InterConnect 2016

The Premier Cloud & Mobile Conference

Session: Lab 1296

Java Batch on z/OS: WebSphere Application Server and WebSphere Liberty Profile Development Lab

Lab Instructions

Author:
Timothy C. Fanelli Principal Consultant, ZBL Services, Inc. tfanelli@zblservices.com

February 21 – 25
MGM Grand & Mandalay Bay
Las Vegas, Nevada

Contents

Introduction

Welcome

Welcome to the Java Batch on z/OS: WebSphere Application Server and WebSphere Liberty Profile development lab! This section will provide some very basic information about working with this lab's computing environment, an outline of the tasks that you will complete, and an overview of the learning outcomes.

The Development Environment

About the Linux Environment

The development environment for this lab is a Linux operating system, running Redhat Enterprise Linux (RHEL) 6.6. Linux is a free, open-source operating system kernel; RHEL is a commercial implementation of a Linux Operating System, and is a popular alterative to other commercially available OS's such as Microsoft Windows.

If you are not familiar with Linux, or have limited experience, do not worry! Almost all of the work we do in this lab will be completed in *cross-platform* IBM Rational tools, including:

- Eclipse Mars with WebSphere Developer Tools for Liberty Profile

Each of these products is built on top of the Eclipse Integrated Development Environment (IDE), which is a popular open-source IDE used by application development teams world wide.

In addition, we will also be using WebSphere Application Server ("WAS") for Developers or WebSphere Liberty Profile's ("Liberty") v8.5.5.0 release. WAS is a full-fledged traditional application server environment, complete with several enhancements from IBM to support many complex enterprise tasks, including Batch Job execution; Liberty is a small footprint, composable application server offering from the IBM, and is part of the IBM WebSphere Application Server version 8.5 and later offerings. Both WAS for Developers and Liberty are available for free for development and testing on the WebSphere DeveloperWorks! websites, here:

http://www-03.ibm.com/software/products/en/appserv-wasfordev

http://wasdev.net/downloads/liberty-profile-beta/

Since all of the tools and server software we will use are cross-platform, you will be able to install and use all of the software from this lab on your Windows, and even Mac, developer workstations! We chose Linux for this lab simply because it is freely available, and simple for your lab instructors to use in case you need assistance debugging or diagnosing any issues that arise during this lab.

User ID and Password

In case you accidentally reboot the computer, log out of the workspace, or if the screen saver locks the screen, you may need either the User ID or Password to log back in or unlock the screen. They are:

User ID: ibmdemo
Password: ibmdem0 *(note: that's a zero, not a capital "oh"!)*

Objectives and Outcomes

This lab is broken into two parts:

In Part 1, we will start with an existing Liberty Profile Java Batch application, which we will deploy and test on a local Liberty Profile server so you can get a feel for the mechanics of Liberty Profile's Java Batch runtime.

In Part 2, we will create a new project from scratch, using the **doctorbatch.io** framework offering, from ZBL Services, Inc., to build a Java Batch application, which can read and write records from a sequential dataset on z/OS. This application will be tested locally on our development workstation, and we'll discuss how it will run on z/OS.

If time allows, once everyone has completed both parts, we'll demonstrate live on screen how we can take the same batch application, and package it for deployment to WebSphere Application Server, and also execute it on a live z/OS system!

At the end, you should have a feel for the end-to-end workflow covering development, testing, deployment and execution of Java Batch jobs in both runtime environments. You will also see how you can architect your batch applications for easy portability between runtime environments, using the **doctorbatch.io** library.

Part 1: Deploy and Execute a Liberty Profile Java Batch Application

In this section, we will look at an existing WebSphere Liberty Profile Java Batch application, deploy it to WebSphere Application Server, and execute a batch job.

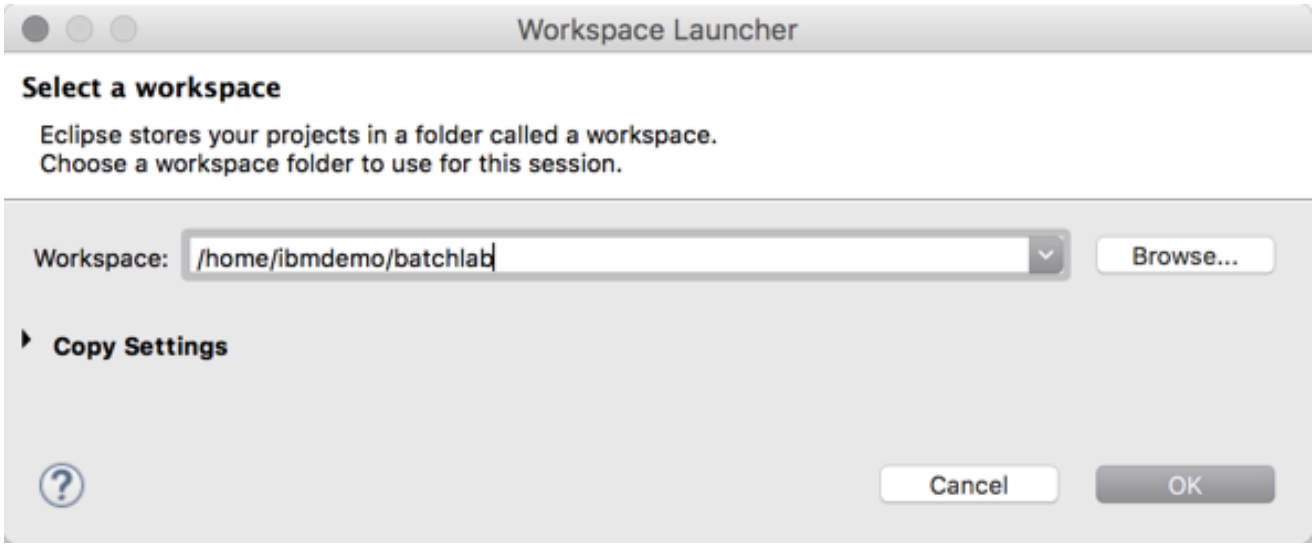
Launching the Eclipse IDE

Step 1: Locate the “Eclipse” icon on your desktop, and double-click it. This will launch the Eclipse development environment.

About this step:
The Eclipse development environment is a free, open-source IDE. The development image also has the WebSphere Developer Tools installed, available here:

https://developer.ibm.com/wasdev/downloads/liberty-profile-using-eclipse/

We will refer to this environment from now simply as the “Integrated Development Environment”, or IDE for short. When you launch the IDE, it will prompt you to choose a workspace location. This location is the path on your computer in which your projects will be stored.



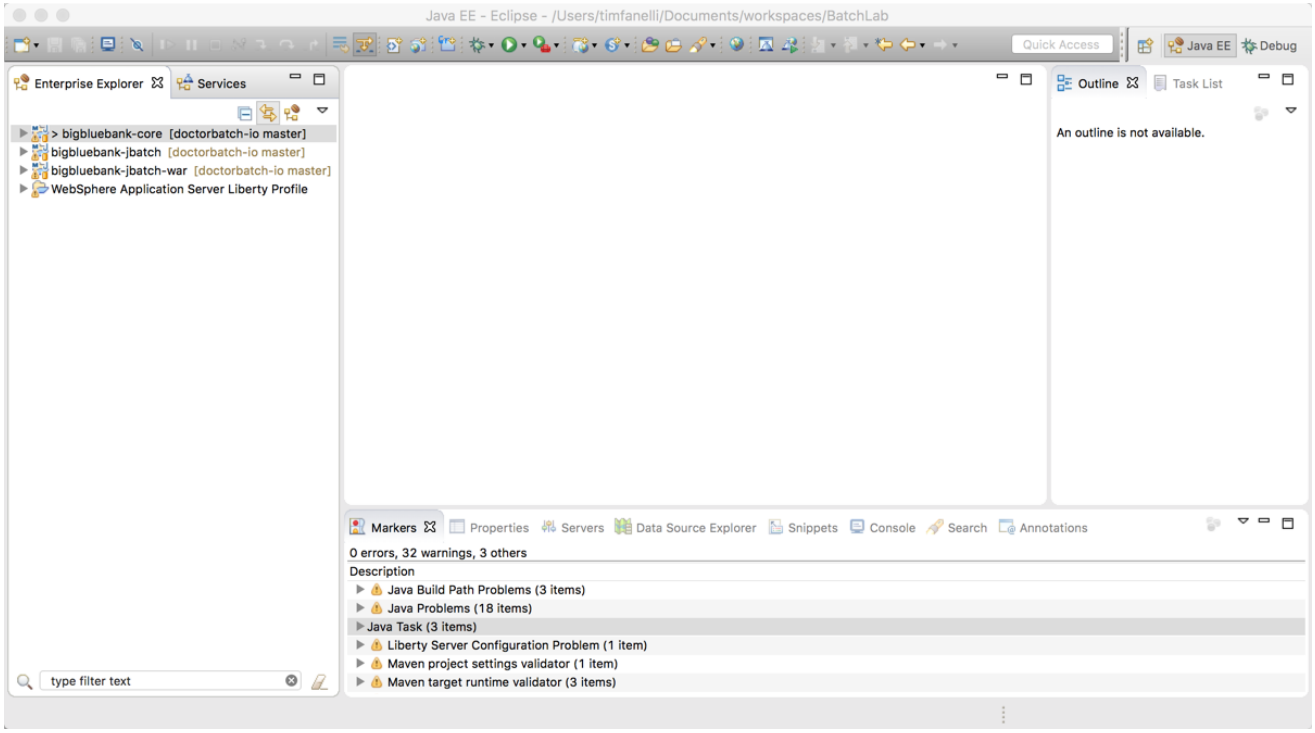
Step 2: In the “Workspace” field, enter:

/home/ibmdemo/batchlab

Step 3: Click “OK.”

Hint: this is already the default value, so you can just go ahead and click OK!

The IDE will load the existing workspace, with a Java Batch project all ready to go.



About the BigBlueBank Application

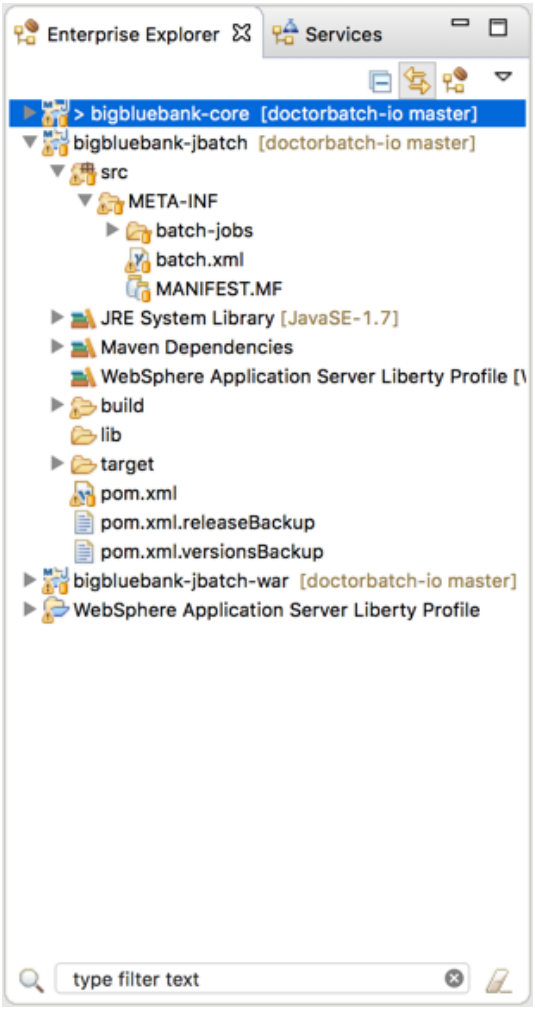
The workspace contains three projects that make up the Big Blue Bank application:

Application Name	Description
bigbluebank-core	This Java Project contains the application's data model, and reusable business logic.
bigbluebank-jbatch	This Java Batch Project wraps the bigbluebank-core library in a Java Batch project, which describes the inputs, outputs, and processing behavior for processing DepositRecord entries in a batch run.
bigbluebank-jbatch-war	This web project wraps the bigbluebank-jbatch for deployment to Liberty Profile. It does not contain any application code; just some meta-information and packaging required by Liberty Profile's implementation of the Java Batch standard.

Let's explore these projects to get a feel for the contents of a Java Batch application.

Step 1: Expand open the bigbluebank-jbatch project, and look inside it's "src/META-INF" and "lib" folders:

You can see in the “src” folder that there is no Java code; the business logic implementation is actually contained in our bigbluebank-core project.



Under the “src” folder is a subfolder named “META-INF” which contains a batch.xml file, and a batch-jobs folder. These artifacts provide important information about the structure of the Java Batch project, and the jobs to execute, which we will explore more in depth through out this lab.

Later, we’ll inspect the “pom.xml” file, which is the Project Object Model (or simply “pom”). The POM declares a project’s dependencies, which are managed by Apache Maven. There, we’ll see that this project includes ZBL Services’ **doctorbatch.io** library, which provides most of the application’s reusable functionality.

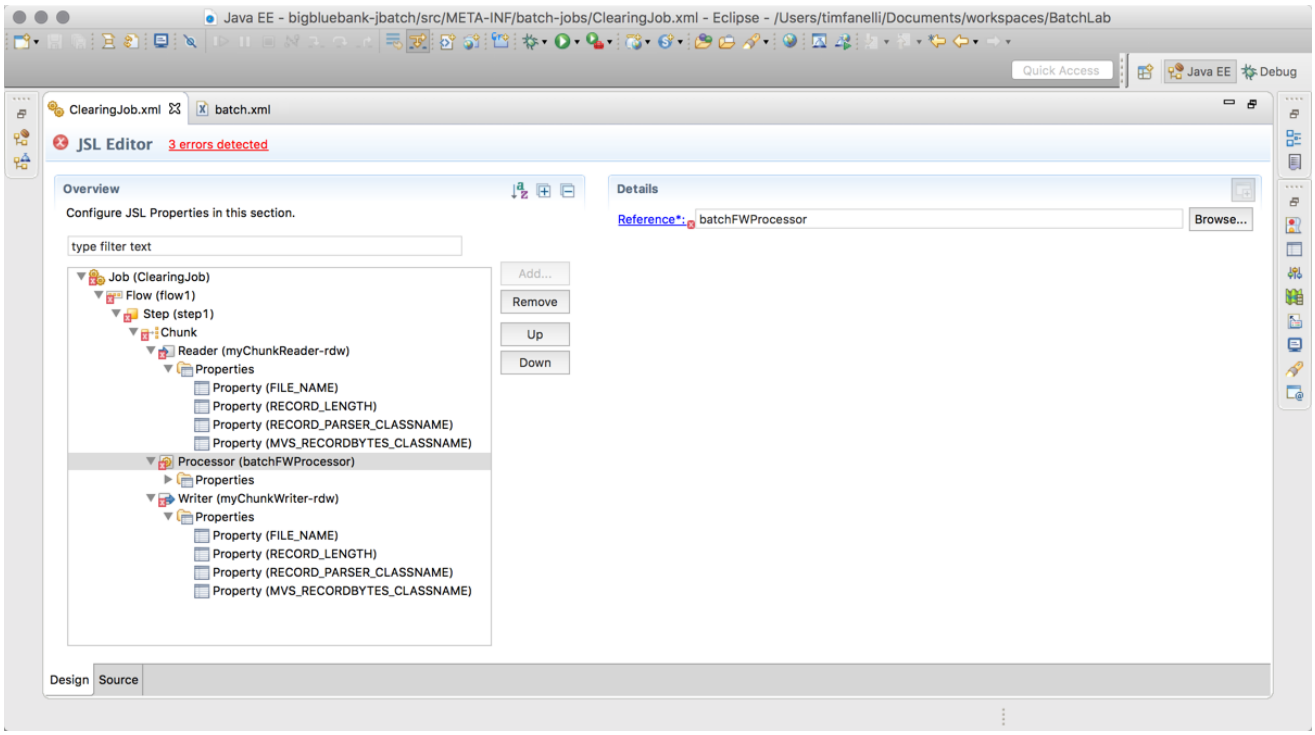
In addition to **doctorbatch.io**, this application also uses the following libraries from IBM:

- jzos-2.4: This library is part of your Java on z/OS installation. It provides utilities required by the doctorbatch.io library to access MVS record-oriented datasets.
- marshall-1.0.0.jar: This library is also part of your Java on z/OS installation. It is used to map native z/OS data to Java data types, allowing us to easily interoperate with your existing data layouts.

Finally, look in the “src/META-INF/batch-jobs” folder. Here, you’ll find a single file: ClearingJob.xml. This is our XML Job Specification Language document, which describes the batch job to the Java Batch runtime environment. Let’s inspect this document, so we can learn how a Java Batch job is structured.

Step 2: Double-click the ClearingJob.xml file to open it in the editor panel.

Step 3: In the editor-panel (the main part of the screen), you’ll see an open tab for the ClearingJob.xml file. Double click the tab to maximize the view.



In the “Design” view, you can see the tree-structure that makes up this job. The “Source” view will show you the raw XML. To switch between Design and Source views, use the tabs at the bottom-left corner of the editor panel.

NOTE: At the time of this writing, an error existed in the workspace configuration which caused the workspace to display “3 errors detected” message; however, no errors exist in the project.

You can see that this job’s ID is “ClearingJob”, and contains a single flow called “flow1”. A flow is a container for job steps, and will be important to us later when we partition the job in part 3. Our flow contains a single step, called “step1”, which uses Java Batch’s chunk processing model.

The chunk step contains several items:

1. Reader: used to obtain records for processing from a persistent data store, such as IBM DB2, MVS sequential datasets, or Unix flat files.
2. Processor: used to validate, transform, or process records obtained from the Reader.
3. Writer: used to store processed records into a persistent data store, which may be the same or different than the persistent store used by the Reader.

The Reader, Processor, and Writer each specify a “Reference”. The Reference is resolved to an implementation class by looking up the reference name inside the batch.xml, which we located earlier.

---

4. Properties: the Reader, Processor, and Writer each contain a Properties element, which is used to pass externalized parameters (for example, data set name) into the job at runtime.

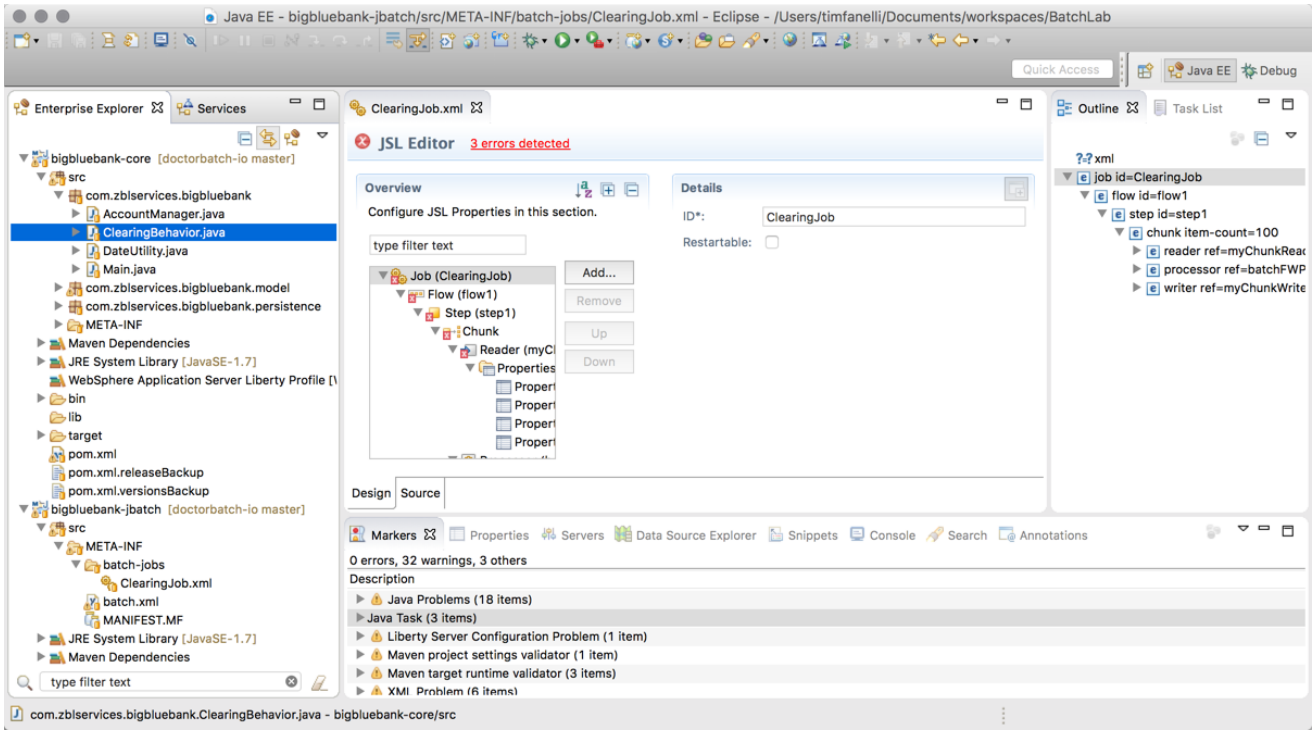
> The Reader, Processor, and Writer implementations dictate which properties must be defined, and will vary from project to project.

Now, let's take a look at the bigbluebank-core project.

Step 4: Double-click the tab for ClearingJob.xml at the top of the editor panel, to restore the editor panel to it's original size.

Step 5: Collapse the bigbluebank-jbatch project, and expand open the bigbluebank-core project.

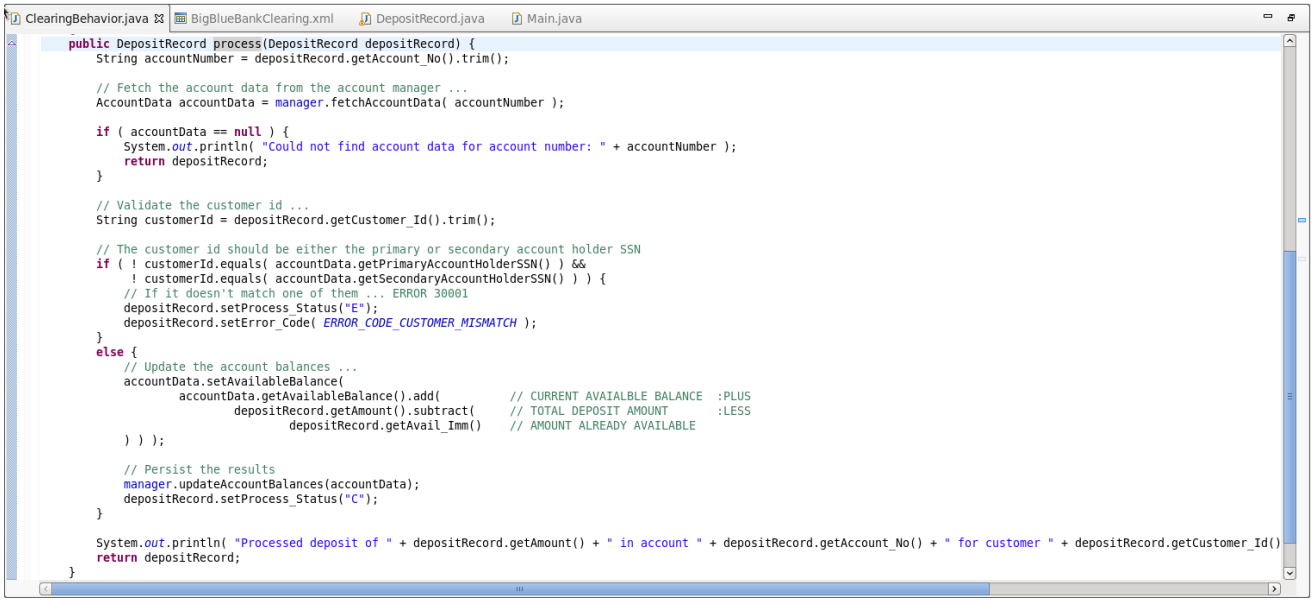
Step 6: Expand open bigbluebank-core project, the "src" folder under it, and the "com.zblservices.bigbluebank" package under that. Your view should now look like this:



Step 7: Double-click the ClearingBehavior.java file to open it in the editor panel.

This class implements the RecordProcessor interface defined by the **doctorbatch.io** library. Chunk steps that use the framework's SimpleProcessor class rely on the application developer to provide an implementation of this interface to encapsulate the custom business logic for your specific application.

Step 8: In the editor panel, scroll down to find the “process” method beginning on Line 31.




```
    public DepositRecord process(DepositRecord depositRecord) {
        String accountNumber = depositRecord.getAccount_No().trim();

        // Fetch the account data from the account manager ...
        AccountData accountData = manager.fetchAccountData( accountNumber );

        if ( accountData == null ) {
            System.out.println( "Could not find account data for account number: " + accountNumber );
            return depositRecord;
        }

        // Validate the customer id ...
        String customerId = depositRecord.getCustomer_Id().trim();

        // The customer id should be either the primary or secondary account holder SSN
        if ( ! customerId.equals( accountData.getPrimaryAccountHolderSSN() ) &&
             ! customerId.equals( accountData.getSecondaryAccountHolderSSN() ) ) {
            // If it doesn't match one of them ... ERROR 30001
            depositRecord.setProcess_Status("E");
            depositRecord.setError_Code( ERROR_CODE_CUSTOMER_MISMATCH );
        }
        else {
            // Update the account balances ...
            accountData.setAvailableBalance(
                    accountData.getAvailableBalance().add(          // CURRENT AVAIALBLE BALANCE  :PLUS
                            depositRecord.getAmount().subtract(     // TOTAL DEPOSIT AMOUNT       :LESS
                                    depositRecord.getAvail_Imm()    // AMOUNT ALREADY AVAILABLE
            ) ) );

            // Persist the results
            manager.updateAccountBalances(accountData);
            depositRecord.setProcess_Status("C");
        }

        System.out.println( "Processed deposit of " + depositRecord.getAmount() + " in account " + depositRecord.getAccount_No() + " for customer " + depositRecord.getCustomer_Id()
        return depositRecord;
    }
```

This method takes a DepositRecord instance as its input and produces a modified DepositRecord instance as it’s output. The inputs and outputs are serialized by the Batch Data Streams defined in our xJCL.

The business logic here will be invoked once for each input record. Each record is processed according to the logic coded in this method. The record processing logic here is simple:

1. A DepositRecord passed in, which contains an account number, a customer ID, a deposit amount, and the amount of that deposit that was made available for immediate withdrawal.
2. We load the AccountData record for this account through a utility called the AccountManager. The AccountManager is application-code that knows how to load an AccountRecord from a persistent data store, such as a DB2 database, or VSAM file.
3. The DepositRecord is validated against the AccountData, confirming that the customer ID in the DepositRecord matches one of the account holder’s social security numbers in the AccoundData record.
    a. If the record is invalid, we set an error code in the DepositRecord
    b. If the record is valid, we post the remaining deposit amount to the account’s available balance, and set a successful status code.
4. We log an output message to STDOUT
5. The modified DepositRecord is returned so that it can be stored or updated by the Batch Data Stream.

Once again, in Part 2, we’ll go through the application more slowly and build one from scratch; so if you are still unsure how all these pieces tie together, don’t worry! It’s not important right at this moment.

Deploy and Test the BigBlueBank Application

Now that we've explored the BigBlueBank Clearing application a bit, we're ready to deploy and execute it.

Step 1: In the bottom portion of your IDE, below the editor panel, you will see several tabs. The third tab is called "Servers". Click the Servers tab.

Don't see the "Servers" tab?

1. Make sure you have restored the editor panel to it's original size, by double-clicking the open tab at the top of the editor screen.
2. Make sure you are in the "Java EE" perspective. Raise your hand if you're not sure what that means!

Step 2: Right-click "WebSphere Application Server Liberty Profile at localhost" and click "Start" to launch the local Liberty Profile server for testing.

Before you came to this lab, we created a WebSphere Liberty Profile Server and configured it to run Java Batch applications! If you're interested in the configuration and topology for WAS environments, please don't hesitate to ask! We're happy to discuss it with you, and have a whole other class dedicated to just this topic.

After you click Start, the Servers tab will disappear momentarily, and you'll see the Console tab instead. The view may alternate back and forth between these two until the application server has started completely. This is normal behavior!

Step 3: Confirm that the application server has launched successfully, by looking at the Console tab in the bottom portion of your IDE. The last message in the console log should read:

```
[AUDIT   ] CWWKF0011I: The server defaultServer is ready to run a smarter planet.
```

Your application server is now started, so we can deploy applications to it, and execute and test them.

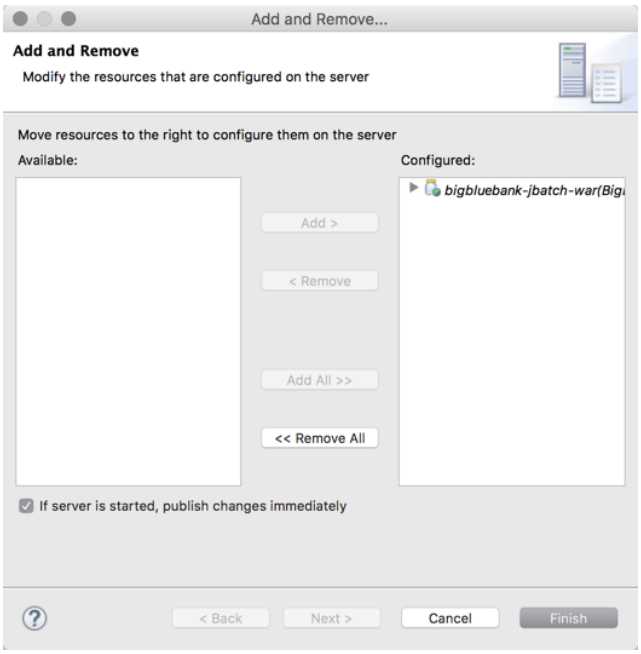
Step 5: Click back on the Servers tab

Step 6: Right click "WebSphere Application Server Liberty Profile at localhost", and choose "Add and Remove"

In the dialog, you will see the list of projects in your workspace that are eligible for deployment to WebSphere Liberty Profile. There should be only one project available: bigbluebank-jbatchWAR

This is an "Web Archive", which contains the bigbluebank-jbatch and bigbluebank-core projects as well as all their dependencies.

Step 7: Highlight the project under the "Available" list, and then click "Add >" to move it to the "Configured" list.



Step 8: Click Finish to save the changes to the server's configuration.

If your server is already started when you do this step, you may notice the "Console" tab takes focus periodically in the bottom portion of your IDE. This is distracting, but normal. The Console is showing the WAS server's log output. This tab takes focus when changes are displayed.

Congratulations, you just deployed your first batch application to a WebSphere Liberty Profile! Let's test it out to make sure it all works.

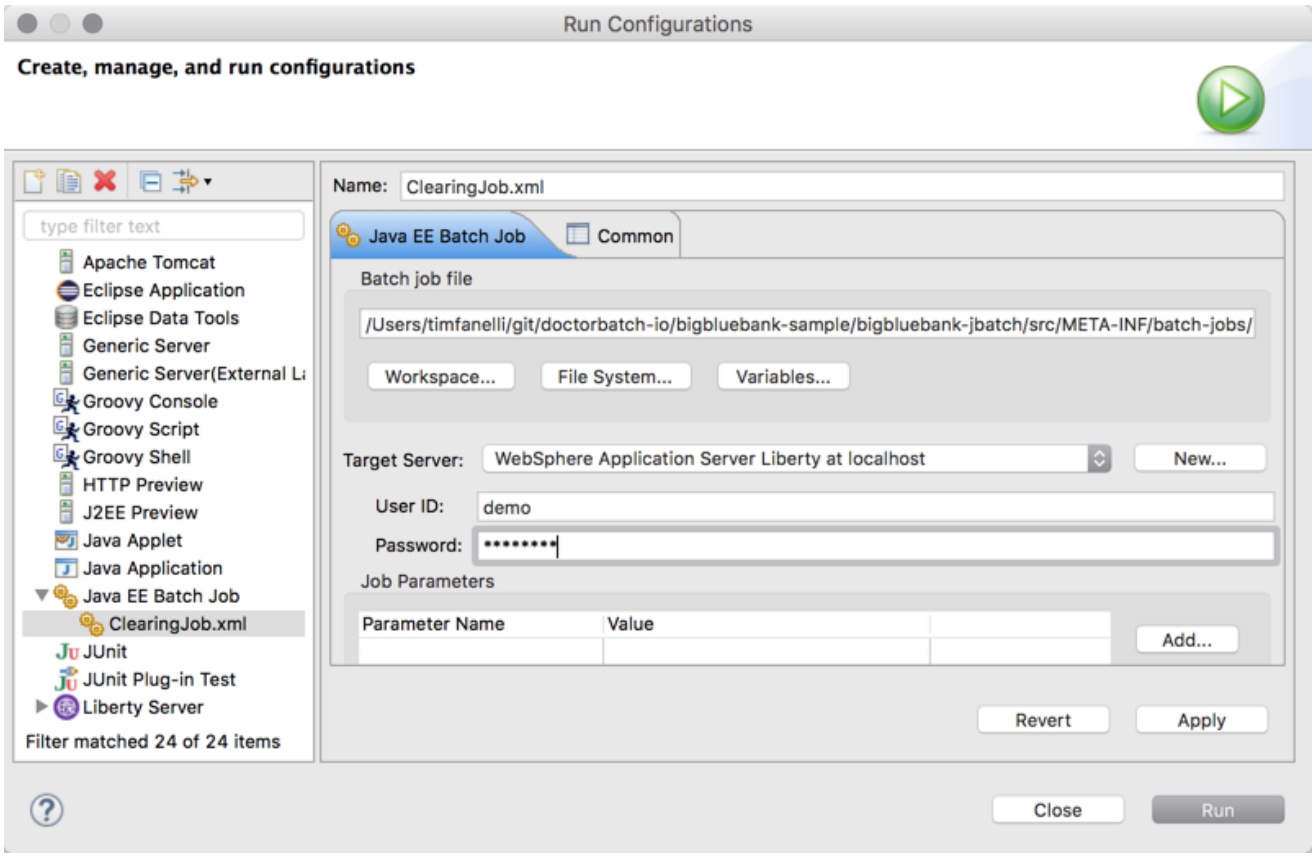
Step 9: Locate the ClearingJob.xml file again. It's under the bigbluebank-jbatch project's "src/META-INF/batch-jobs" folder.

Step 10: Right-click ClearingJob.xml, and select Run As → Java EE Batch Job

Step 11: In the dialog that appears, notice that your server is already populated in the "Target Server" dropdown.

Step 12: Observe that the username and password are already prefilled as well; this is because the configuration was already created for you.

User ID: demo
Password: password



Step 13: Click Run

Step 14: Observe that there are many lines of output generated in the Console view, showing processed deposits of varying amounts for different customers. Any output generated by the batch application is displayed in the WebSphere Liberty Profile's STDOUT log.

Since Liberty Profile is a multi-threaded environment capable of handling dozens, hundreds, or even thousands of simultaneous job instances, it's important to be able to isolate each job's output from any other job's.

The Java Batch runtime in Liberty Profile does this automatically for us. Each job instance keeps its own separate log file. We can see this job's output independently by exploring the job instances visible in the Java EE Batch Job Logs view.

Step 15: In the bottom portion of your Eclipse IDE, locate and click the Java EE Batch Jobs tab.

> This tab may have already appeared when your batch job completed its execution. If you are having trouble locating it, you can also select "Show View" from the "Window" menu, and select "Java EE Batch Jobs" in that list.

Step 16: The job instances are sorted by instance number, which is a sequential job counter. You'll find your most recent job submission at the bottom of the list.

Step 17: Double-click your job submission to open that job's log in the web browser.

If you are prompted for a username and password, enter:

User: demo

Password: password

Liberty Profile's Java Batch implementation exposes job operation capabilities through a REST API interface. The WebSphere Developer Tools for Eclipse provide views for interacting with these interfaces.

Later, when we demonostrate the execution of these jobs on z/OS, you will see Liberty Profile's command line and scripting interfaces as well, for integration with your existing job scheduling subsystems.

Congratulations! You just deployed and tested a Java Batch application in WebSphere Liberty Profile. Believe it or not, this job actually parsed a Record Oriented z/OS dataset with EBCDIC and COMP-3 encoded data right on our Linux workstations! This would work equally well on Windows.

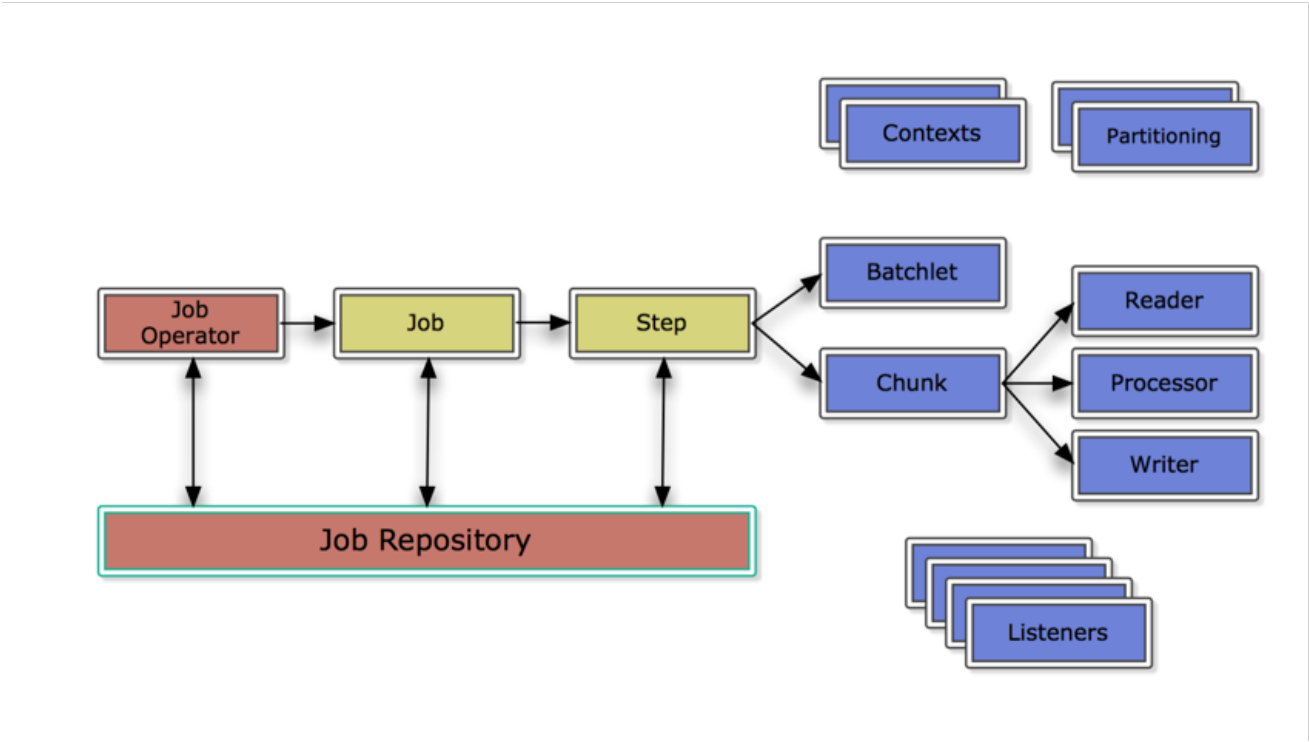
Part 2: Developing with doctorbatch.io

Now that we've seen the mechanics of deploying and testing an existing application, it's time to take a step back, and drill into the development of a WebSphere Batch application.

We'll provide an overview of the WebSphere Batch development model, and then delve into the doctorbatch.io library a bit, before creating a new project from scratch.

Java Batch Programming Model

The Java Batch programming model is shown below.



While there are many components, the majority of development work will be done by implementing the Reader, Processor, and Writer components of a "Chunk" step.

You can learn more about the Java Batch programming model in the presentation "Three Key Concepts for Understanding JSR-352: Batch Programming for the Java Platform", available on Slide Share, here:

http://www.slideshare.net/timfanelli/three-key-concepts-for-java-batch-27473387

doctorbatch.io library

The doctorbatch.io library provides several reusable components that make it easy to write Java Batch applications, particularly when any of the following requirements must be met:

1. The application should be interoperable with WebSphere Application Server's Batch runtime environment
2. The application must access data stored in MVS Data Sets on z/OS
3. The application must consume business rules in IBM Operational Decision Manager

The **doctorbatch.io** library provides an easily extensible model, so that new capabilities, such as input or output to DB2, RESTful APIs, or MQ, could be added by application developers.

Most importantly, the **doctorbatch.io** library provides application portability across WebSphere Batch and the new Java Batch (JSR-352) in Liberty Profile, as we'll see later in Part 3.]

The **doctorbatch.io** library is an Apache 2.0 Licensed Open Source project, and is available here:

http://zblservices.com/doctorbatch-io/

Creating a New Java EE Batch Project

Creating a Clean Slate

Let's start by removing the existing batch application from our workspace, so we don't get confused later …

Step 1: In the Servers tab, right click "WebSphere Application Server Liberty Profile at Localhost" and choose "Add and Remove"

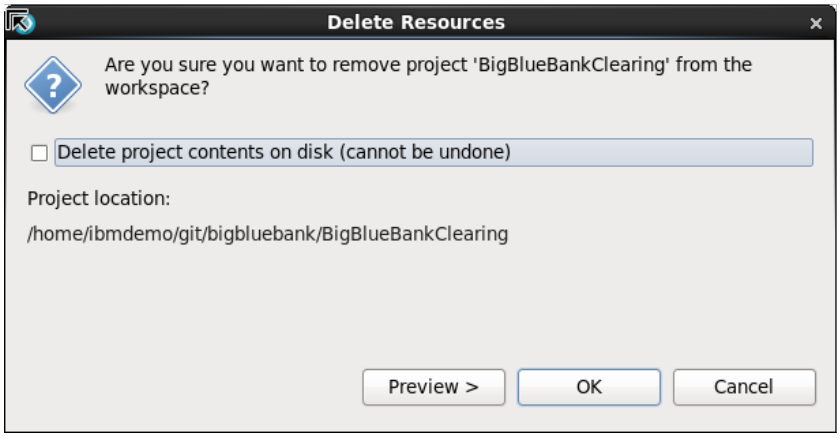
Step 2: In the dialog that appears, highlight "bigbluebank-jbatch-war" from the "Configured" list

Step 3: Click "Remove" to move it back to the "Available" list.

Step 4: Click "Finish"

This will remove the application from the server. Again, you'll see the Console window appear briefly.

Step 5: Right click the bigbluebank-jbatch project, and select "Delete"



Do not check the "Delete project contents on disk" checkbox.

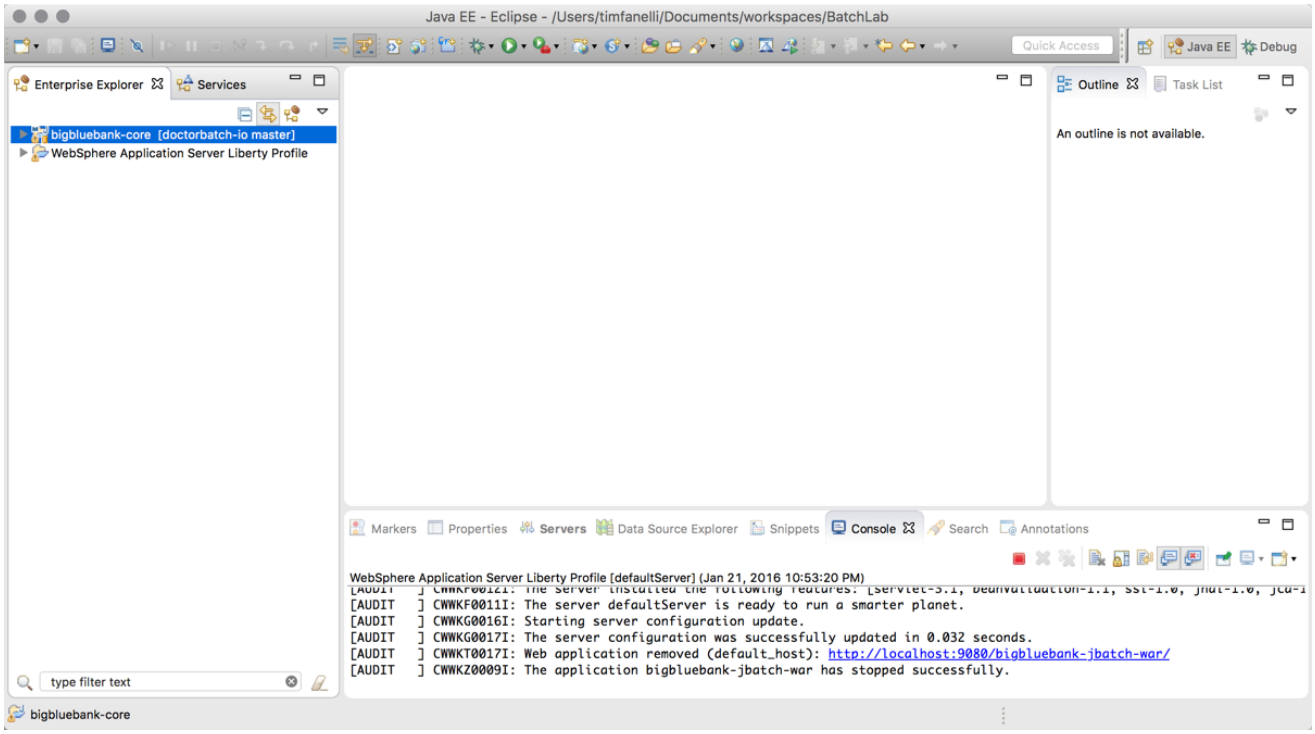
Step 6: Click OK

Step 7: Repeat steps 5 and 6 for the bigbluebank-jbatch-war project.

Your workspace should now only have the bigbluebank-core project in it.

Step 8: If there are any tabs still open in the Editor panel, you may go ahead and close those now, too.

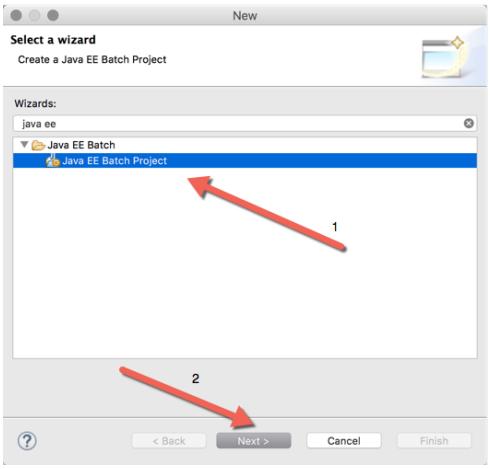
Your workspace should look similar to this:



Create a New Java EE Batch Project

Now that we've wiped the slate clean, we can create a new Batch Project and rebuild what we just deleted.

Step 1: In the File menu, select New → Other. In the dialog that appears, enter "Java EE Batch" in the filter, then select "Java EE Batch Project" from the tree.



Step 2: In the dialog that appears, enter a project name. This can be anything you like, really, but we'll use:

MyBatchApplication

Step 3: Click "Finish"

Notice that there are now *three* new projects in your workspace:

1. MyBatchApplication
2. MyBatchApplicationWAR

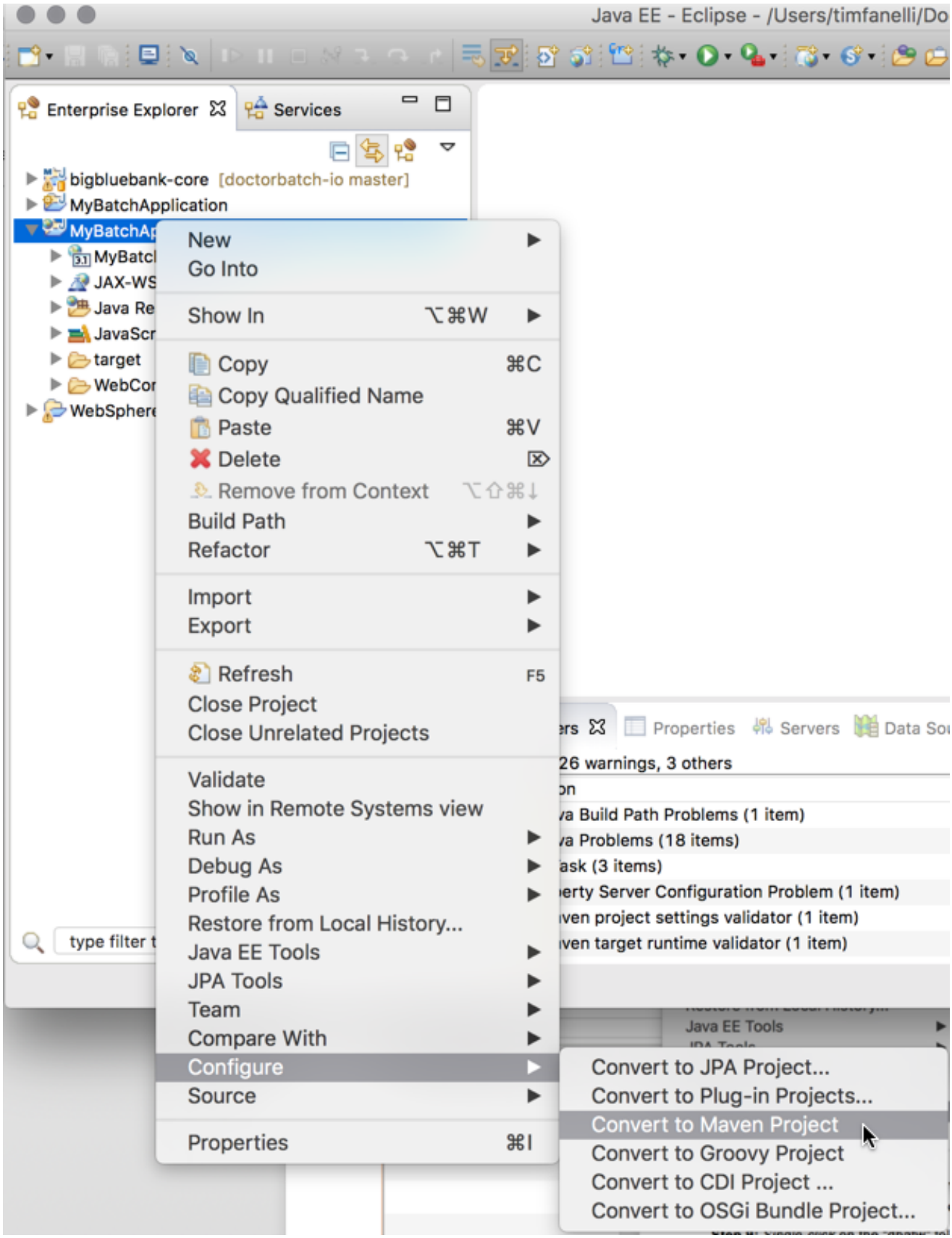
The Java EE Batch Project wizard knows to construct the required POJO and WAR projects, which are necessary to package and deploy the application to WebSphere Liberty Profile.

Import the doctorbatch.io **Dependencies**

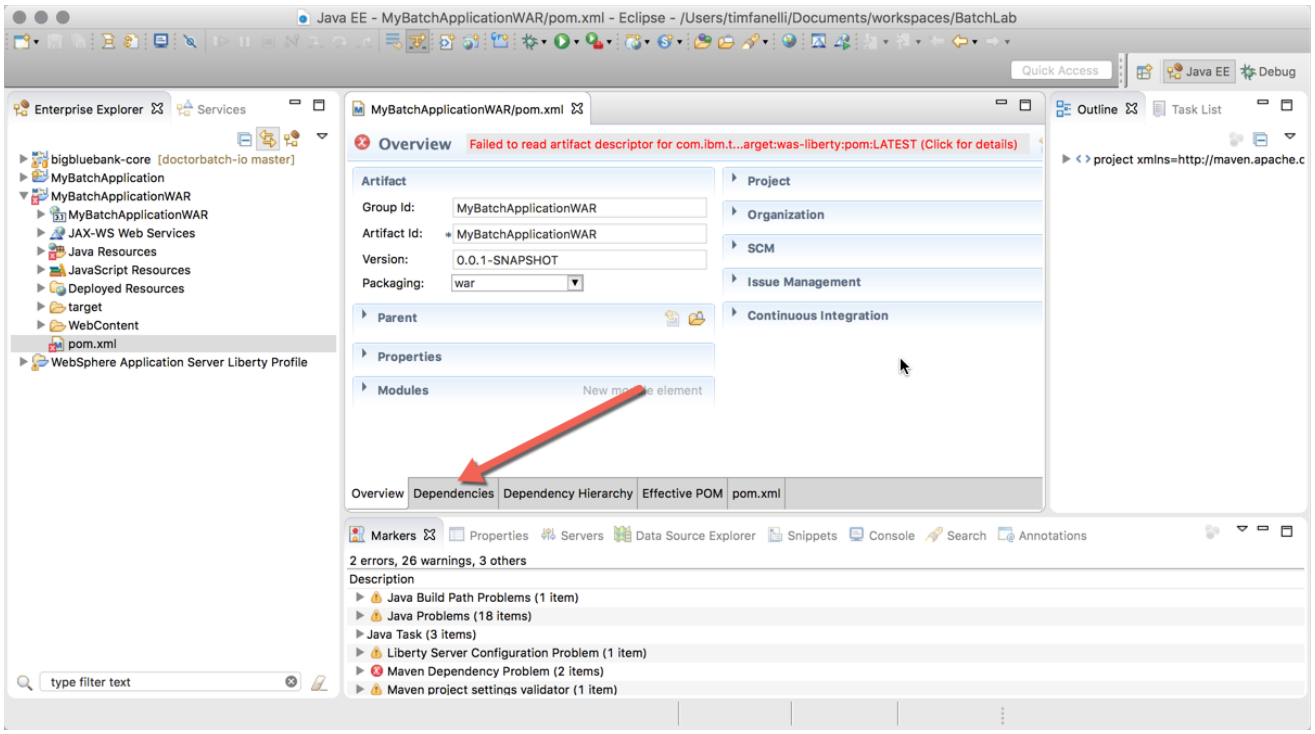
The doctorbatch.io library is a separate library, which is not included with either WebSphere or Eclipse, so we must manually import it into our batch project, and set up some meta-data so that it gets deployed with the application.

One very common way to manage third party dependencies is to use Apache Maven. **doctorbatch.io** is readily available in Maven Central, which means Maven can manage the dependency for us.

S**tep 1:** Right click your MyBatchApplicationWAR project, and select Configure → Convert to Maven Project

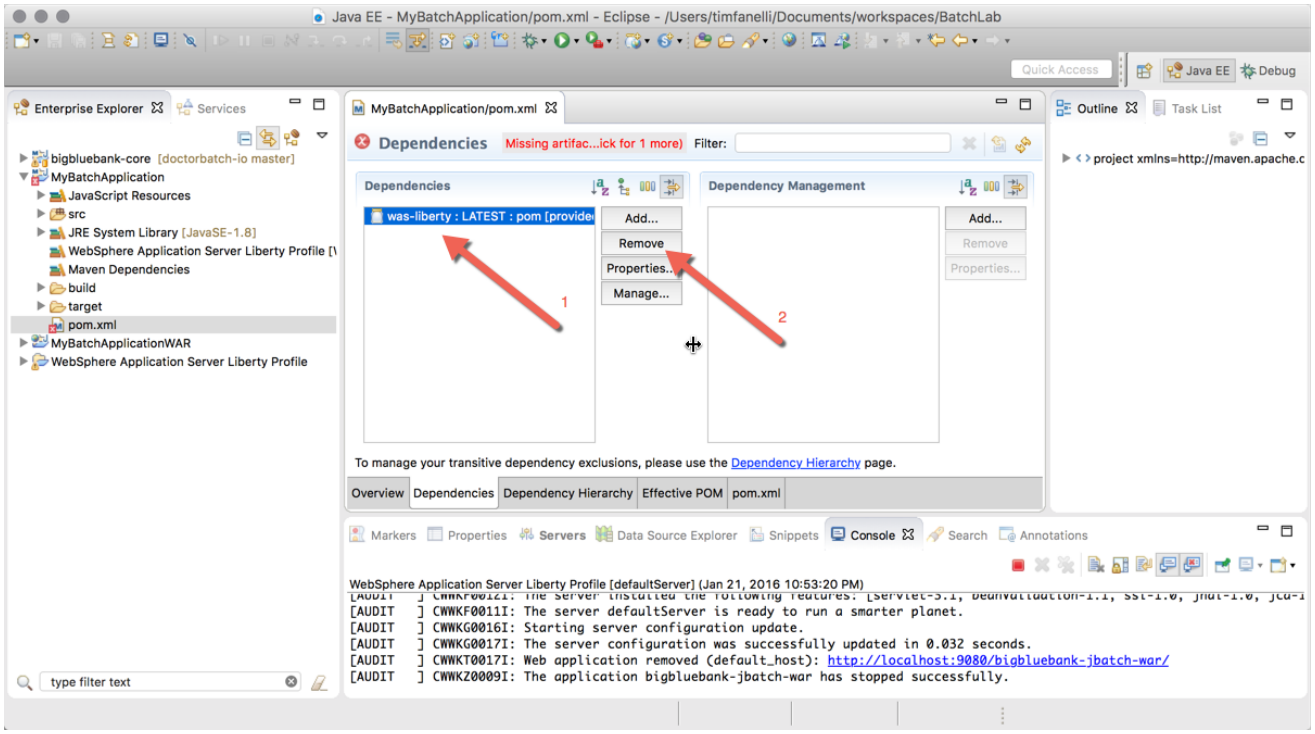

Java EE - Eclipse - /Users/timfanelli/Do
Enterprise Explorer
Services
bigbluebank-core [doctorbatch-io master]
MyBatchApplication
target
New
Go Into
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context ⌥⇧⌘↓
Build Path
Refactor ⌥⌘T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
JPA Tools
Team
Compare With
Configure
Source
Properties ⌘I
Properties
Servers
26 warnings, 3 others
Build Path Problems (1 item)
Problems (18 items)
Server Configuration Problem (1 item)
project settings validator (1 item)
target runtime validator (1 item)
Java EE Tools
Convert to JPA Project...
Convert to Plug-in Projects...
Convert to Maven Project
Convert to Groovy Project
Convert to CDI Project ...
Convert to OSGi Bundle Project...

Step 2: In the dialog that appears, click “Finish”. You will now see the new “pom.xml” file open in the editor panel.



Step 3: Click the “Dependencies” tab (pointed out above)

Step 4: Highlight the “was-liberty” dependency, and click “Remove”



Step 5: Next to the “Dependencies” list, click “Add…”

In the dialog that appears, enter the following values:

Group Id	**com.zblservices.doctorbatch**
Artifact Id	doctorbatch-io-core
Version	1.0.0

Step 6: Repeat Step 5 for both of the following sets of values:

Group Id	**com.zblservices.doctorbatch**
Artifact Id	doctorbatch-io-javabatch
Version	1.0.0

Group Id	**com.zblservices.doctorbatch**
Artifact Id	doctorbatch-io-javabatch-mvs
Version	1.0.0

Step 7: Save the file.

This added the **doctorbatch.io** library dependencies to your project.

Set the Java Build Path

The next step is to add bigbluebank-core dependency to the Java Batch project's build path, so that the Java compiler can resolve references to the Java code inside the libraries.

Step 1: Right click the MyBatchApplication project, and choose Properties

Step 2: On the left, choose "Java Build Path"

Step 3: Click on the "Projects" tab

Step 4: Click "Add …"

Step 5: In the selection dialog, check the box next to "bigbluebank-core".

Step 6: Click OK

Step 7: Click OK again

The bigbluebank-core libraries is now part of the compilation path for the MyBatchApplication project.; however, we still have two more tasks to make sure they are deployed with the application.

Adding the Dependencies to the WAR

The bigbluebank-javabatch-war project is the bundle that is deployed to the WebSphere Liberty Profile. When it was created, the bigbluebank-javabatch project was automatically added to it, and we already added the **doctorbatch-io** libraries using Maven. Now, we need to add our bigbluebank-core dependency as well.

Step 1: Right-click the bigbluebank-javabatch-war project, and choose Properties

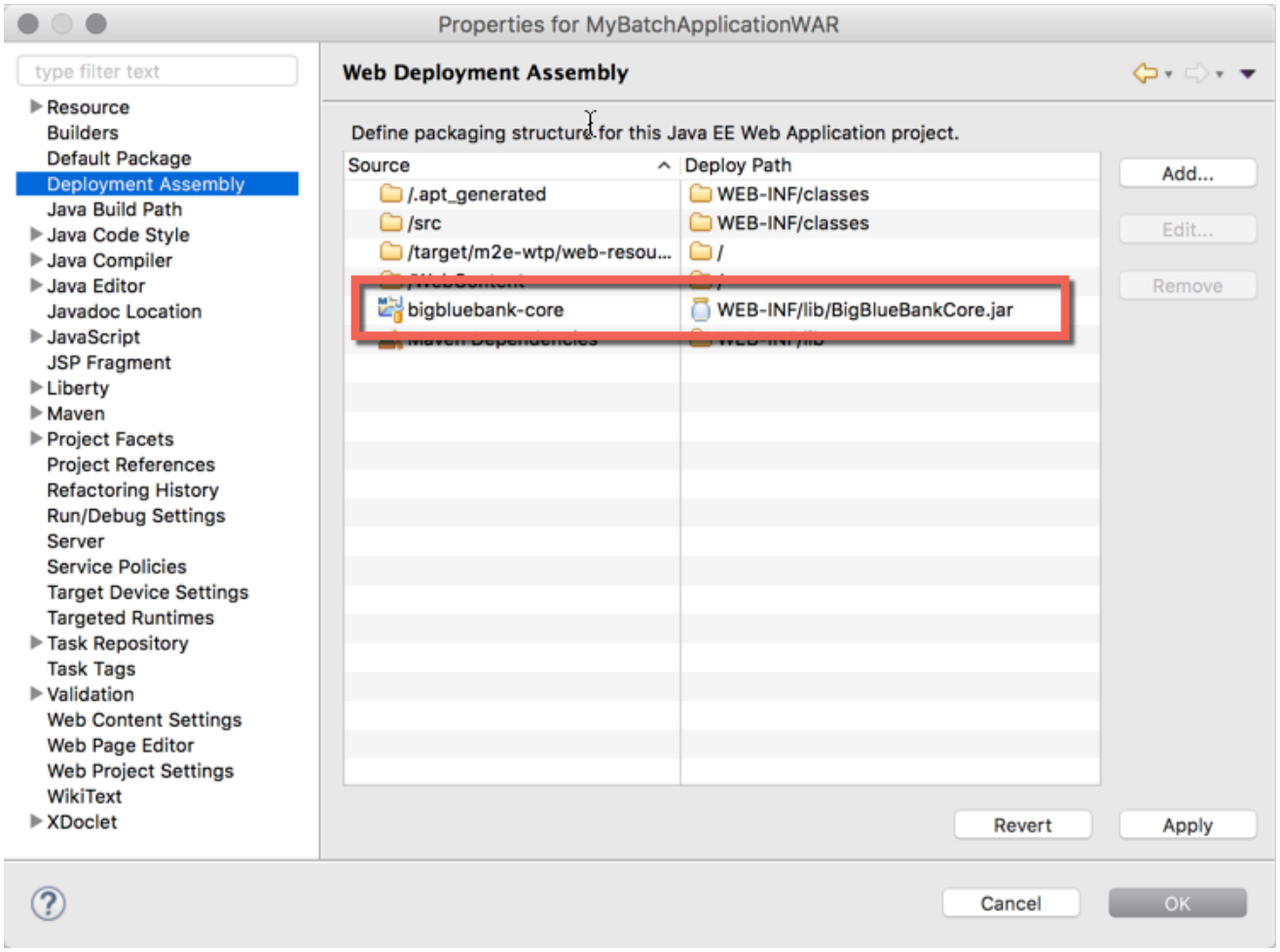
Step 2: On the left, select “Deployment Assembly”

Step 3: Click Add…

Step 4: Select “Project,” then click Next

Step 5: Highlight *bigbluebank-core*, then click Finish

You will see bigbluebank-core with a deploy path of WEB-INF/lib/BigBlueBankCore.jar



Step 6: Click OK

Load the COBOL Copy Book Layout

With the dependencies in place, there is one last step before we can create our batch project. Our application reads and processes deposit records, which are stored in a z/OS sequential dataset. The record layout looks like this:

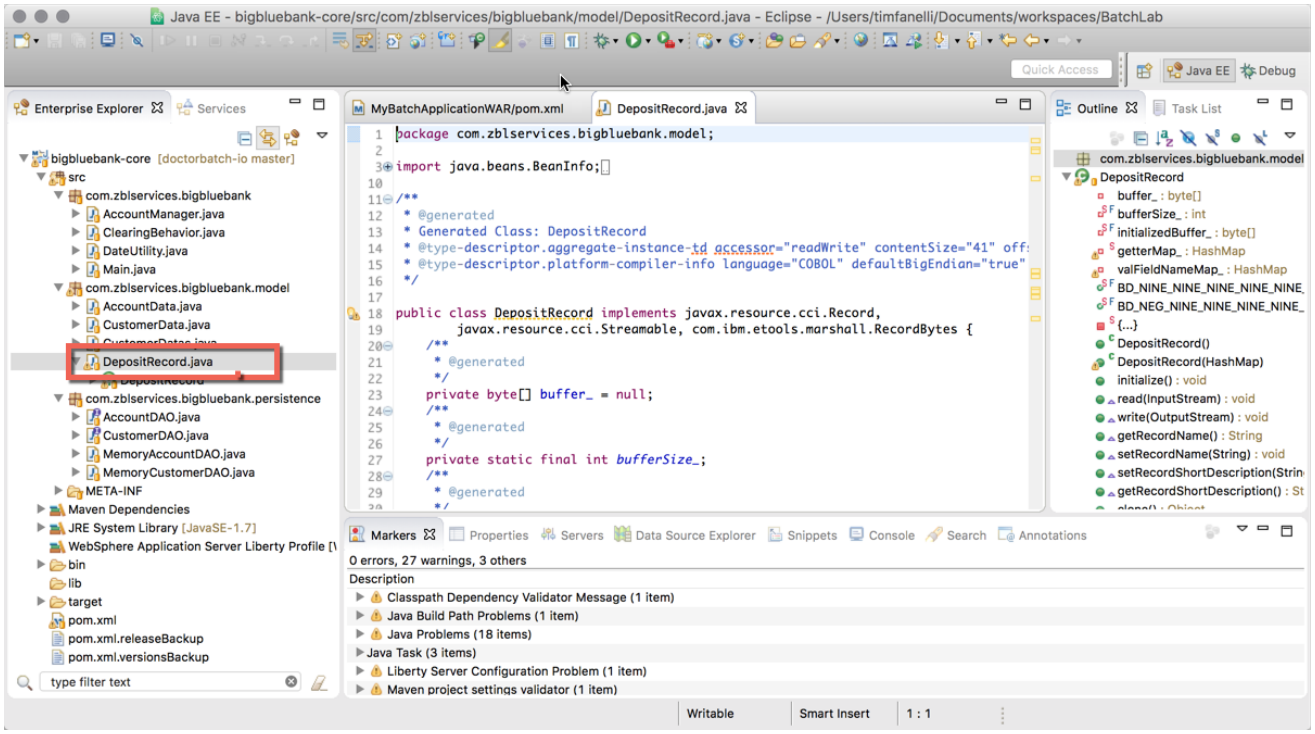
```
01 BBB-DEPOSIT-REC.
    05 CUSTOMER-ID  PIC X(10).
    05 ACCOUNT-NO   PIC X(11).
    05 DEPOSIT.
        10 AVAIL-IMM    PIC S9(5)V99.
        10 AMOUNT       PIC S9(5)V99.
    05 PROCESS-STATUS   PIC X(1).
        88 UNPROCESSED value 'U','u'.
        88 CLEARED     value 'C','c'.
        88 ERR         value 'E','e'.
    05 ERROR-CODE   PIC X(5).
```

We need to create a Java class that can:

1. Understand and parse the record layout of the sequential dataset
2. Perform EBCDIC to UTF-16 code page translation of PIC X data
3. Perform BCD to Java BigDecimal translations of the numeric data

IBM's JZOS library contains a utility called the Record Field Generator, which creates this Java class for us. Use of this tool is out of scope for this lab, but if you are interested in learning more, please let us know!

For now, you can inspect the DepositRecord class in the bigbluebank-core project.



Creating a Batch Job

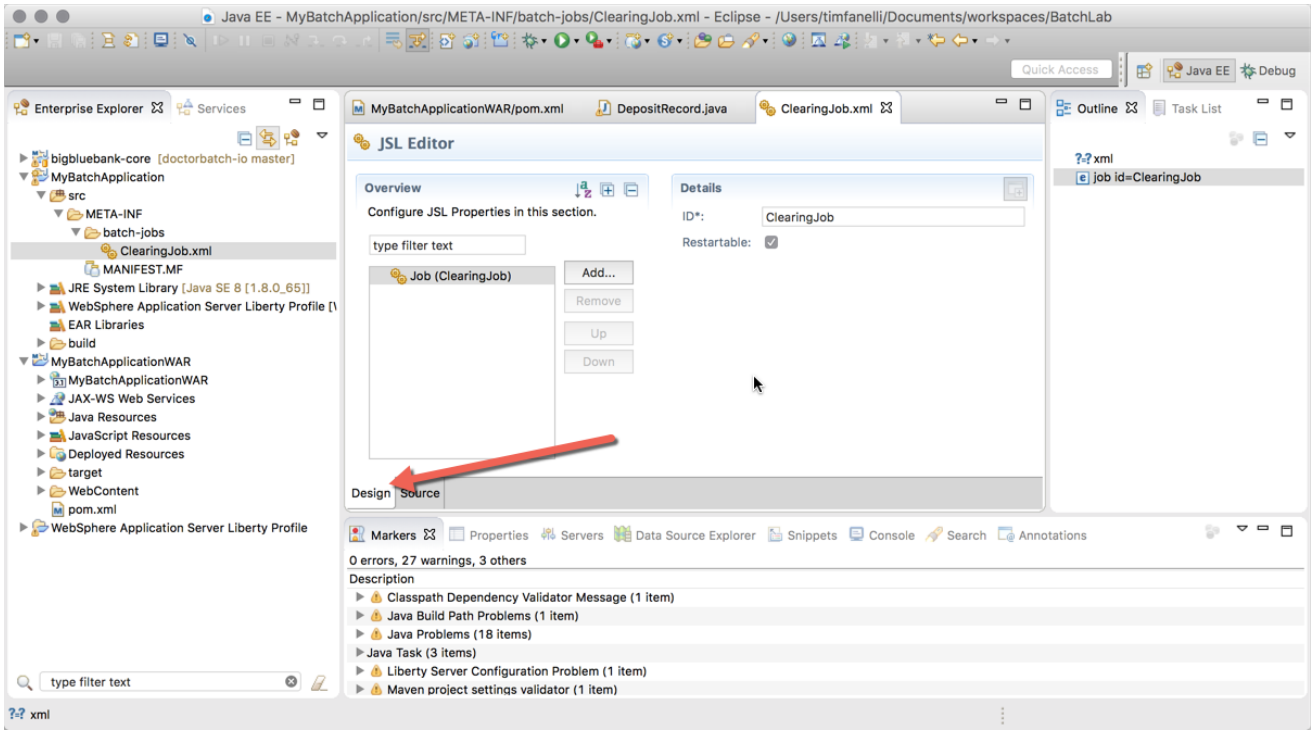
Now that we have all our dependencies in place, we're ready to actually create the batch job!

Step 1: Right-click the MyBatchApplication project, and choose New → Batch Job

Step 2: In the dialog that appears, enter a Job Name:
ClearingJob

Step 3: Click Finish

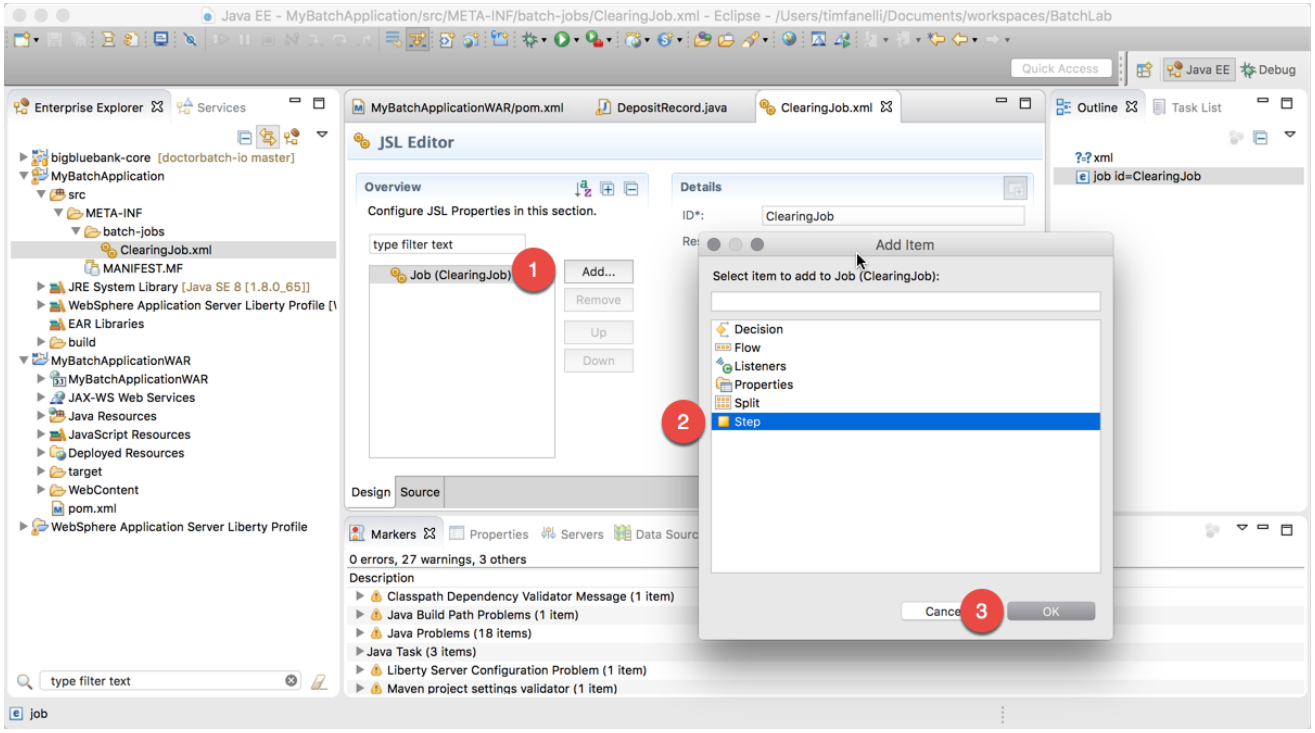
You will now see the ClearingJob.xml file in the editor panel. This file is the Job Specification Language, or JSL, document. JSL orchestrates the flow of a Java Batch job.



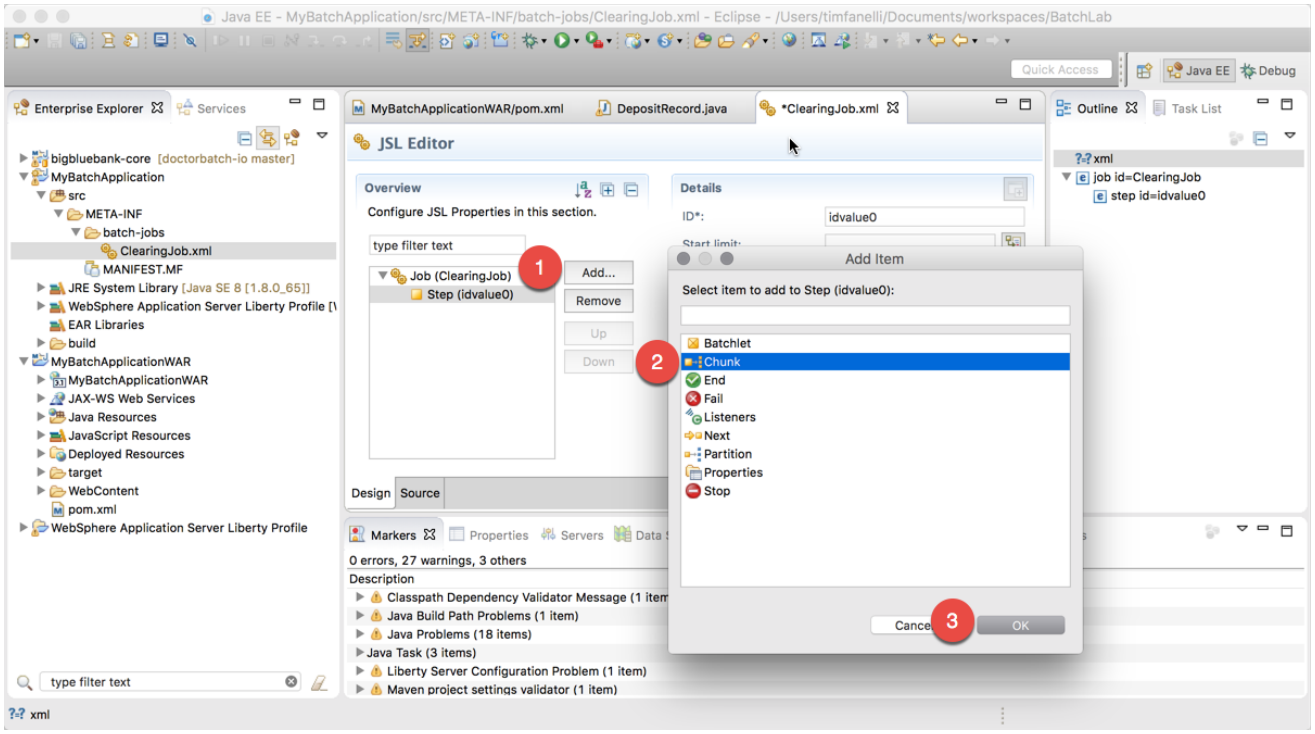
NOTE: If you see raw XML in your editor, click on the “Design” tab, as shown above.

Now we’ll add a “Chunk” style batch step to our job. A “Chunk” step, as we saw earlier, uses a single Reader, Processor, and Writer to implement an ETL style batch workflow.

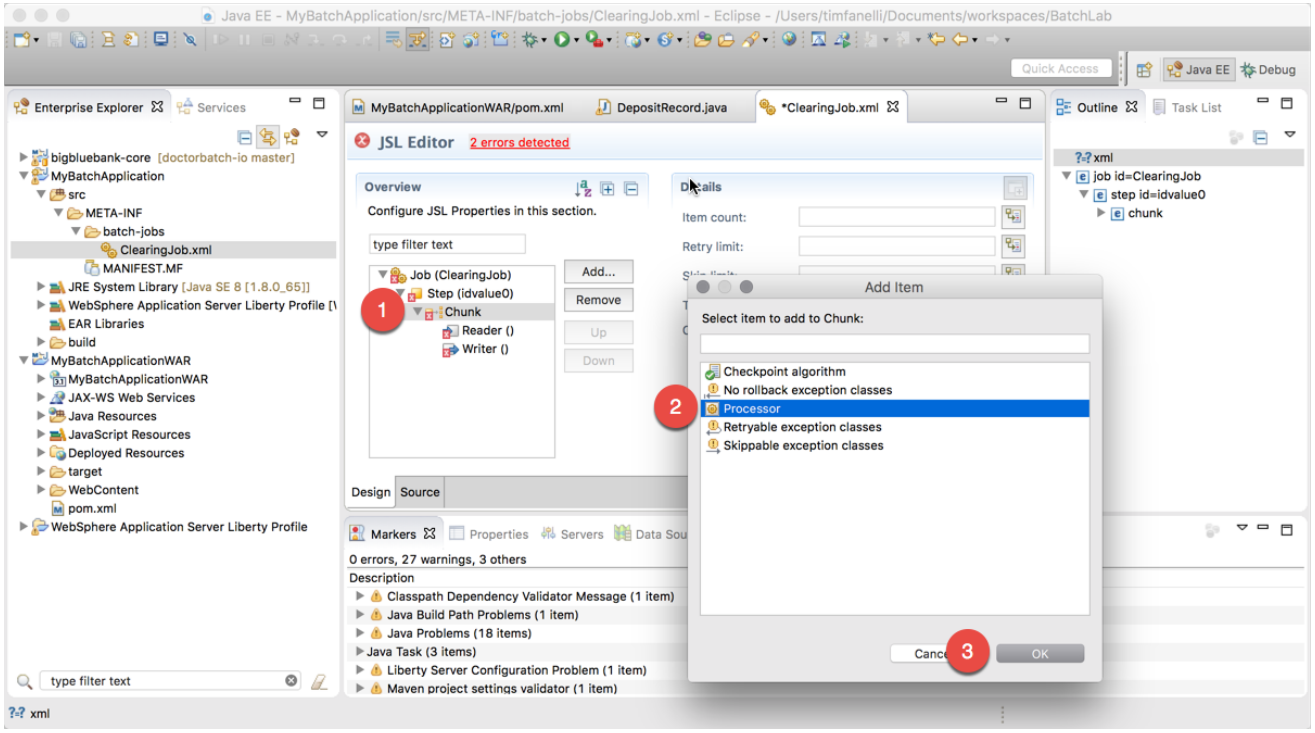
Step 4: In the Design view, click the “Add” button, choose “Step”, and then click OK



Step 5: Click Add again, then choose Chunk, and click OK

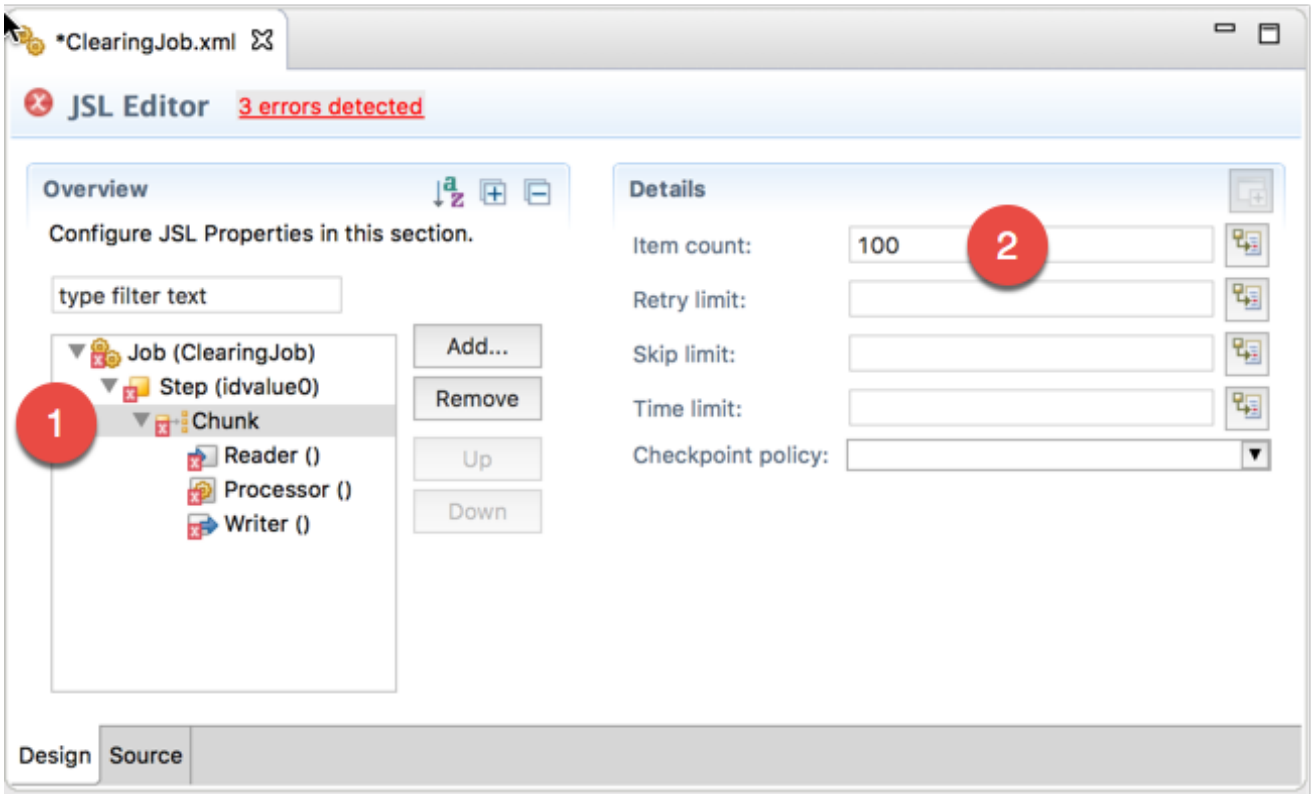


Step 6: Click add one more time, now choose “Processor”, and click OK.

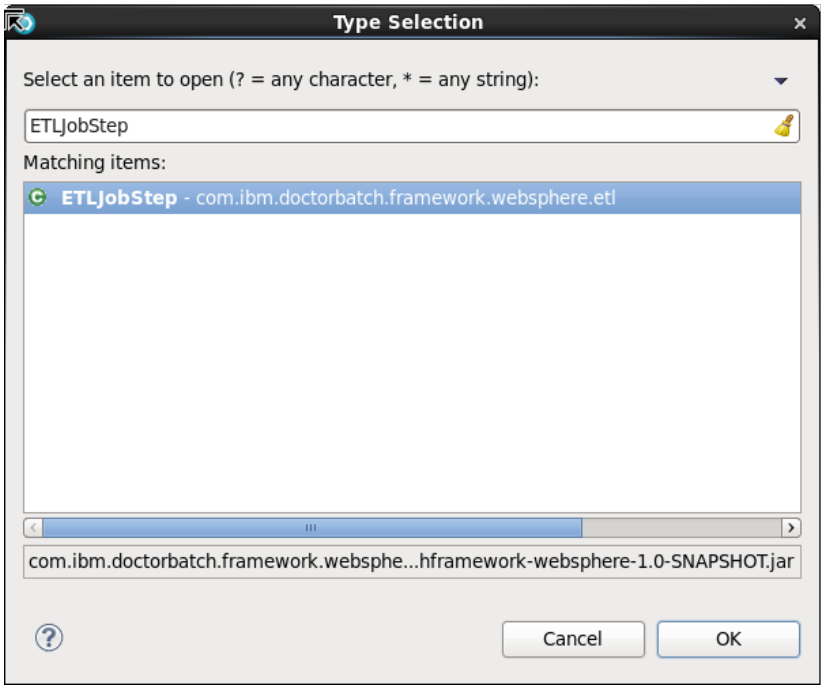


Now that we have a Chunk step, we can set the properties of the Chunk, and its Reader, Processor, and Writer elements.

Step 7: In the Overview of the JSL editor, highlight the “Chunk” element in the tree, then enter “100” in the “Item Count” field.



Step 8: Select ETLJobStep from the “Matching items” list, and click OK



This tells the batch job that we want to use the DBAFW’s ETLJobStep class as our job step implementation.

Remember that the ETLJobStep implementation requires several properties, which we will add here.

Step 9: Next, highlight the “Reader ()” element. In the reference field, enter:

```
com.zblservices.doctorbatch.io.mvs.RDWRecordItemReader
```

Step 10: Click the “Add” button, and choose Properties

Step 11: Highlight the “Properties” element in the Overview tree

Step 12: Click the “Add” button again, and choose “Property”. When prompted for the Property Name, enter the fiirst property name from the table below, and then click OK

Step 13: You will now see the “Details” for the property. Enter the corresponding Value from the table below

	PROPERTY NAME	VALUE
1	RECORD_LENGTH	41
2	RECORD_PARSER_CLASSNAME	com.zblservices.doctorbatch.io.mvs.RecordBytesParser
3	MVS_RECORDBYTES_CLASSNAME	com.zblservices.bigbluebank.model.DepositRecord
4	FILE_NAME	/tmp/depositRecordData.dat

Step 14: Repeat steps 10 through 13 for each of the properties in the table.

You will complete the series of steps (steps 7 through 14) to setup the Writer, and the Processor, using the values shown here:

Writer:

Reference: `com.zblservices.doctorbatch.io.mvs.RDWRecordItemWriter`

	PROPERTY NAME	VALUE
1	RECORD_LENGTH	41
2	RECORD_PARSER_CLASSNAME	com.zblservices.doctorbatch.io.mvs.RecordBytesParser
3	MVS_RECORDBYTES_CLASSNAME	com.zblservices.bigbluebank.model.DepositRecord
4	FILE_NAME	/tmp/depositRecordDataOutput.dat

Processor:

Reference: `com.zblservices.doctorbatch.io.SimpleProcessor`

	PROPERTY NAME	VALUE
1	RECORD_PROCESSOR	com.zblservices.bigbluebank.ClearingBehavior

That’s it! You just created a brand new WebSphere Batch Job!

But Wait, I Didn't Write Any Code?

That's almost correct. The doctorbatch.io library makes it so that you have to actually write *very little* if any Java code.

We reused the already-existing ClearingBehavior class as our Record Processor to the SimpleProcessor step implementation. This was to save you the pain of actually having to write any Java Code! However, had we not done that, this is all the code you would have actually written.

You can go edit that class now, if you like, so you can see your own changes before we run the application.

If you're not a Java developer, please just ignore this part – it won't be of interest to you anyway!

Step 1: Expand open the bigbluebank-core project

Step 2: Expand open the "src" folder, and the "com.zblservices.bigbluebank" package under it.

Step 3: Double click on the ClearingBehavior.java file, to open it in the editor.

Step 4: Scroll down to the process method, and make any changes you like to the business logic. It may be simplest to just add another logging statement.

Deploy and Execute Your Java Batch Project

You're on your own for this one, folks! Go back to Part 1 if you need a refresher, and don't be shy about raising your hands if you need assistance!

Good luck!

What's Next?

This application code is now ready to package and deploy to any of the following environments:

1. WebSphere Liberty Profile – as is
2. WebSphere Application Server Batch (aka WebSphere Computer Grid) – with a simple repackaging
3. z/OS for production readiness!

To learn more about the **doctorbatch.io** library, JSR-352, or Java Batch on z/OS – go to http://zblservices.com

That's It!

Good job! You just completed the development and testing of a Java EE Batch Job using the JSR-352 features available in IBM's WebSphere Liberty Profile. This is just the very basics. JSR-352

provides a very powerful, and performant environment for high volume, low latency, bulk data processing applications.