FUSE and Camel Workshop

RHTE 2014

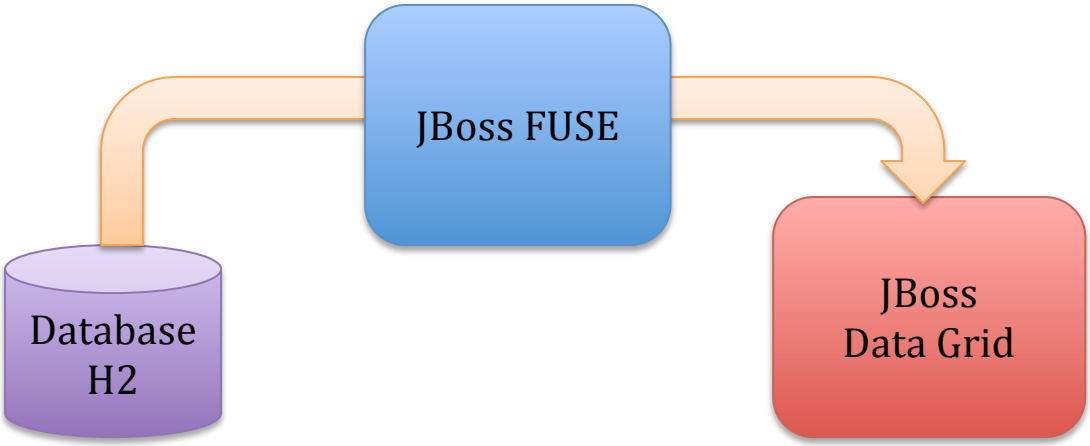
Christina Lin
JBoss Technology Evangelist

Index

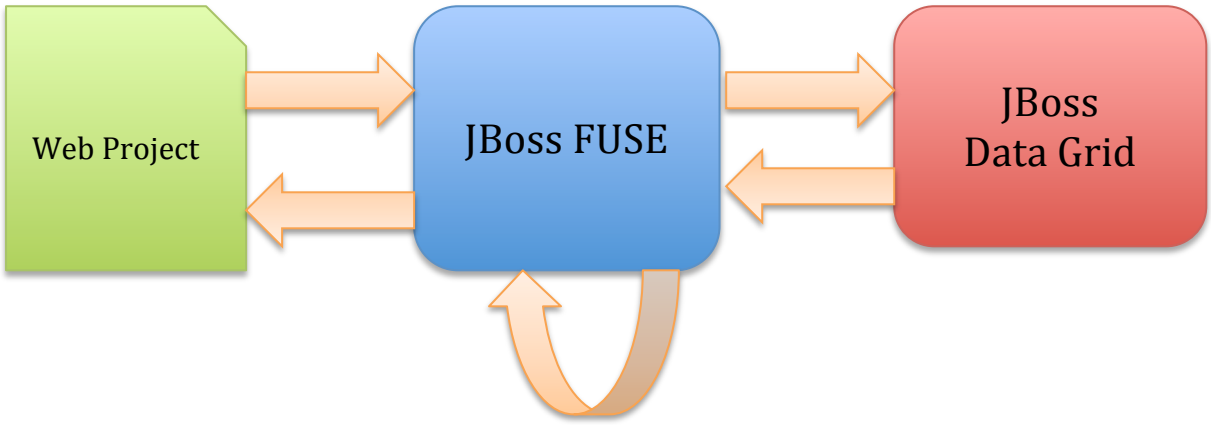
Introduction

Because we only have one hour, this lab is going to be a very simple demonstration to get you started on using several components and Enterprise Integration Pattern in Red Hat JBoss Fuse. This is a subset and simply version of a project call myWiri, which will be out shortly in July 2014 by Christina. Look forward to it! ☺

We are first going to load data from H2 database to Red Hat JBoss Data Grid, provide a faster elastic memory grid cache for user to look up.



Then we are going to build a web application with a interface for user to lookup data from the cache we have created. The communication will be asynchronized with WebSocket protocol with Fuse Camel route. In the route, it will determine if the requested content can be found in the cache. We will use the content routing integration pattern to reply the content that we are going to reply back.



Prerequisite

- This lab only runs in Linux and Mac. Please make sure you have the right platform.
  (Or you are always welcome to contribute Window script to this project ☺!)

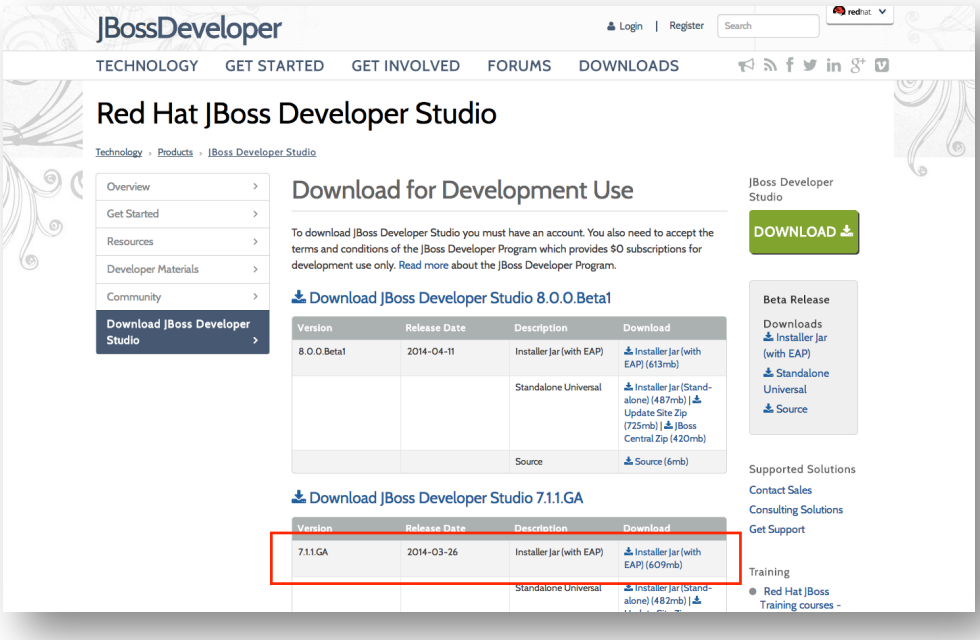
- Java Runtime Environment 6 or 7 and Maven 2.1+ should be already installed in your machine.

Installation

Install Red Hat JBoss Developer Studio

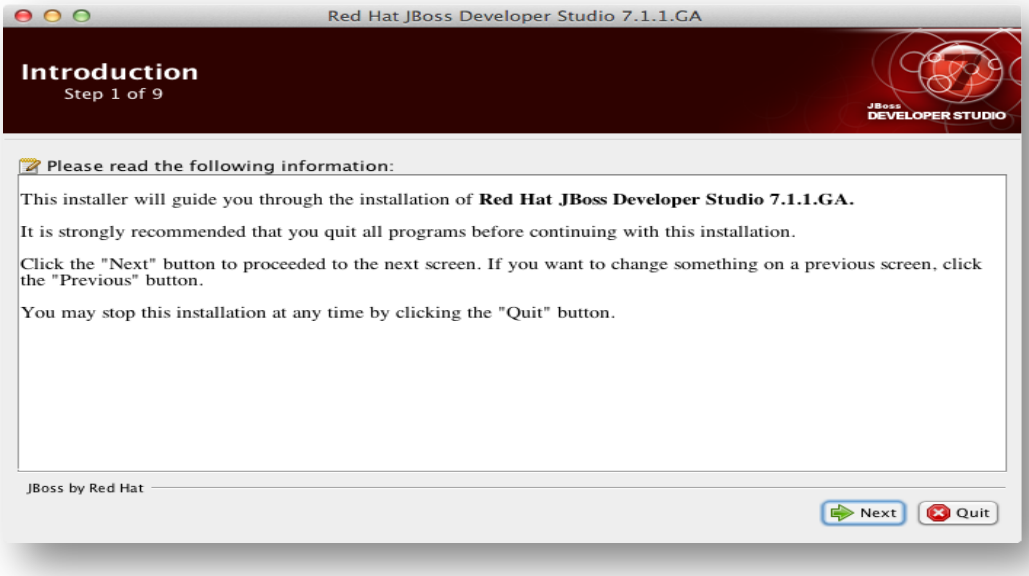
This lab will guide you through installing Red Hat JBoss Developer Studio and the tools for developing JBoss Fuse Integration Projects.

If you don't already have *Red Hat JBoss Developer Studio 7.1.1 GA*, download it from https://www.jboss.org/products/jbds.html.



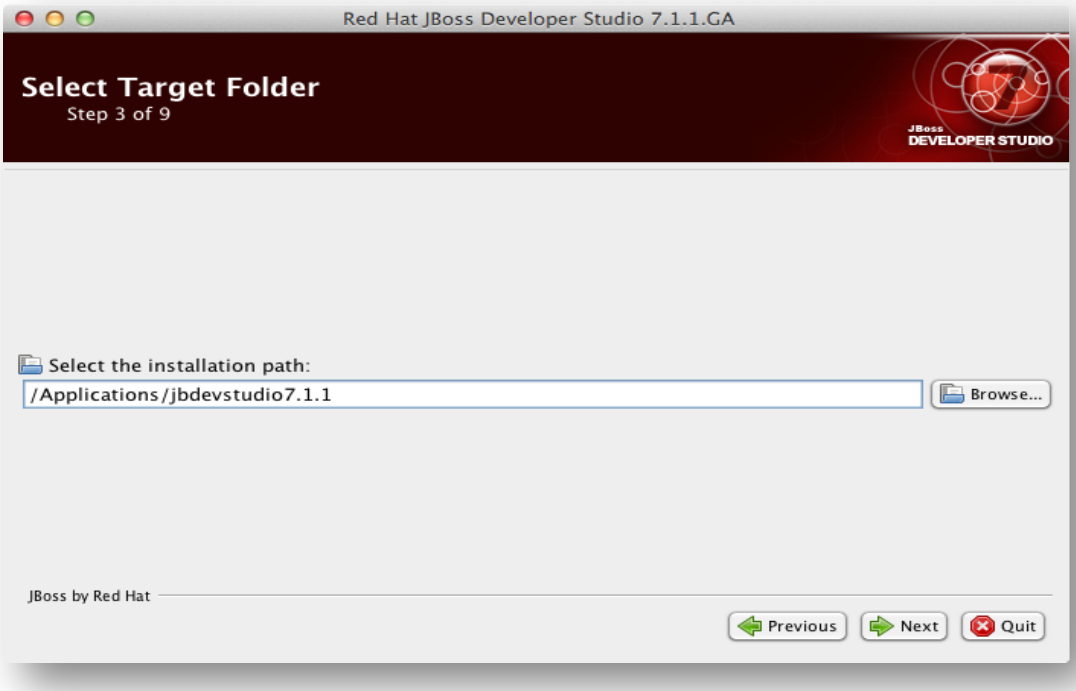
Run the JBDS installer using java at the command prompt or double click on the jar file.

```
java -jar {jbdevstudio- installer.jar}
```

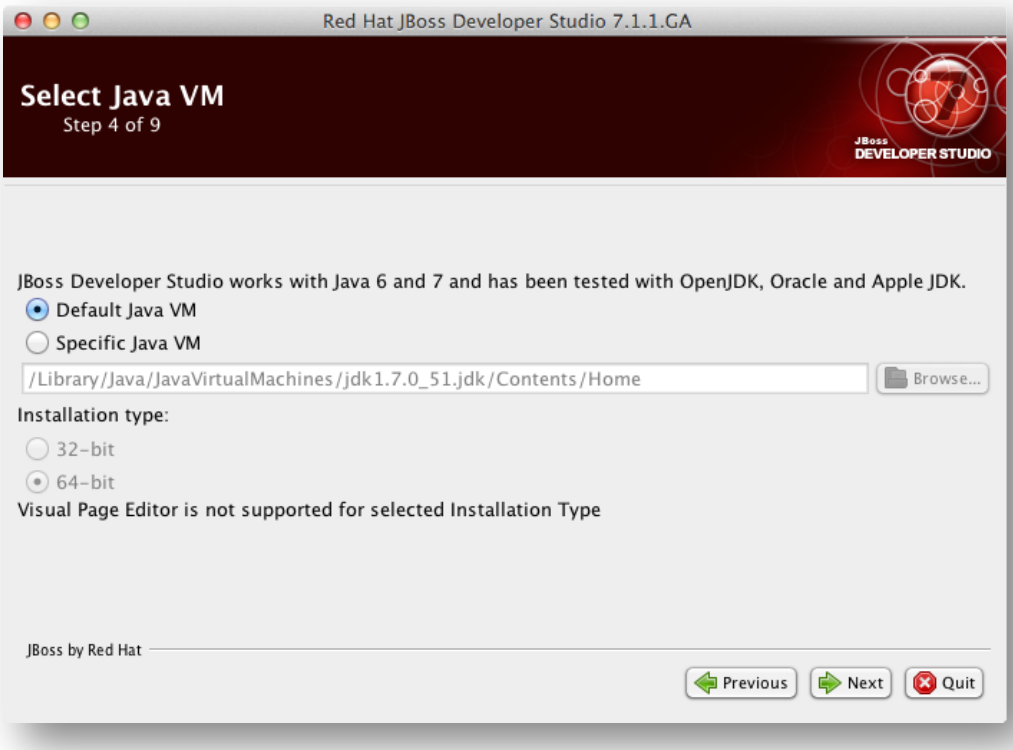


Follow the installer prompts to complete the installation process.

After reading and agreeing to the terms of the End User License Agreement, select I accept the terms of this license agreement and click *Next*.

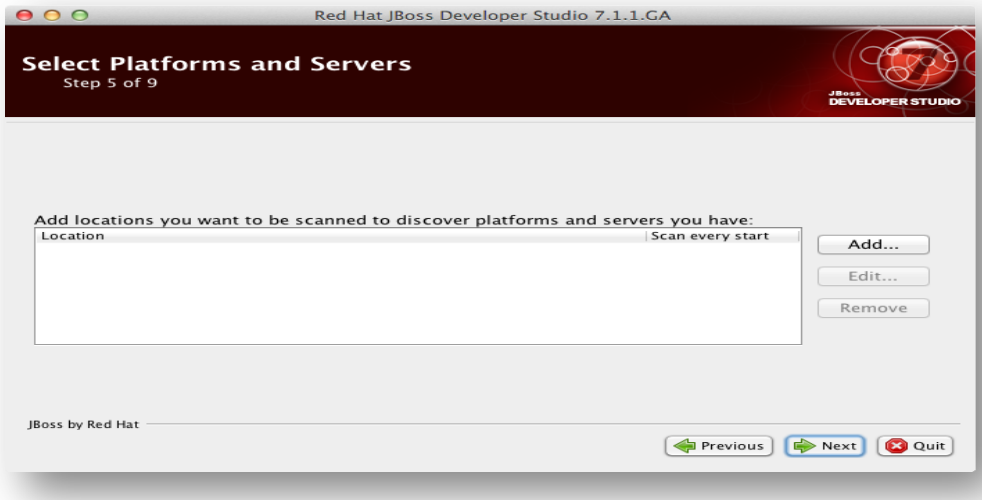


In the Select the installation path field, type the path where you want Red Hat JBoss Developer Studio to be installed or click Browse to navigate to the location. When the Select the installation path field shows the correct path, click Next.

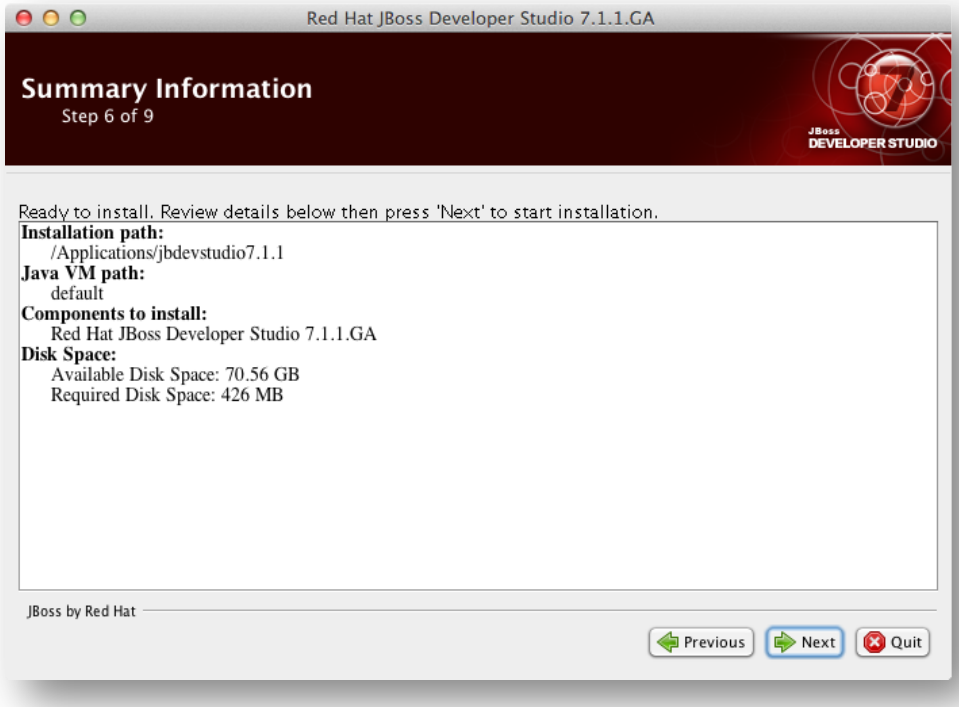


In the Select Java VM step, Default Java VM is automatically selected. Ensure that the

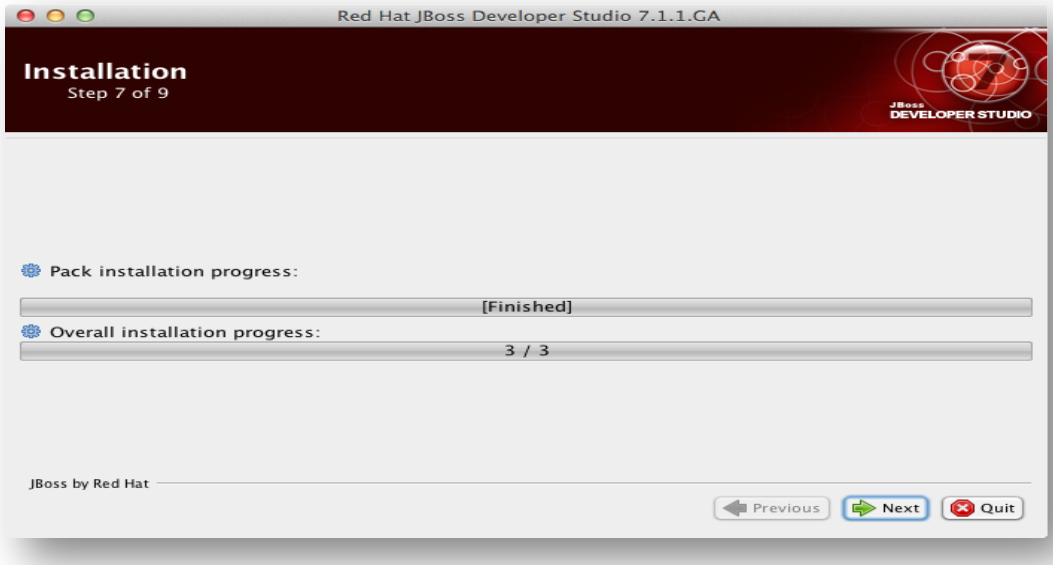
disabled text field contains the path of the Java developer kit you want to use. To change the specified Java developer kit, select Specific Java VM and type the path of the Java developer kit in the text field or use the Browse button to locate the Java developer kit. When the text field shows the correct Java developer kit path, click Next.



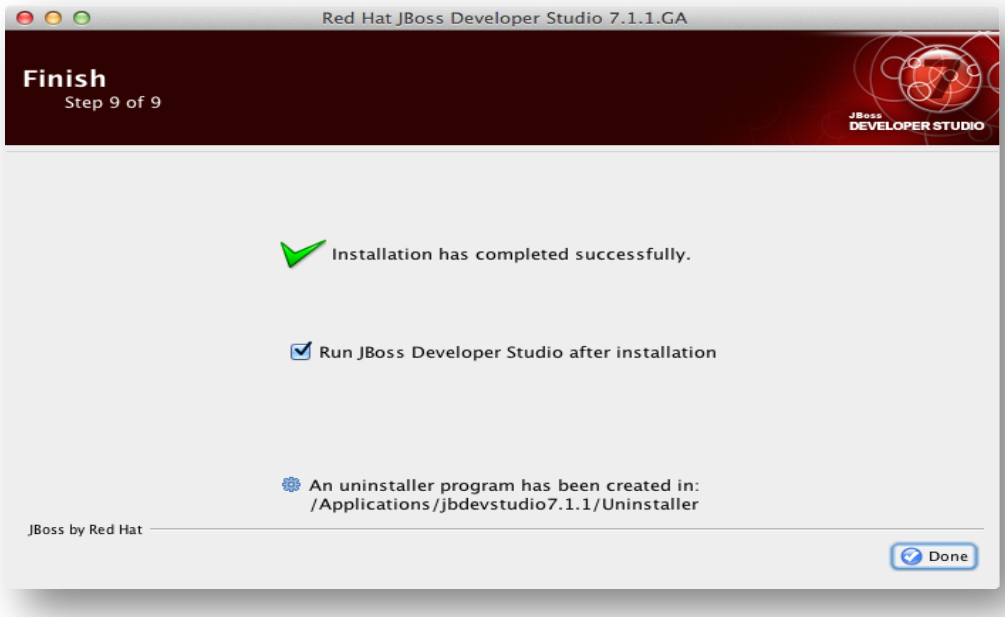
In the Select Platforms and Servers step one can add server to make use of automatic runtime detection for finding already installed application servers. Skip this for now since we will add servers later. Click Next to proceed.



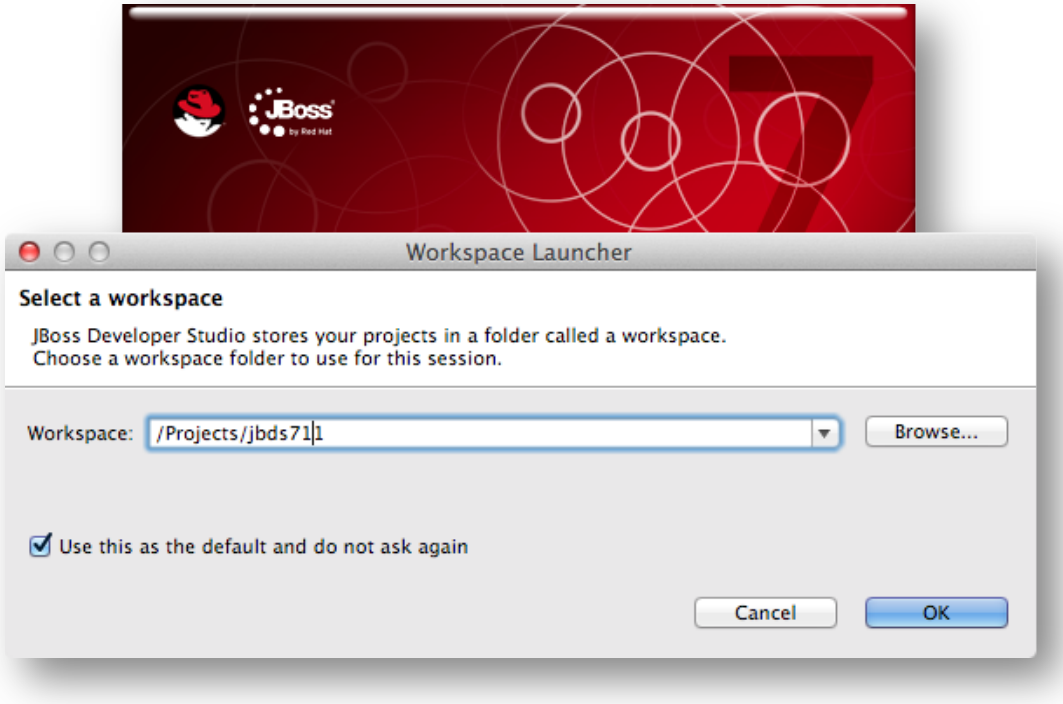
Review the details in the Summary Information window and, if they are correct, click Next. The installation commences.



When the installation progress bar shows Finished, click Next. The installation process is now complete.



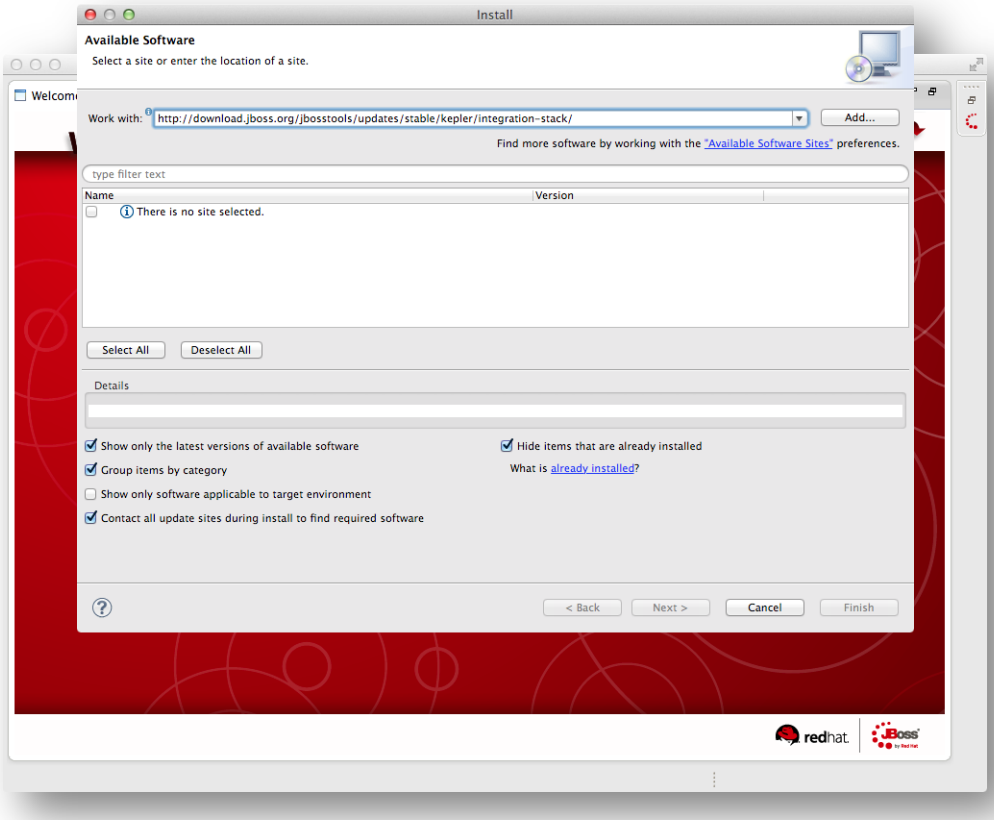
Click Done to close the Installer window. When the installation is completed, run Red Hat JBoss Developer Studio 7.1.1. When the Red Hat JBoss Developer Studio starts, you are asked to choose the workspace folder for the session. The workspace is where your projects are stored.



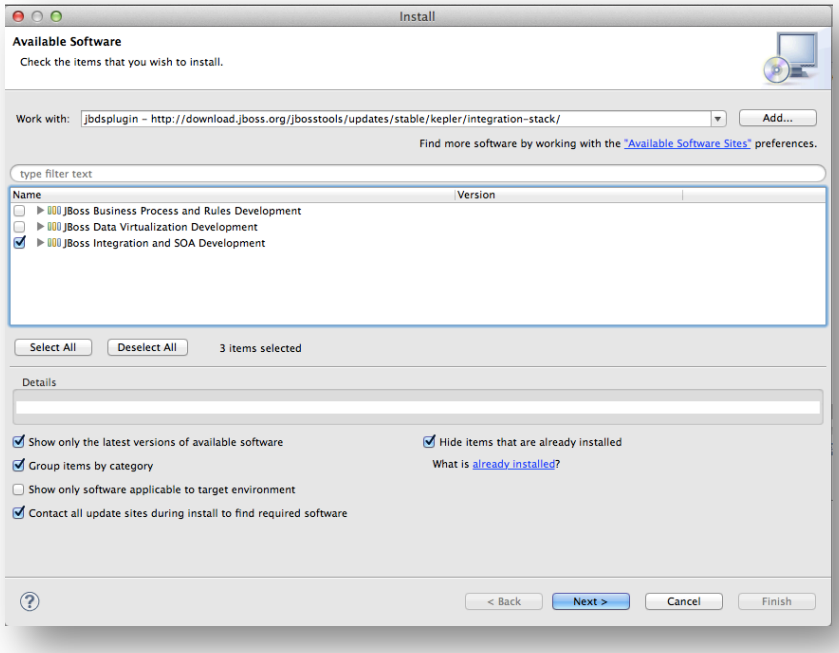
To set the workspace location, follow these steps:

1. In the Workspace field, type the path for a new or existing workspace or use Browse to navigate to the workspace location.
2. If you do not want to be asked to choose a workspace folder each time the IDE starts, select the Use this as the default and do not ask again check box.
3. Click OK.

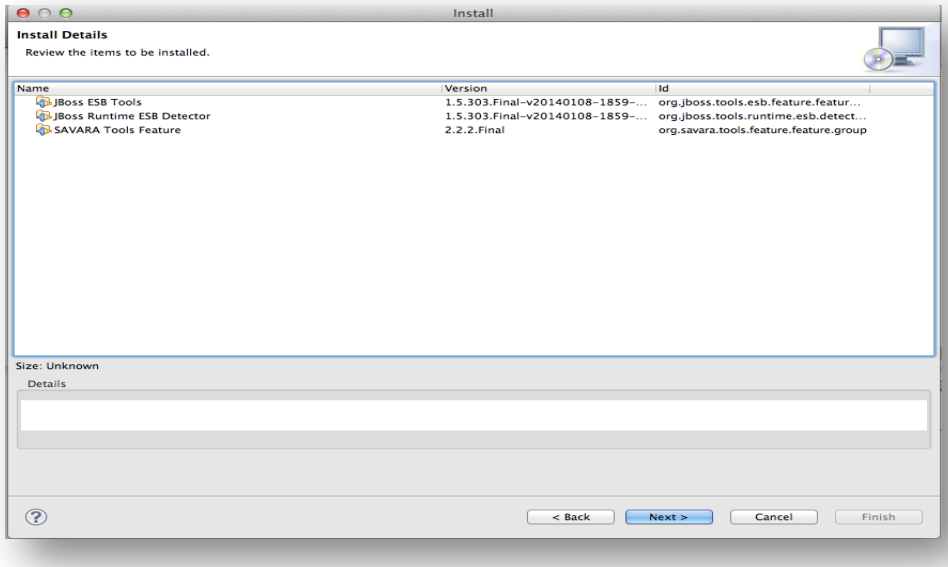
The workspace location prompting behavior can be altered at any time by clicking Window → Preferences. Expand General → Startup and Shutdown → Workspace. Select or clear the Prompt for workspace on startup check box to alter the behavior as appropriate. From the JBDS menu bar, choose Help → Install New Software...



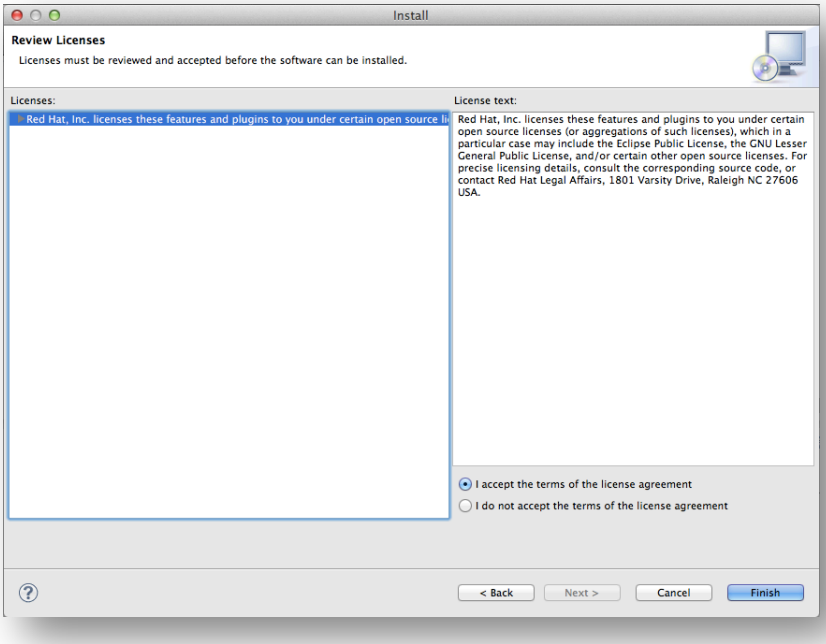
In the "Work with:" field on the Install wizard, paste the following link: http://download.jboss.org/jbosstools/updates/stable/kepler/integration-stack/ and click the Add... button.



Select JBoss Integration and SOA Development option in the list and click Next.



Review Install Details and click the Next > button.

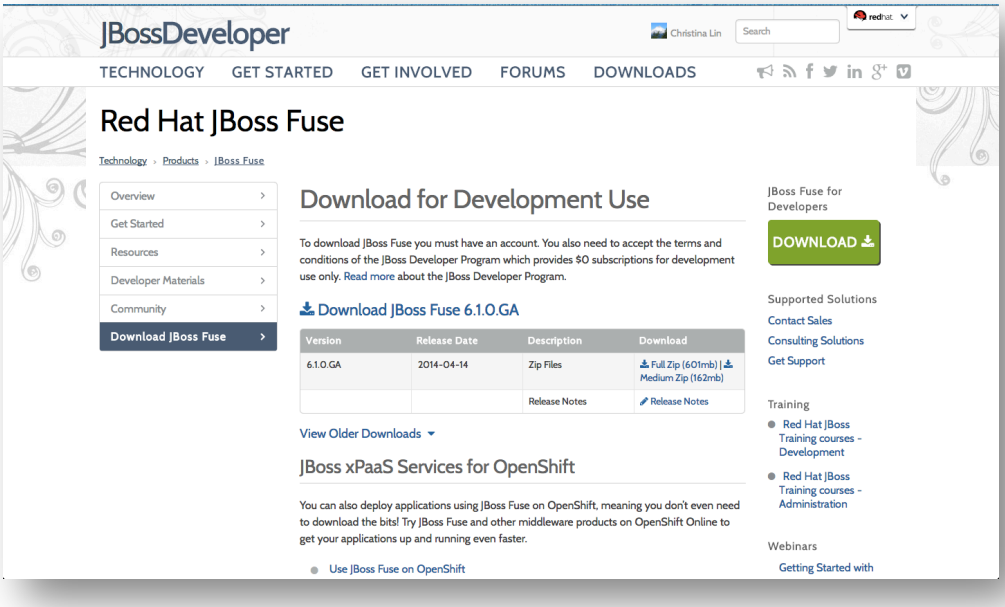


The Security Warning Window will appear and click the OK button to proceed.

The Software Updates window appears. Press the Yes button to restart Red Hat JBoss Developer Studio to apply the changes to take effect.

Install Red Hat JBoss Fuse

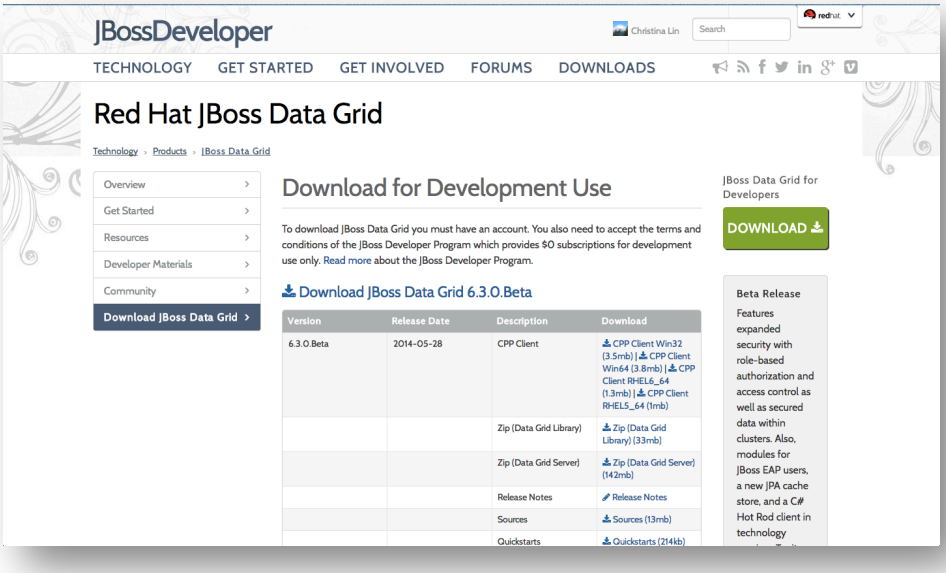
If you don't already have *Red Hat JBoss Fuse 6.1.0 GA*, download it from https://www.jboss.org/products/fuse.html.



The installation will be done for you in the lab. But if you would like further installation guideline, please go to https://access.redhat.com/site/documentation/en-US/Red_Hat_JBoss_Fuse/6.1/html/Installation_Guide/files/front.html, for more details.

Install Red Hat JBoss Data Grid

If you don't already have *Red Hat JBoss Data Grid 6.2.0 GA*, download it from https://www.jboss.org/products/datagrid/download/ .



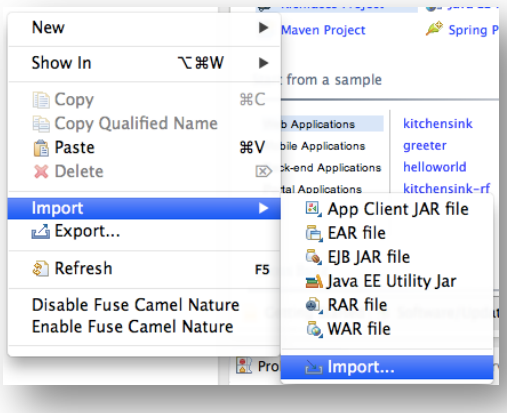
The installation will be done for you in the lab. But if you would like further installation guideline, please go to https://access.redhat.com/site/documentation/en-US/Red_Hat_JBoss_Data_Grid/6.2/html/Getting_Started_Guide/, for more details.

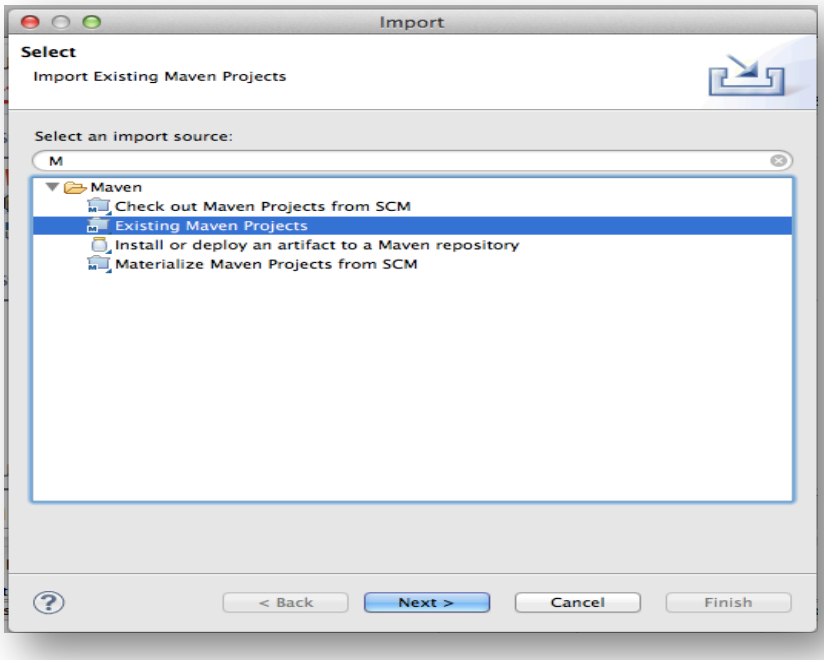
Lab 1

Environment

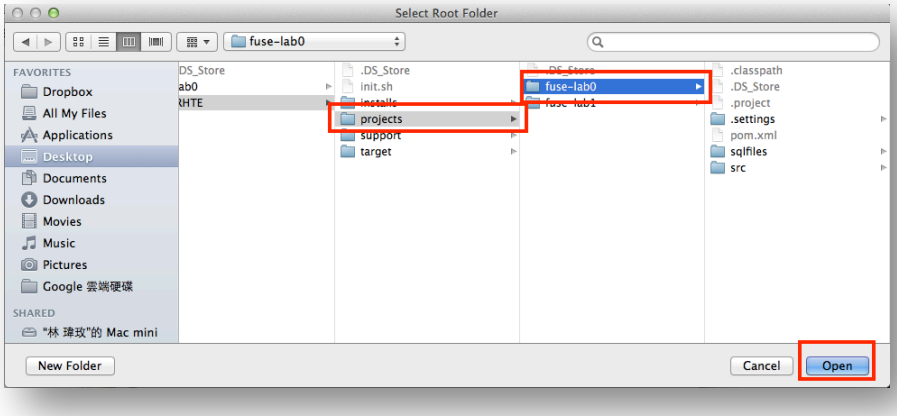
We will be working under fuse-lab0 directory.

1. First we are going to load project into JBDS. Open up JBDS, and import an existing maven project.

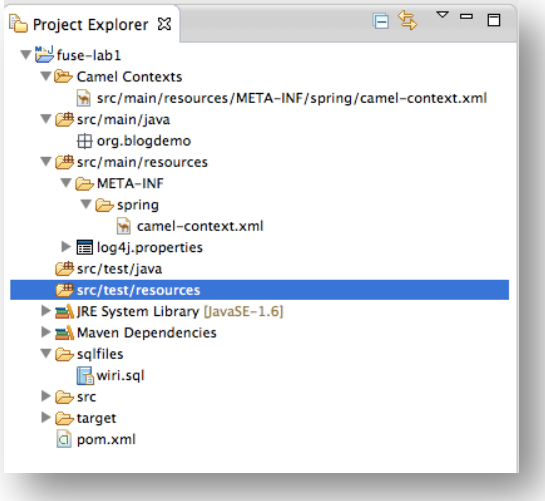




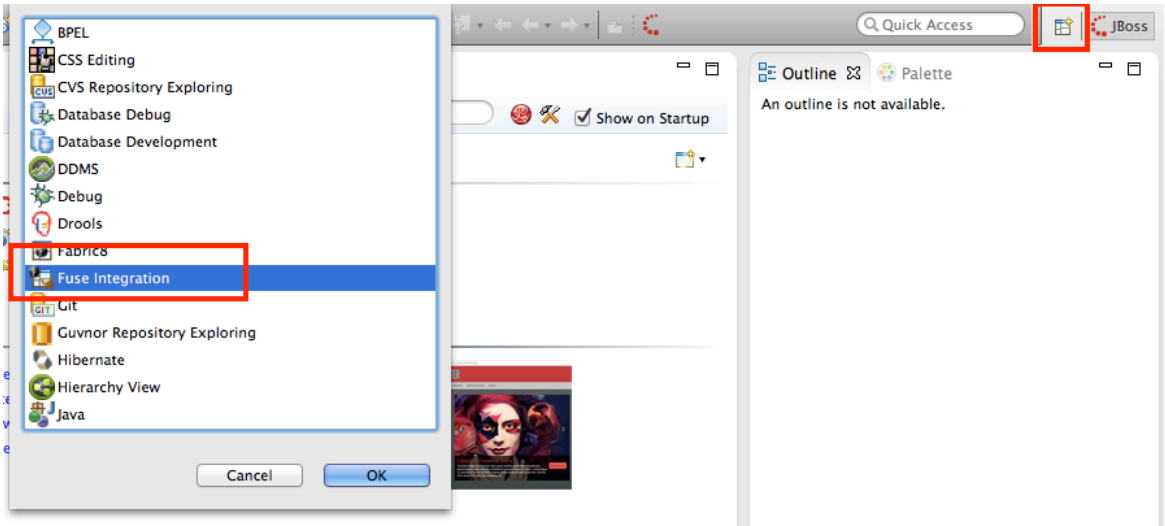
Go to the first project directory, fuse-lab0



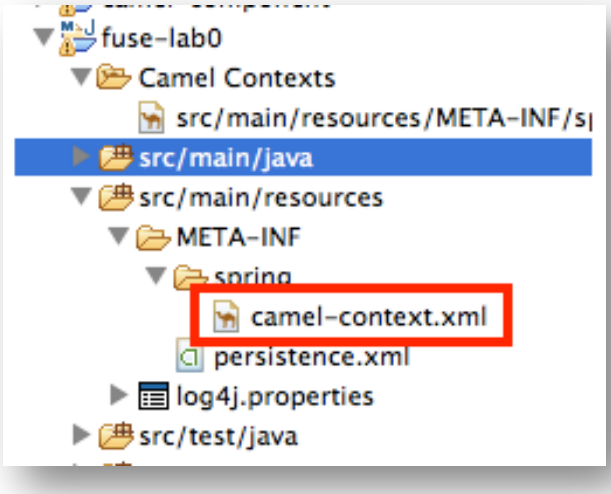
You will see the the project loaded into the console.



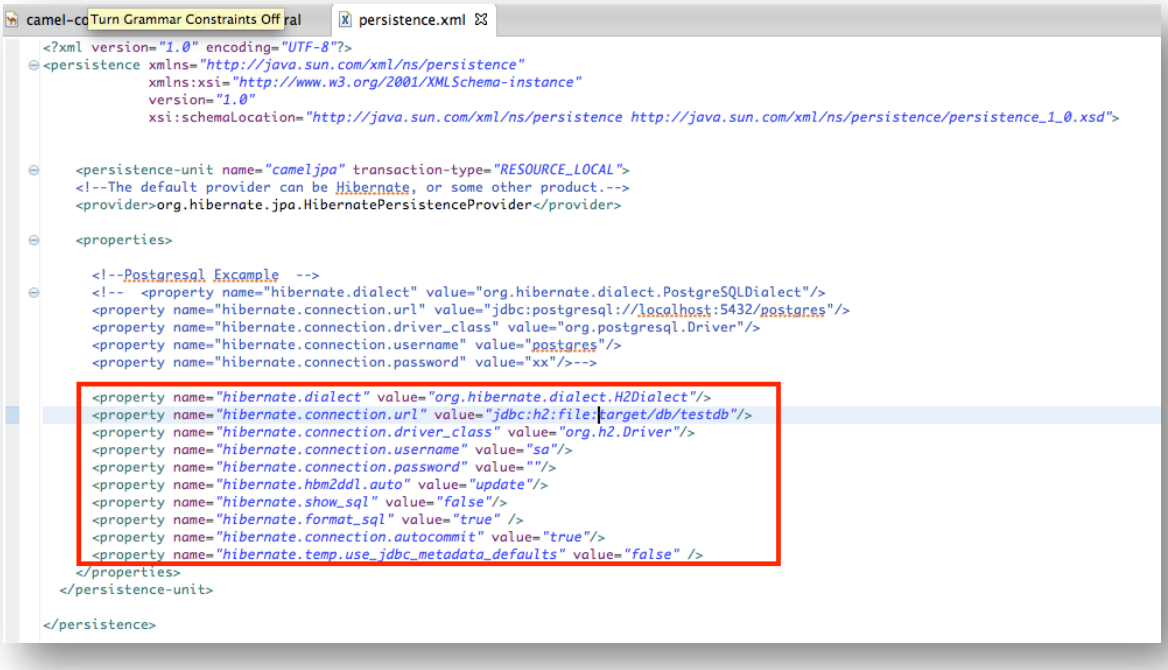
If your JBDS does not go to FUSE perspective, go to the upper right corner, click on the + , then select Fuse Integration.



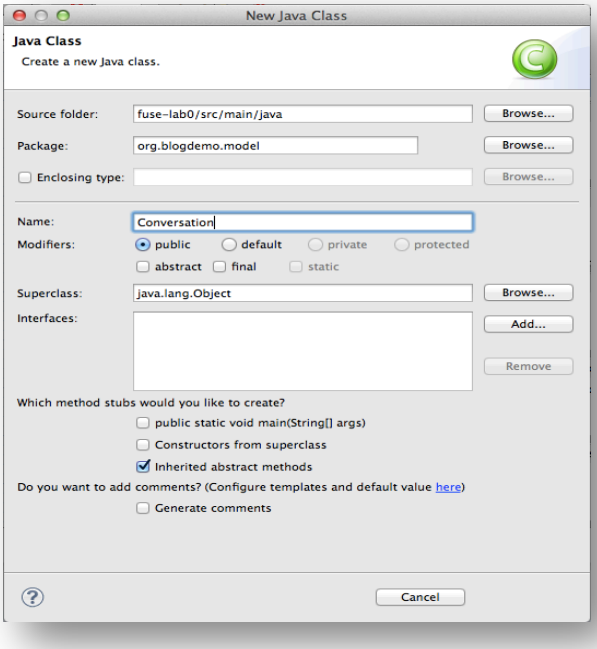
2. Look at the Camel context, I have already added the JPA related setting for you. But see the "*entityManagerFactory*" is connected to a **cameljpa**, so we need to specify Database setting for JPA. We are going to add a persistence.xml to our project.
   Copy from **`support/persistence.xml`**
   to project's **`src/main/resources/persistence.xml`**



   The persistence.xml contains all the database setting.



3. Before we get started constructing the route, we need to create the bean for the data we are going to send, Conversation Bean.
   Under package : `org.blogdemo.model`
   Class Name: `Conversation`



Paste the following code into Conversation.java

```
package org.blogdemo.model;

import static javax.persistence.LockModeType.PESSIMISTIC_WRITE;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table(schema="fusedemo",name="Conversation")
@NamedQuery(name="getAll",query="SELECT conversationitem from
Conversation conversationitem", lockMode = PESSIMISTIC_WRITE)
public class Conversation implements Serializable{

	private static final long serialVersionUID = 113744899661593117L;

	@Id
	@Column(name = "coninput")
	String coninput;

	@Column(name = "conresponse")
	String  conresponse;

	public String getConinput() {
		return coninput;
	}

	public void setConinput(String coninput) {
		this.coninput = coninput;
```

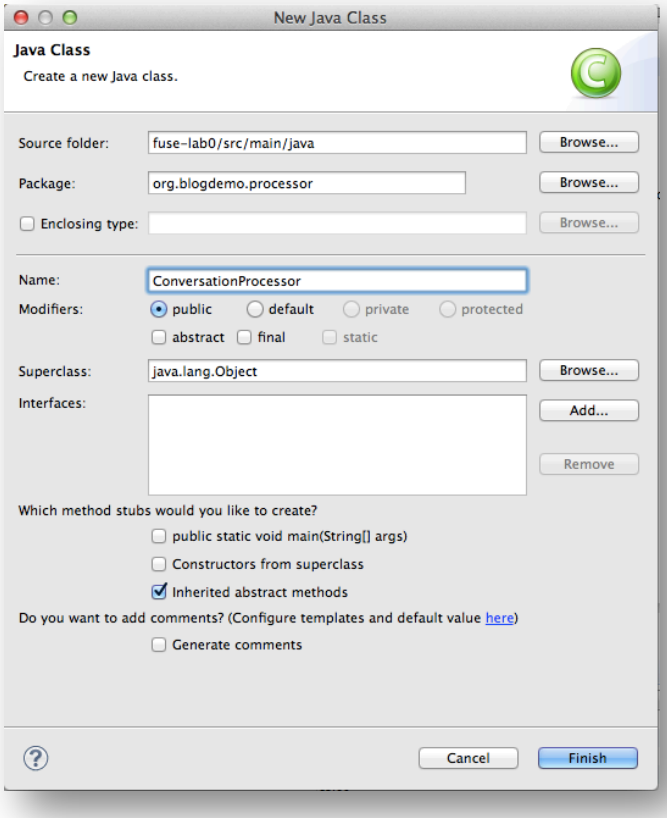
```
        }

        public String getConresponse() {
              return conresponse;
        }

        public void setConresponse(String conresponse) {
              this.conresponse = conresponse;
        }


}
```

4. We are also going to add a bean processor, which in this lab, only print out the content of the bean. Create Processors.
   Under package : `org.blogdemo.processor`
   Class Name: `ConversationProcessor`



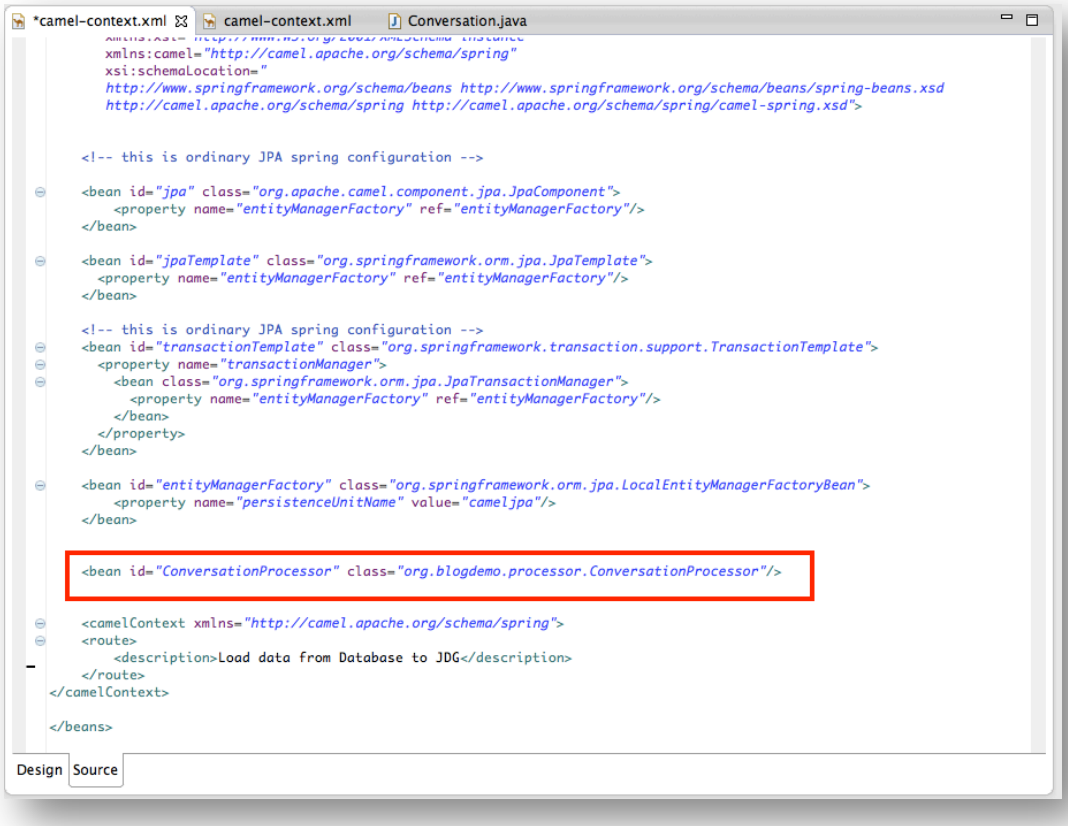
   Paste the following code into ConversationProcessor.java

```
package org.blogdemo.processor;
import org.blogdemo.model.Conversation;

public class ConversationProcessor {
  public Conversation processConversation(Conversation conversation){
    System.out.println("Input:["+conversation.getConinput()+"]
Response:[" +conversation.getConresponse()+"]");
    return conversation;
```

```
    }
}
```

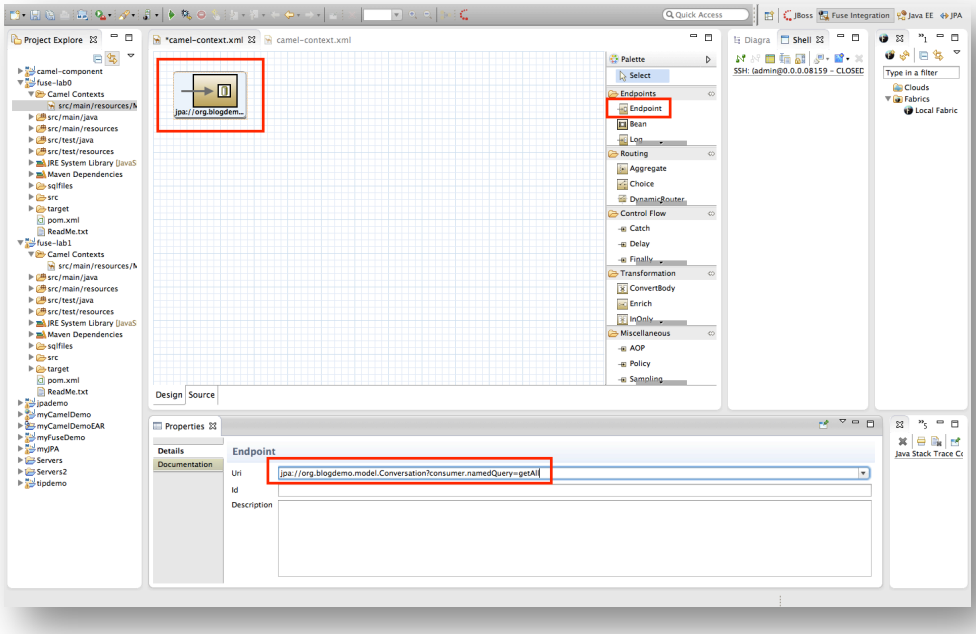
And add bean declaration on the route (camel-context.xml).



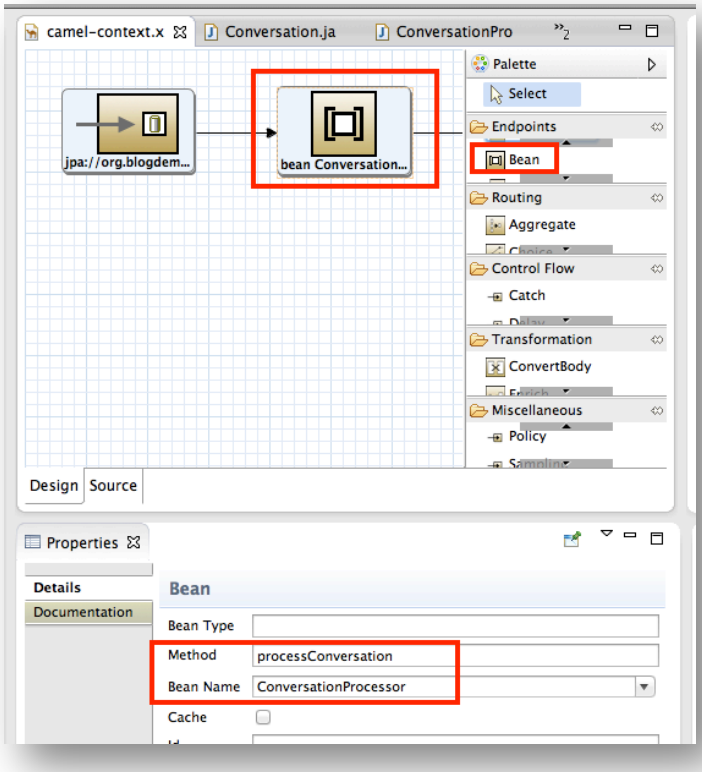
<bean id="ConversationProcessor" class="org.blogdemo.processor.ConversationProcessor"/>

5. Open up and edit Camel Route (camel-context.xml) and choose design tab.
   a. Add JPA endpoint

   Uri: *jpa://org.blogdemo.model.Conversation?consumer.namedQuery=getAll*

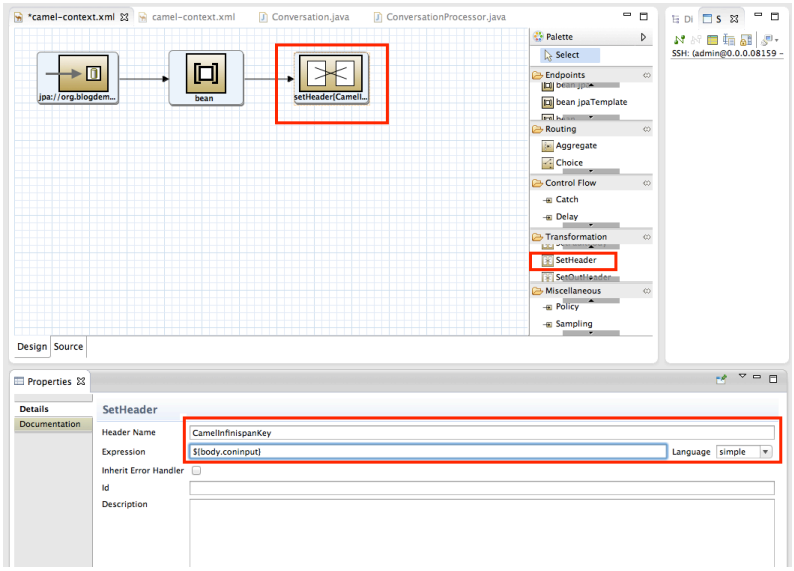


b. Add Processor Bean to print out the content form database before we insert it into JDG Cache.

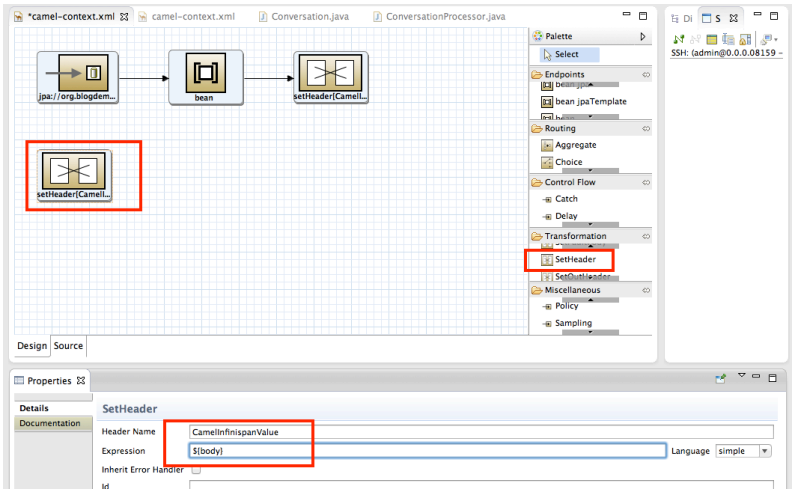


Method: `processConversation`
Bean Name: `ConversationProcessor`

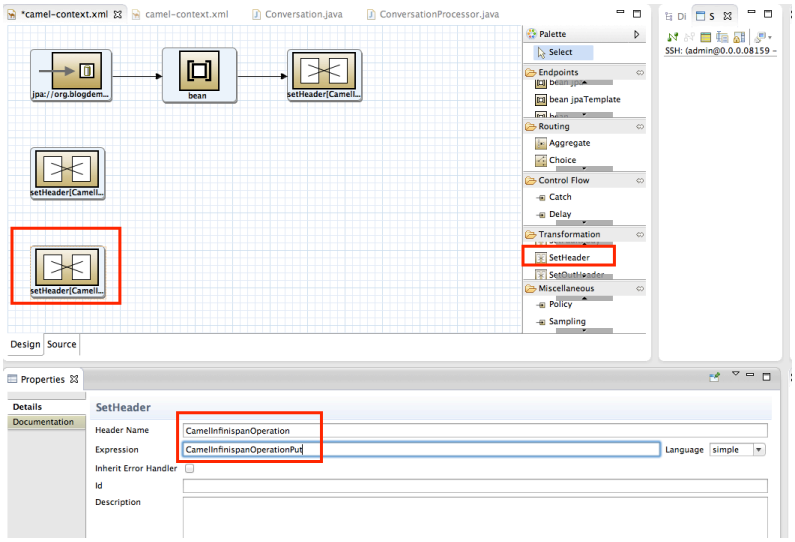
c. All the content needs to be set in the header for it to put inside the cache. That's why we need content transformation adding the right key and value.



HeaderName : `CamelInfinispanKey`
Expression : `${body.coninput}` (Note this is the key in the Conversation Bean)

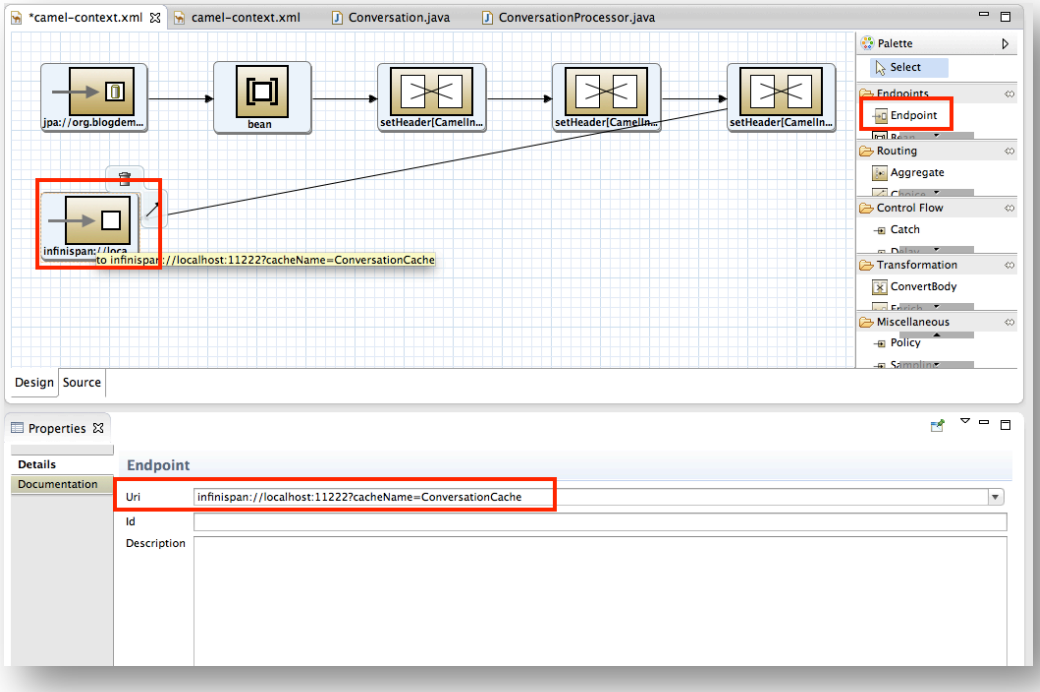


HeaderName : `CamelInfinispanValue`
Expression : `${body }` (Note this is the Conversation Bean)



HeaderName : `CamelInfinispanOperation`
Expression `CamelInfinispanOperationPut`

d. Add InfiniSpan Endpoint
Uri: *`infinispan://localhost:11222?cacheName=ConversationCache`*

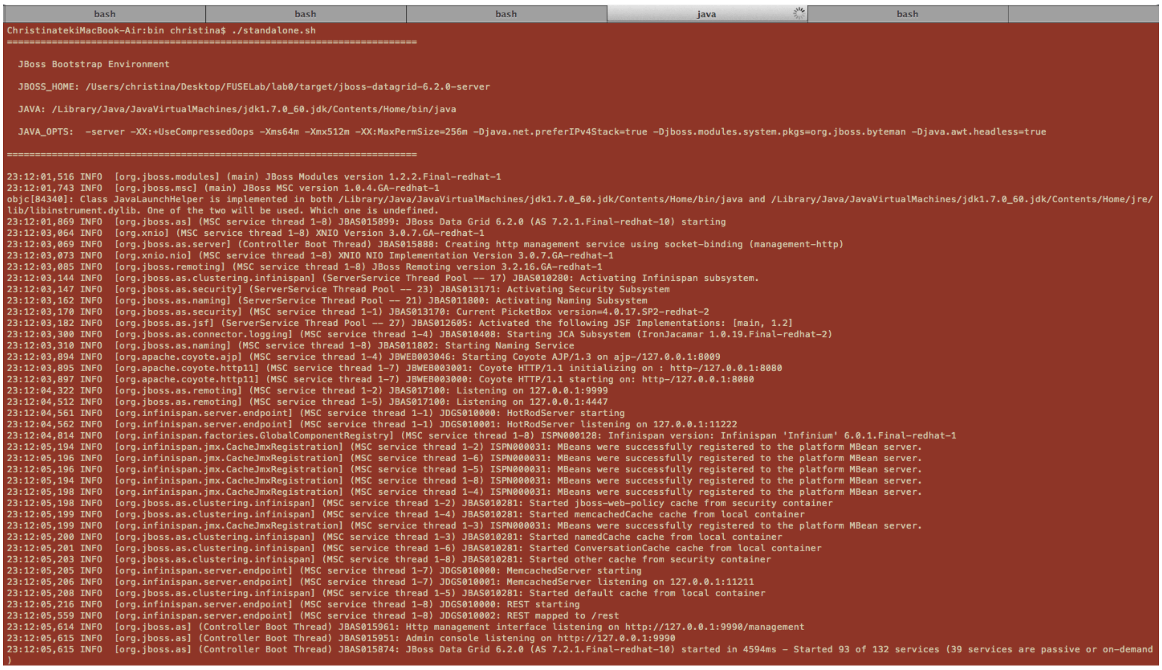


6. Don't forget to link all the components together and save.

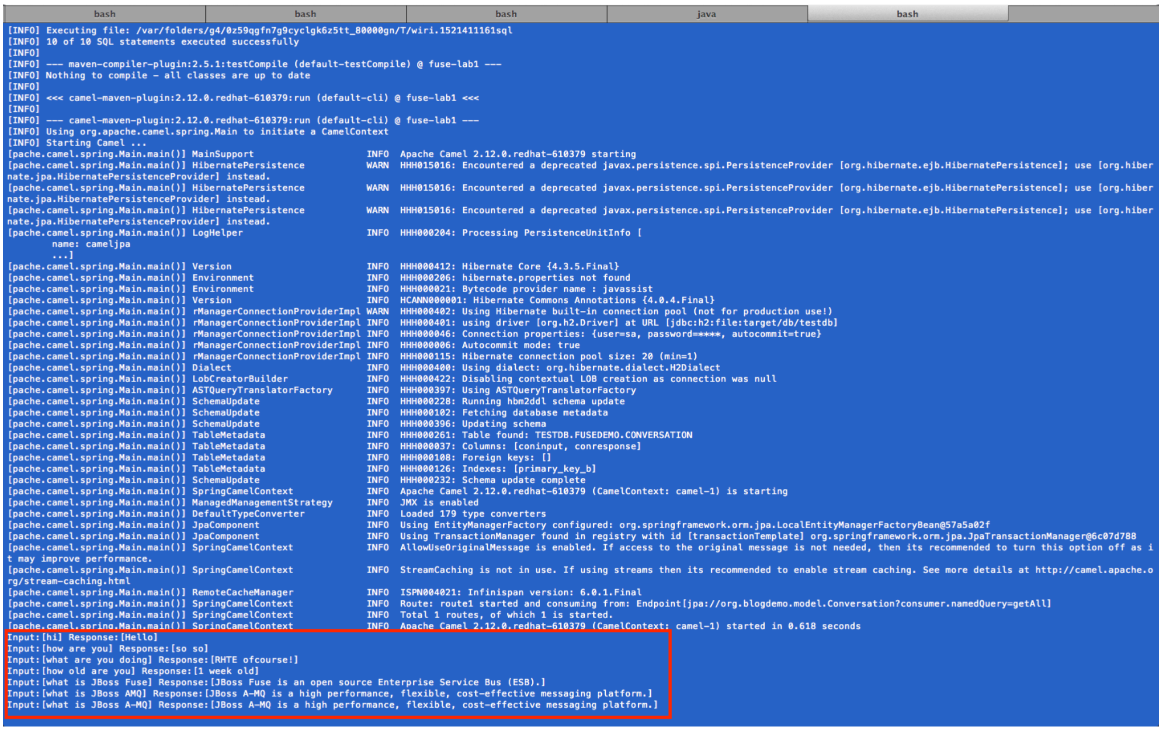
7. Go to command line load data to H2 database. Please run
`mvn process-test-resources`
in the project directory.

8. Compile camel project
`mvn compile`

9. Start up JDG, go to `target/jboss-datagrid-6.2.0-server/bin`
   run *`./standalone.sh`*



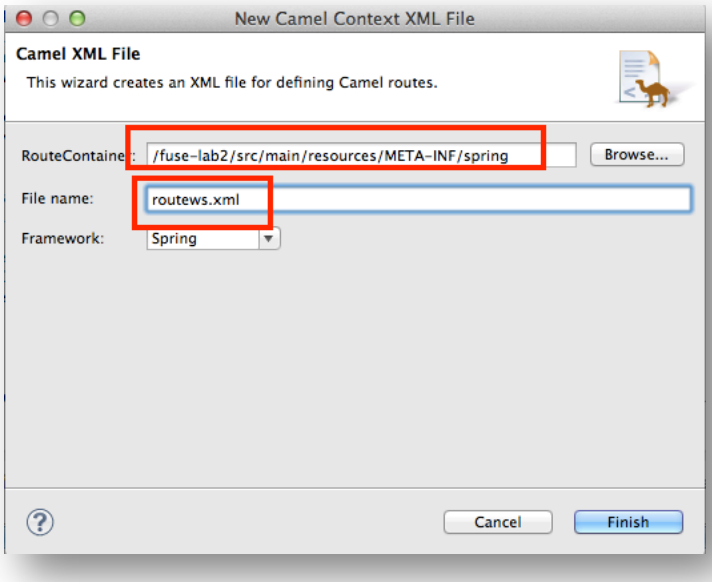
10. Run Camel Route, if you see the data loaded, then congratulation, you have done it!
    `mvn camel:run`



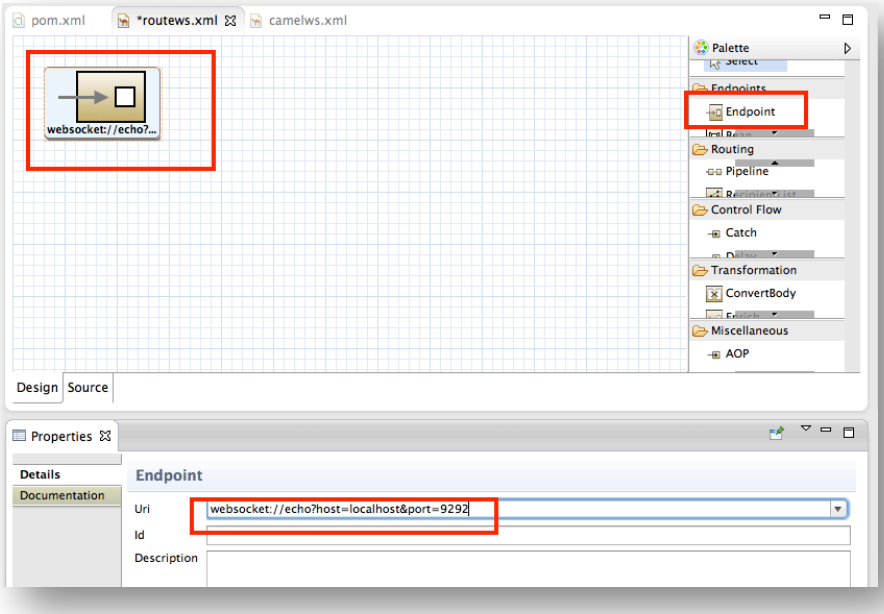
Lab 2

If you did not complete the last lab or having trouble with it. Go to
Download the completed lab to start with lab2.

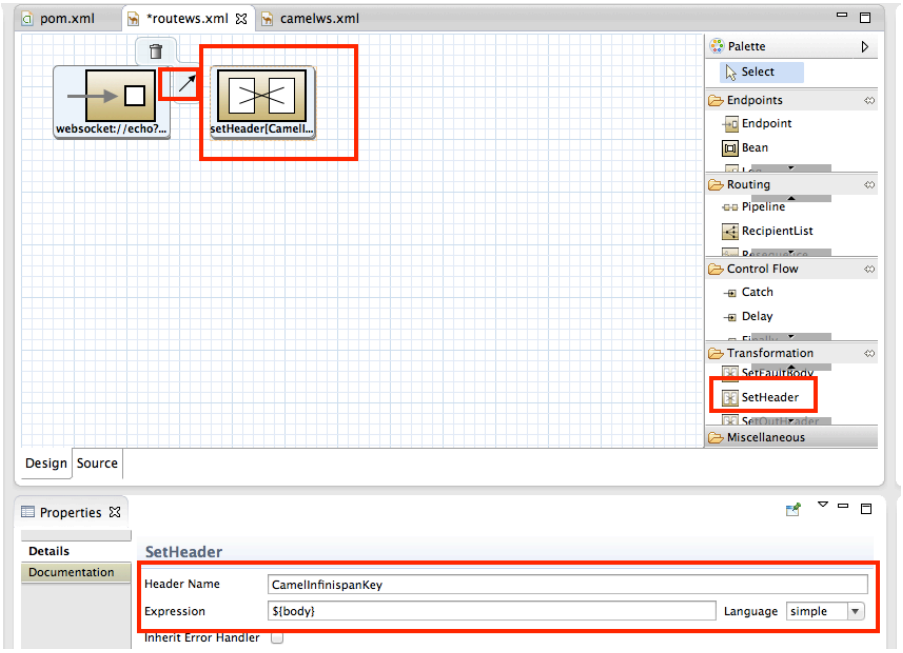
1. Create new route, under the same directory as your other route, name it `routews.xml`



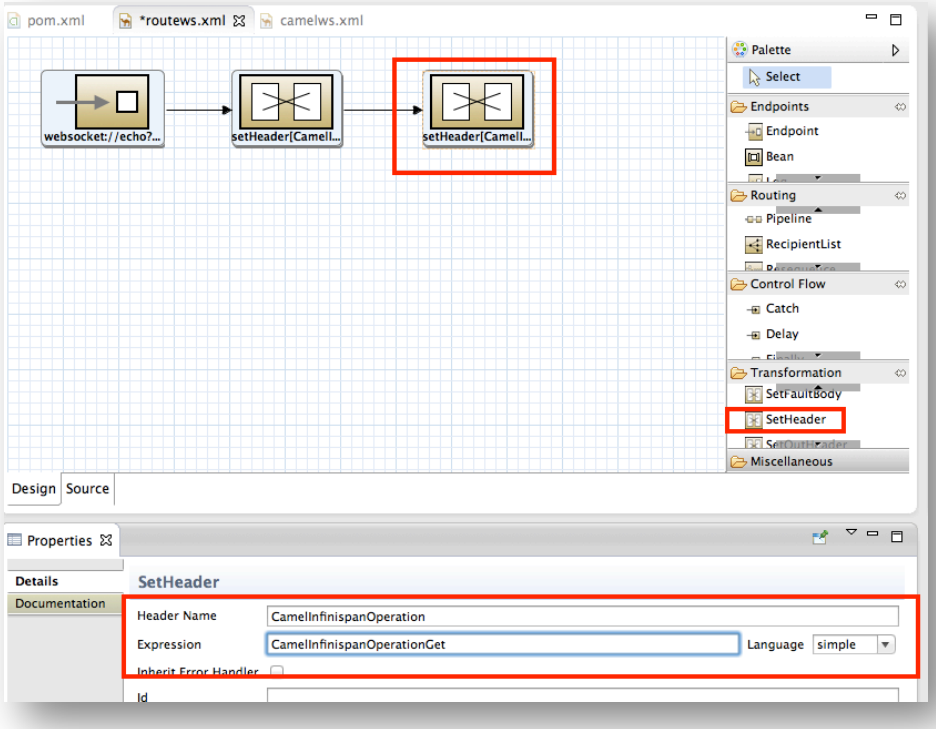
2. Since we are listening through WebSocket, we need to add WebSoket endpoint.
   Drag Endpoint to the left canvas .
   Uri: *websocket://echo?host=localhost&port=9292*



3. We want to get content out from the JDG cache, needs to set the key to look up in the header and also specify the action GET.
4.

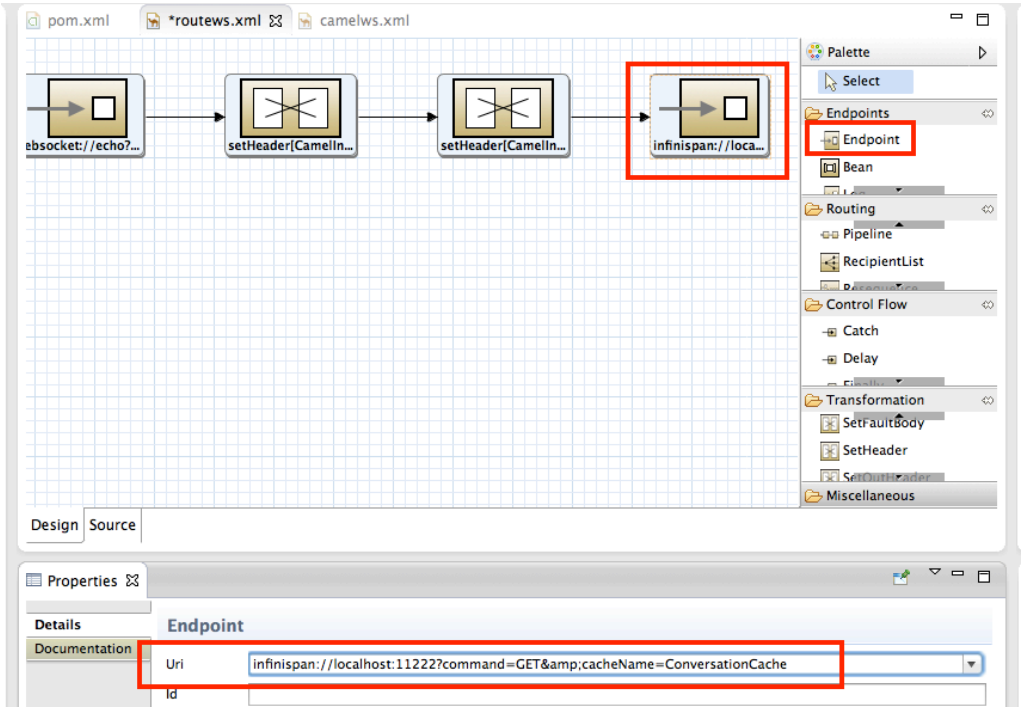


Method: `CamelInfinispanKey`
Bean Name: `${body}`

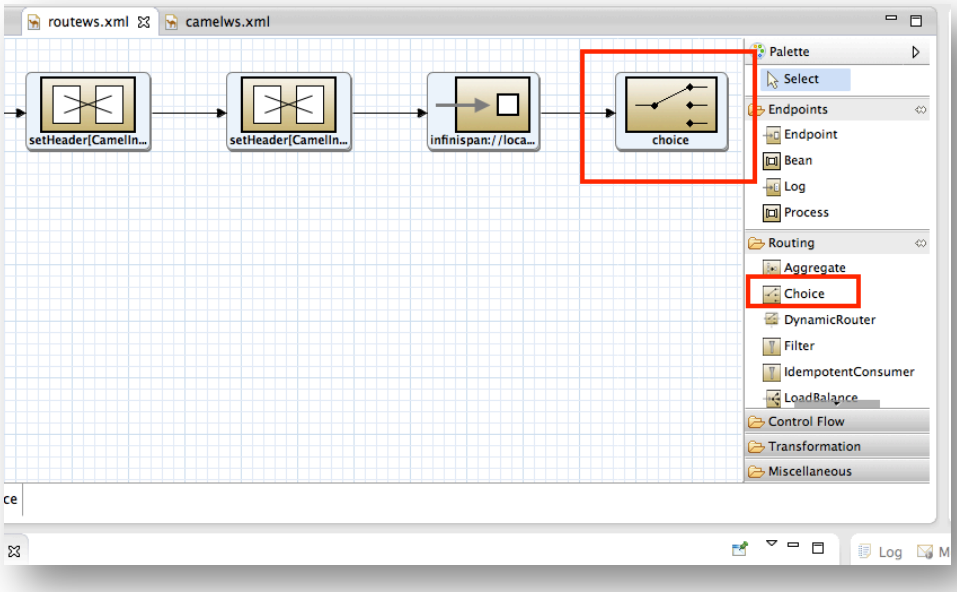


Method: `CamelInfinispanOperation`
Bean Name: `CamelInfinispanOperationGet`

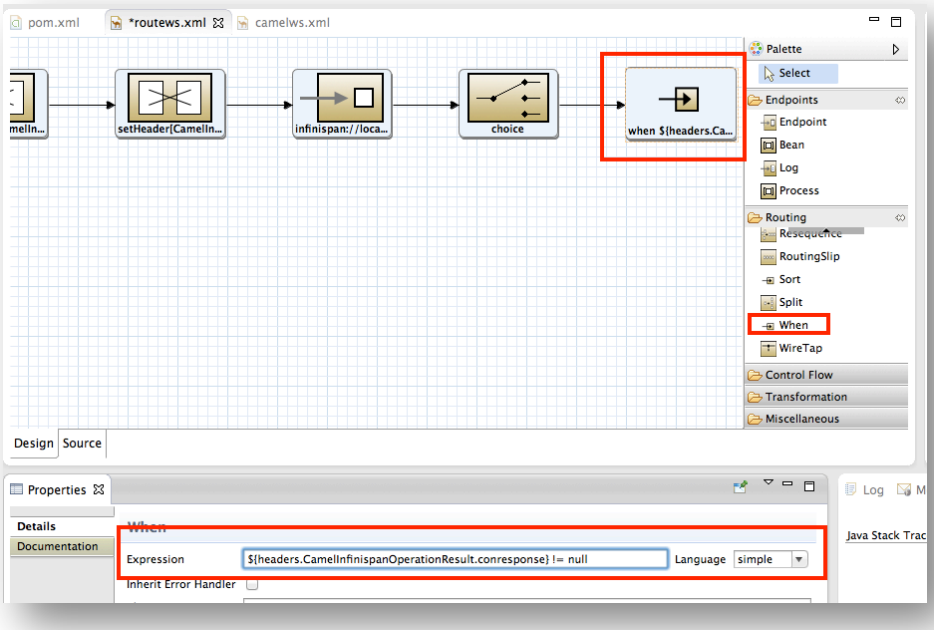
Add the JDG endpoint.
URI: `infinispan://localhost:11222?command=GET&cacheName=ConversationCache`



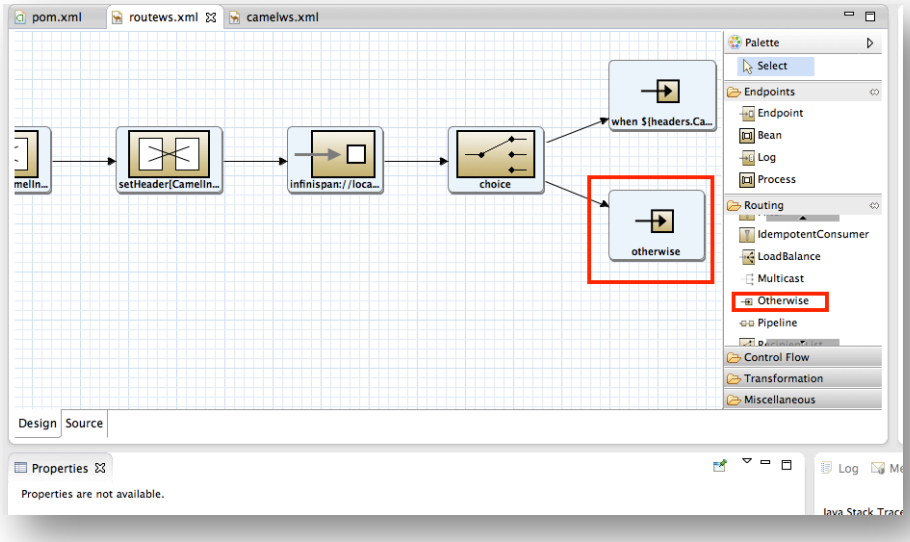
5. Base on the content we retrieve from JDG, determine what we reply back to webpage.
   We are going to use Content Switching, drag Choice to the left corner.



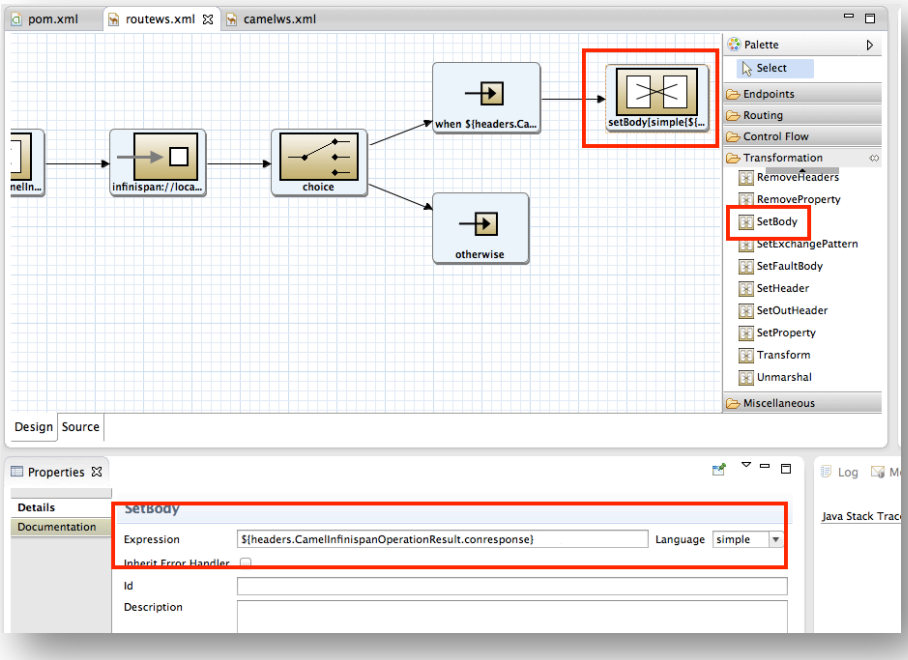
Add when component, we want to make sure it's found in JDG, so we add
`${headers.CamelInfinispanOperationResult.conresponse} != null`



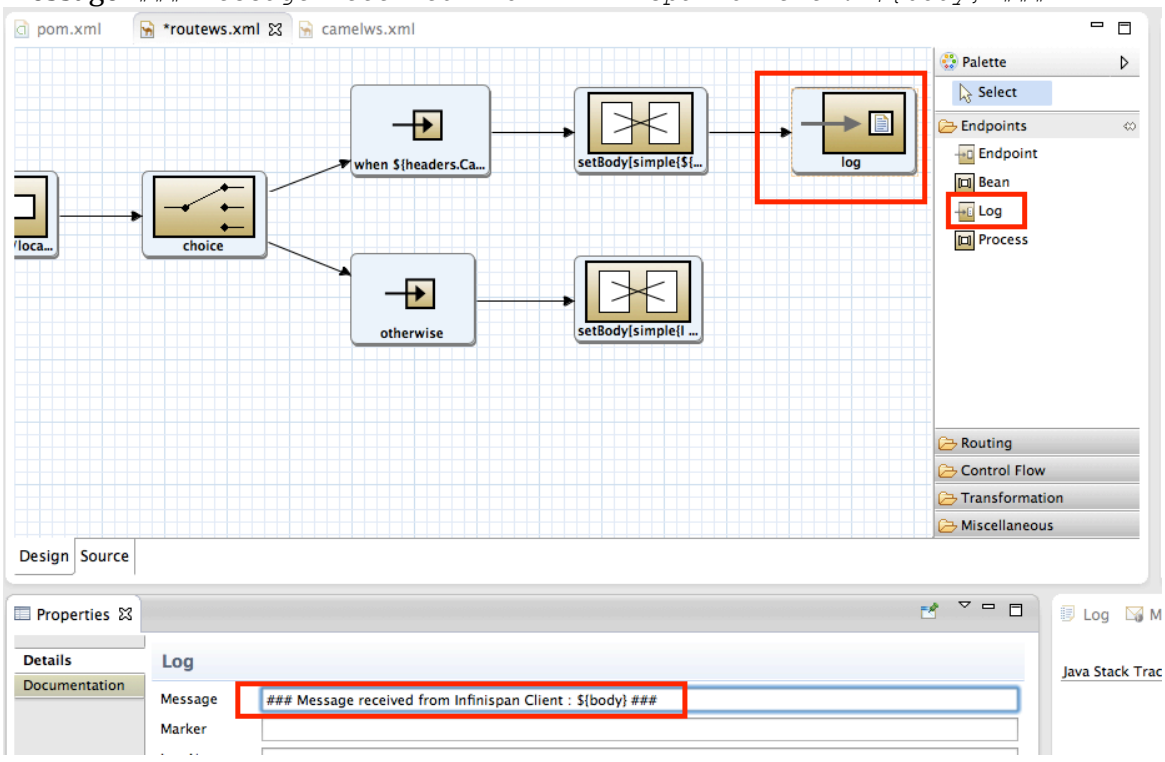
We need also to have a response if the content was not found in JDG, so, add Otherwise.



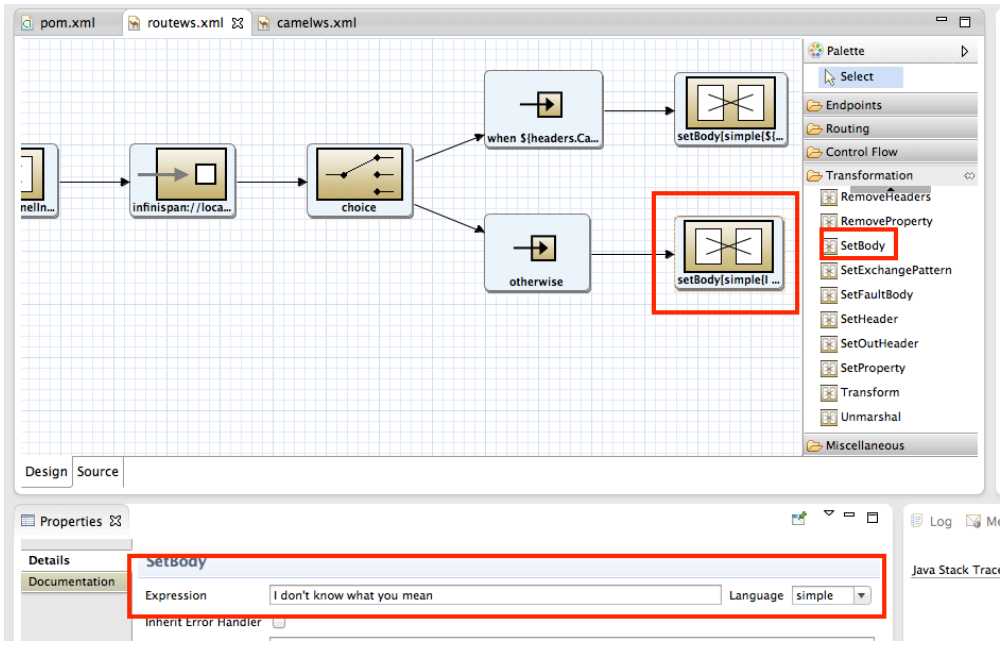
When the content is found we want to reply the content that we found in JDG. So we will set the result to body
Expression:
${headers.CamelInfinispanOperationResult.conresponse}



Add log,
Message: `### Message received from Infinispan Client : ${body} ###`

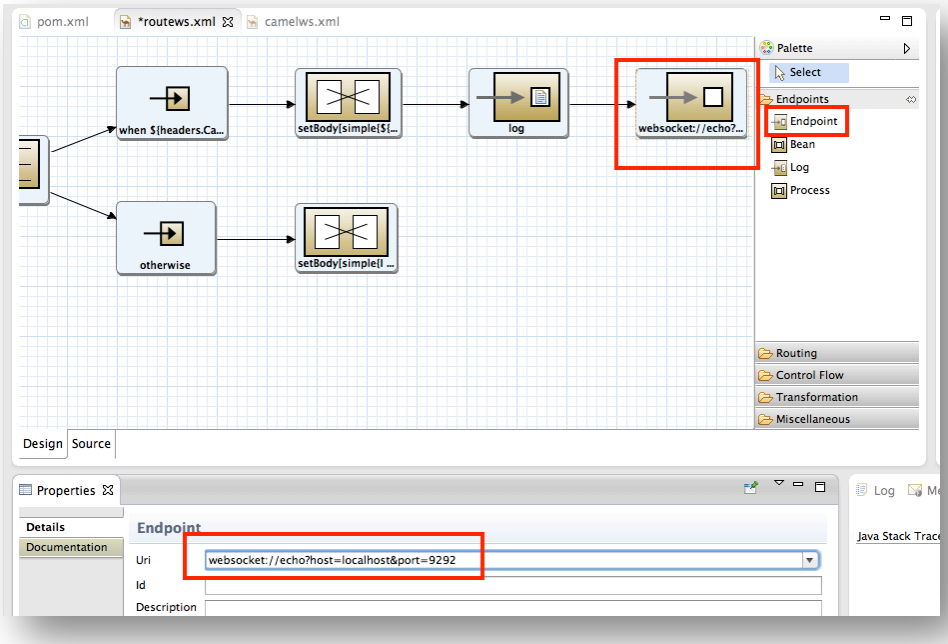


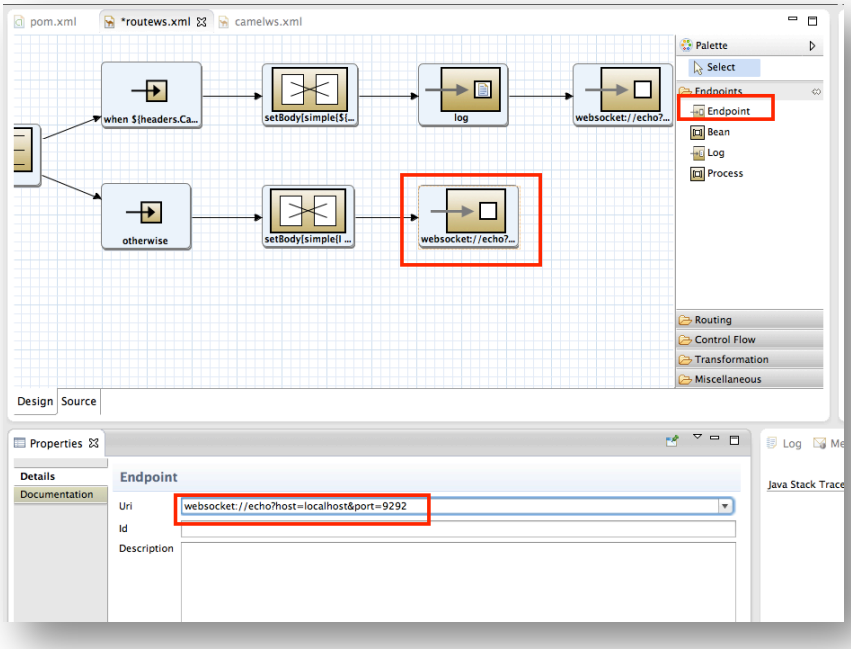
And if there are nothing found in JDG, we need a notify message:
Expression: `I don't know what you mean`



6. Lastly we need these result to be send back to the webpage, Send data back to WebSoket endpoint
   URI: `websocket://echo?host=localhost&port=9292`

   Connect to both route.



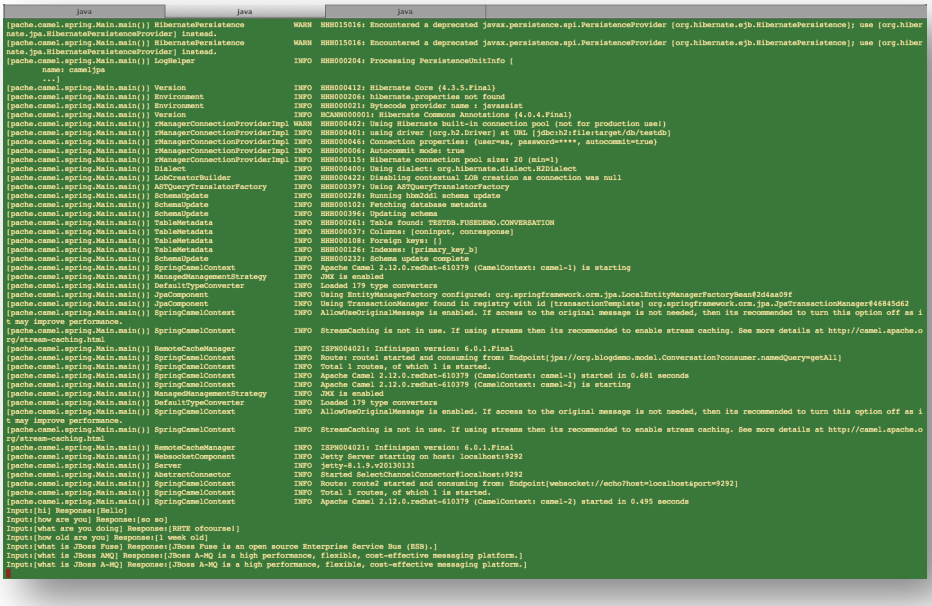


That's it for development.

We can now start running the example:

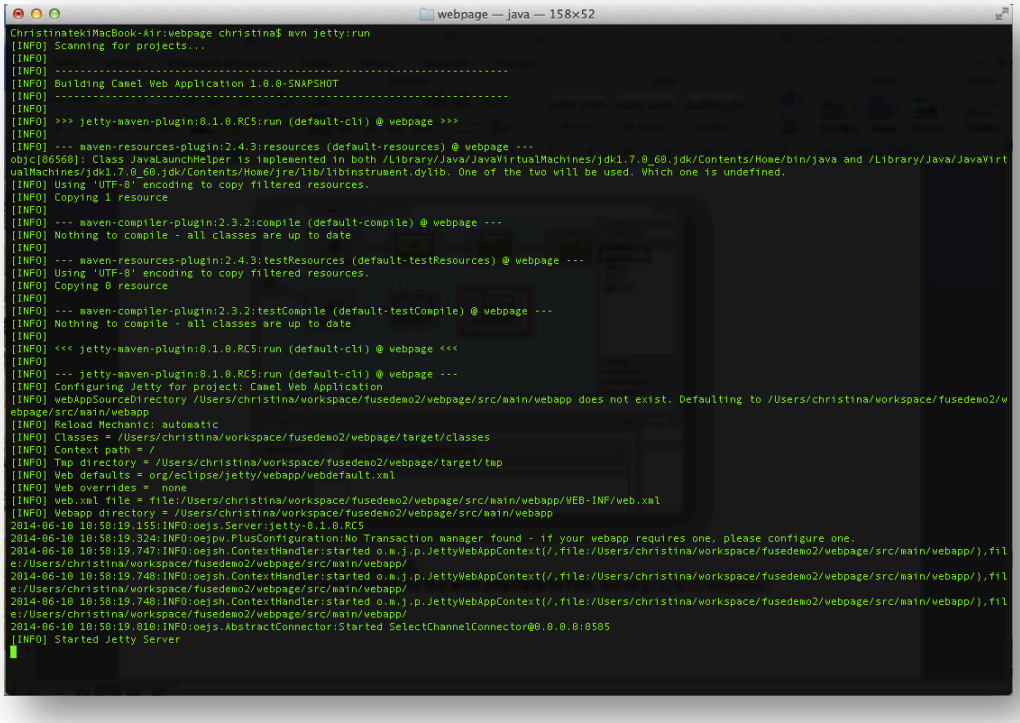
Frist make sure your JDG is still running.
Start up the camel route by running under /project/labx/directory

```
mvn camel:run
```

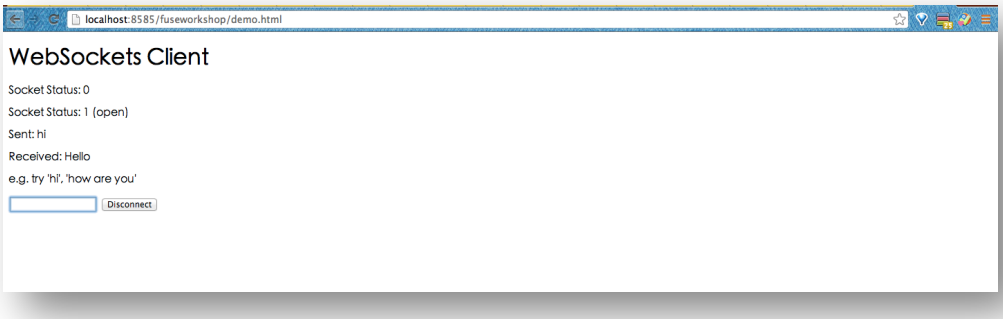


Startup the web server for the web interface to run. Go to `project/webpage/` directory.

mvn jetty:run



Go to http://localhost:8585/fuseworkshop/demo.html
You will see the console, try “hi”, you should see the built in response.



Type yallow , you should see the not found response.