congress?

9. (optional) Locate the class **com.dovetail.jzos.jbatch.ParseXmlToFile** in Eclipse and answer the following questions:
- What method is called to process the congress member's data once it has been parsed?

 - What class calls this method? _____

Java batch web service client

Accessing SOAP protocol web services is easy to do with Java in batch on z/OS. The Apache Software Foundation provides an open source SOAP toolkit for Java called Axis. With this toolkit, it is possible to quickly build a Java application around a Web Services Description Language (WSDL) document and use this application to call the web service. This lab illustrates the following techniques:

- Use the Apache Axis WSDL2Java Ant task to generate the stub classes required to invoke a SOAP web service that validates an email address.
- Write a simple Java client that uses the generated classes
- Submit JCL that will execute Java in Batch to call the Web Service and return the results.

NOTE: Web services are generally called with hostname based URLs. However, the SHARE LPAR may not have an active DNS server, which will require that the web service be called by IP address. This is an optional step in the lab that follows. Your instructor will tell you if you need to perform this step.

1. In the BatchSample project, locate the makesoap.xml file in the Package Explorer. Then:

- Left click on the "makesoap.xml" to select
- Right click + Run As+Ant Build...
- In the dialog, locate and select the "JRE" tab
- Choose "Run in the same JRE as the workspace"
- Press "Apply"
- Press "Run"

2. After the script runs, you should see several new classes in the **src** package **net.webservicex.www**. Take a moment to review at these classes.
3. Locate the class **com.dovetail.jzos.jbatch.EmailValidator** and uncomment the package import near the top of the class.
4. Uncomment the lines in the main() method that invoke the web service.
5. Save your changes. Ensure that there are no errors in the class by looking for error markers on the left side of the editor.
6. Locate the JCL member **SOAPSAMP** and change **my_email@host.com** to an email address that you would like to check.
7. (optional) uncomment the line that follows the email address (it starts with `://* http://`) to make the call to the web service with an IP address rather than a hostname.
8. Deploy your changes to z/OS. To do this, you will need to re-run deploy.xml:

- Left click on the "deploy.xml" to select
- Right click + Run As+Ant Build

9. Submit your JCL and verify the results. Try re-submitting with different email addresses.
10. (optional, harder) If you have time, use makesoap.xml to build stub classes for a different web service. You can supply different WSDL to the ant script by editing the file **makesoap.properties** and supplying a new WSDL URL. Here are some possible choices:
 - Temperature Conversion: <http://www.webservicex.net/ConvertTemperature.asmx?WSDL>
 - Stock Quote: <http://www.webservicex.net/stockquote.asmx?WSDL>
 - WHOIS service: <http://www.webservicex.net/whois.asmx?WSDL>

Once you generate your helper classes, you will need to write a Java client similar to EmailVerifier to invoke the web service. Don't forget to change your SOAPSAMP JCL to point this new class!