JBoss Developer Studio 4.0
Hands on Lab

SOA-P

Table of Contents

Introduction

Overview

JBoss SOA Platform is a collection of technologies designed to meet an organization's SOA needs. SOA-P includes an ESB, BPM engine (jBPM), Rules engine (JBoss Rules), UDDI Registry (jUDDI), as well as a full JEE application server. To cover each of these areas in depth is beyond the scope of this workshop. Instead, this workshop is designed to give you an overview of the SOA Platform as well as some experience using JBoss Developer Studio to create and deploy SOA-P applications.

SOA-P ESB Architecture

Understanding the SOA-P ESB architecture is important to really understanding what is happening in the following labs. Here is an architecture overview of the SOA-P ESB that we will discuss:

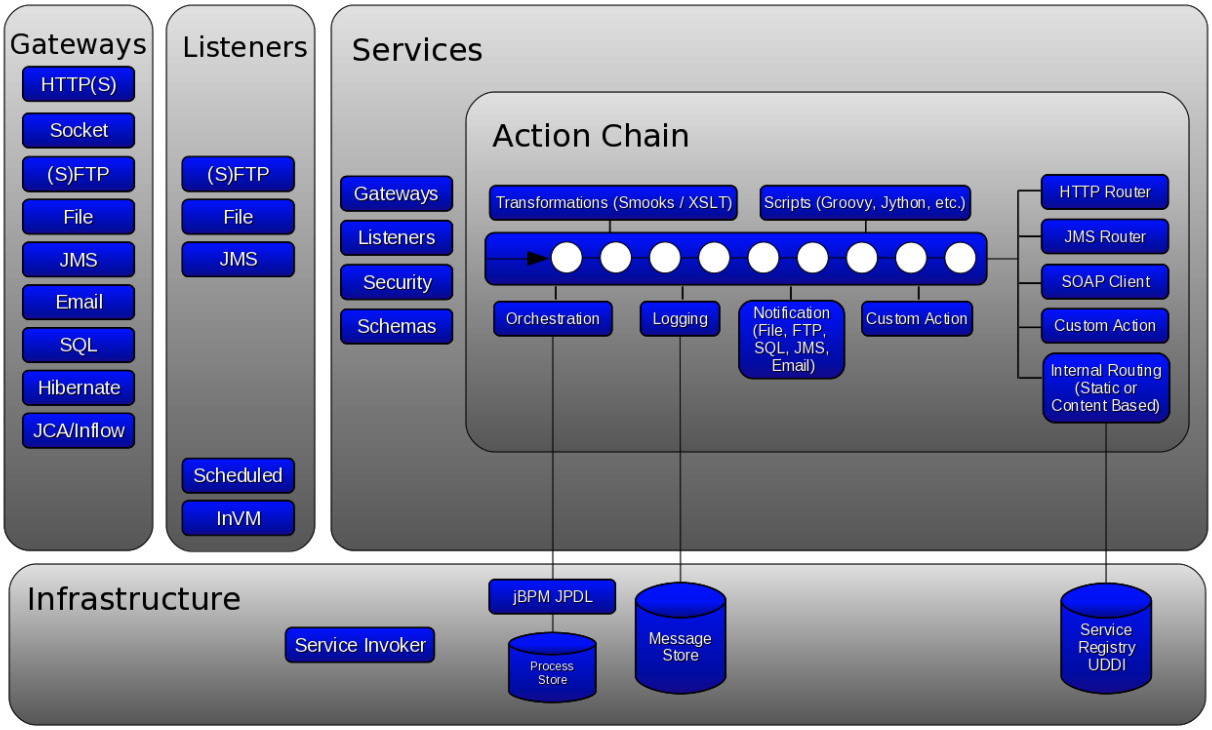


Illustration 1: ESB Architecture

The typical scenario for ESB messages is that a message comes in from the left through a “Gateway”. A Gateway can be any of the protocols listed in the gateways box above or a custom gateway implementation. Any type of message can come into a gateway: text, binary, image, XML, SOAP, etc. The gateway's job is to get the message, wrap it in a JBoss ESB internal message structure and pass it to a “Listener”. The listeners in this context are expecting a

JBoss ESB Message structure. They will take the message they are given and hand it to a service. Depending on the service, this may include security authorization or schema validation. Assuming both of those are okay, the message will travel to the action chain for the service. The action chain is simply a chain of Java actions that can perform processing on the message. The type of processing that can be done on the message is limited only by Java. Some Java actions may do transformations, some may execute scripts, some may log, some may call out to external systems like web services, etc. In the end, the last action in the chain is invoked. This is definitely an over simplification of the process, but will hopefully give some context to the terminology that will be used throughout the workshop.

Included FIles

Several files are included with this workshop. There is a copy of SOA-P 5.1 (platform agnostic). There are copies of JBoss Developer Studio for Windows, Mac, and Linux.

System Expectations

It is expected that you have a Windows, Linux or Mac notebook and you are comfortable working and running Java programs on it. It is expected you will have the environment PATH set to include a JDK 6.0 to use for these labs. It is also a good idea to have JAVA_HOME set to your JDK that you plan on using. Please make sure you do this before running any of the labs. Two examples of what these settings might look like is below:

```
PATH=${Some Path}/jdk1.6.0_17/bin:${Some Path}/ant/apache-ant-1.7.1: ${More Path Info}
```

```
JAVA_HOME=${Some Path}jdk1.6.0_17
```

To verify that this is correct you will have to look at these values on your system One simple way to check the JDK version that you have is to run:

```
java -version
```

to see which one is in your path, and it should be a JDK 6 version to run this lab. Also note that Ant has been installed for you and you will have to install a few other things as the lab progresses.

Please note that having an existing CLASSPATH environment variable set may cause odd issues with jar class loading, it is recommended to have this empty and not set. Please make sure to back up this value for when the lab is over. You are welcome to not do this, however weird things may happen when you are running through the labs.

What is Expected of You

Please feel free to raise your hands with any questions that you have about the lab; feel free to ask why it is you are doing something, or if something does not feel right. Please know that all care was made in creating this user guide, but all screen shots and steps along the way might be off by just a little so please be patient with any issues.

Lab Number 1: Install and Configure SOA Platform

Get the File

In the ${USER_HOME}Downloads/Platforms directory you will find the SOA-P installer, it platform agnostic and it should look something like this:

```
soa-5.1.0.GA.zip
```

Just Unzip and Go

Installing the SOA-P is very very simple, and has the following high level steps:

Create a Servers directory in the user home directory and make this unique based on your initials.

Unzip the contents of the file above into that directory.

mkdir ${USER_HOME}/ServerXXX

```
cd ${USER_HOME}/ServerXXX
```

```
unzip ${USER_HOME}/Downloads/Platforms/soa-5.1.0.GA.zip
```

Your command should look something like this:

Illustration 2: Installing SOA-P

By default, SOA-P is shipped without an administration user configured. We want to enable that default user for some future labs by editing:

${USER_HOME}/ServerXXX/jboss-soa-p-5/jboss-as/server/default/conf/props/soa-users.properties

and removing the "#" to uncomment the admin user. As shown below:

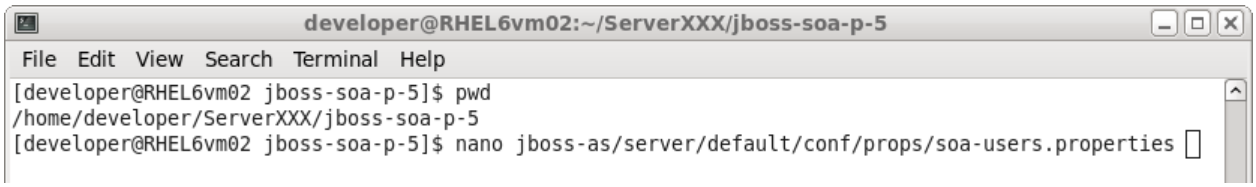


Illustration 3: Edit User Configuration

Illustration 4: Enable admin User

You have now completed the first lab.

Lab Number 2: SOA Platforms Quickstarts

Explore the SOA Quickstarts

Now that you have SOA-P installed, where do you go from here? There are so many things people want to do with a SOA Platform – mediation, web services, JMS, transformations, orchestration, security, routing, FTP, and a host of other things. Wouldn't it be nice if there were a single place that had a working example of pretty much every piece of functionality that could be done? SOA-P has that in the form of "quickstarts". A quickstart is a complete working project (with complete source) that can be built with a single ant task, deployed with a single ant task, and run with a single ant task. You can run the examples to see them work and then look at the code (or even modify it) to see what is really happening under the covers.

So, let's go look at the quickstarts with SOA-P with the following commands:

```
cd ${USER_HOME}/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts
ls
```

Below you can see the available quickstarts, a pretty large list:

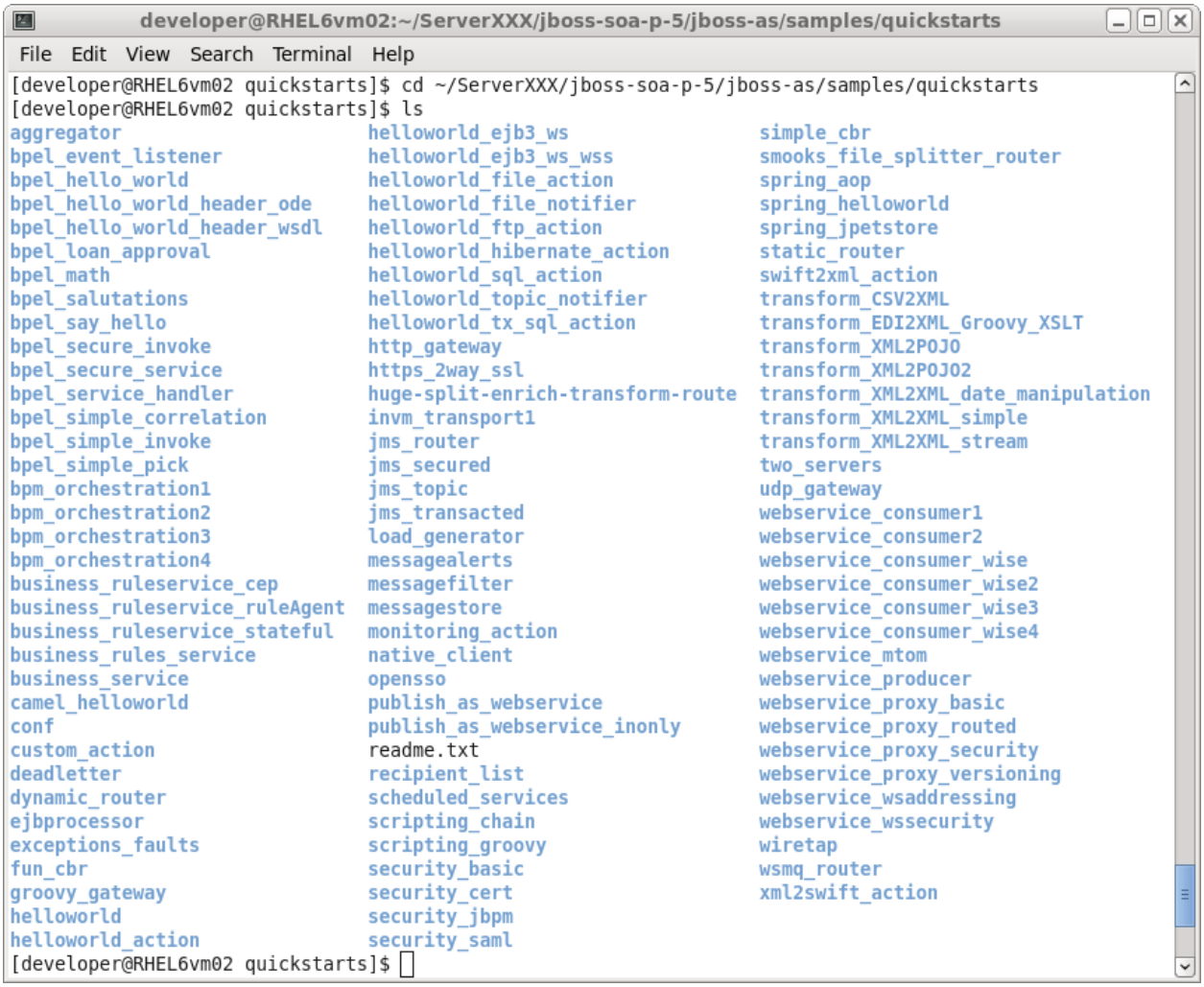


Illustration 5: List of quickstarts

Start the Server

In order to run the quickstarts you need to start the soa-p server by doing the following:

```
cd ${USER_HOME}/ServerXXX/jboss-soa-p-5/jboss-as/bin
./run.sh -c default
```

As seen below:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/bin
File  Edit  View  Search  Terminal  Help
/home/developer/ServerXXX/jboss-soa-p-5/jboss-as/bin
[developer@RHEL6vm02 bin]$ ./run.sh -c default
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /home/developer/ServerXXX/jboss-soa-p-5/jboss-as

  JAVA: java

  JAVA_OPTS: -Dprogram.name=run.sh -server -Xms1303m -Xmx1303m -XX:MaxPermSize=256m -Djava.awt.headless=
true -Dorg.apache.xml.dtm.DTMManager=org.apache.xml.dtm.ref.DTMManagerDefault  -Dorg.jboss.resolver.warn
ing=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Dsun.lang.Clas
sLoader.allowArraySyntax=true -Djava.net.preferIPv4Stack=true

  CLASSPATH: /home/developer/ServerXXX/jboss-soa-p-5/jboss-as/bin/run.jar

=========================================================================

21:34:19,258 INFO  [ServerImpl] Starting JBoss (Microcontainer)...
21:34:19,258 INFO  [ServerImpl] Release ID: JBoss [SOA] 5.1.0.GA_SOA (build: SVNTag=5.1.0.GA_SOA date=20
1102110032)
21:34:19,259 INFO  [ServerImpl] Bootstrap URL: null
21:34:19,259 INFO  [ServerImpl] Home Dir: /home/developer/ServerXXX/jboss-soa-p-5/jboss-as
21:34:19,259 INFO  [ServerImpl] Home URL: file:/home/developer/ServerXXX/jboss-soa-p-5/jboss-as/
21:34:19,259 INFO  [ServerImpl] Library URL: file:/home/developer/ServerXXX/jboss-soa-p-5/jboss-as/lib/
21:34:19,259 INFO  [ServerImpl] Patch URL: null
21:34:19,260 INFO  [ServerImpl] Common Base URL: file:/home/developer/ServerXXX/jboss-soa-p-5/jboss-as/c
```

Illustration 6: SOA-P Starting...

When it is finished starting you will see the message (highlighted for emphasis) below:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/bin
File  Edit  View  Search  Terminal  Help
21:35:14,150 INFO  [HbmBinder] Mapping collection: org.jbpm.taskmgmt.exe.TaskInstance.pooledActors -> JB
PM_TASKACTORPOOL
21:35:14,150 INFO  [Configuration] Reading mappings from resource : org/jbpm/taskmgmt/exe/PooledActor.hb
m.xml
21:35:14,161 INFO  [HbmBinder] Mapping class: org.jbpm.taskmgmt.exe.PooledActor -> JBPM_POOLEDACTOR
21:35:14,166 INFO  [ServerImpl] JBoss (Microcontainer) [5.1.0.GA_SOA (build: SVNTag=5.1.0.GA_SOA date=20
1102110032)] Started in 54s:904ms
21:35:14,167 INFO  [HbmBinder] Mapping collection: org.jbpm.taskmgmt.exe.PooledActor.taskInstances -> JB
PM_TASKACTORPOOL
21:35:14,173 INFO  [Configuration] Reading mappings from resource : org/jbpm/taskmgmt/exe/SwimlaneInstan
ce.hbm.xml
```

Illustration 7: SOA-P Started

Configure the Quickstarts

In order to run the quickstarts you need to move a file, and then make a change in it.

```
cd ${USER_HOME}/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts/conf

mv quickstarts.properties-example quickstarts.properties
```

As shown below:

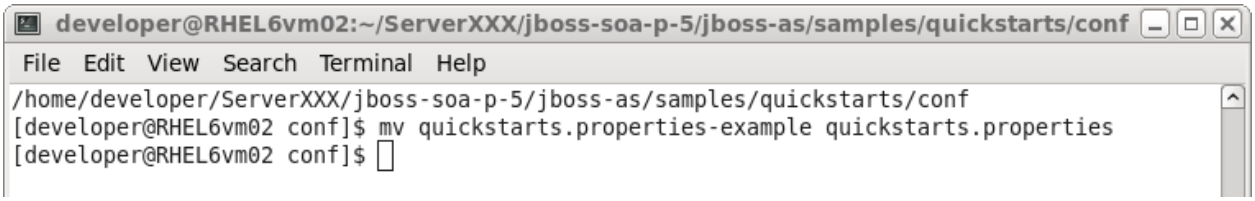


Illustration 8: Move quickstart.properties

The renamed file above now needs to have four things added to it:

1. Uncomment the org.jboss.esb.server.home line and update it with the full path including jboss-as as shown in the below screen shot.

```
org.jboss.esb.server.home=${USER_HOME}/ServerXXX/jboss-soa-p-5/jboss-as
```

2. Uncomment the default configuration:

```
org.jboss.esb.server.config=default
```

3. Uncomment the first jBPM setting:

```
jbpm.console.username=admin
```

4. Uncomment the second jBPM setting:

```
jbpm.console.password=admin
```

As seen in the below screen shot:

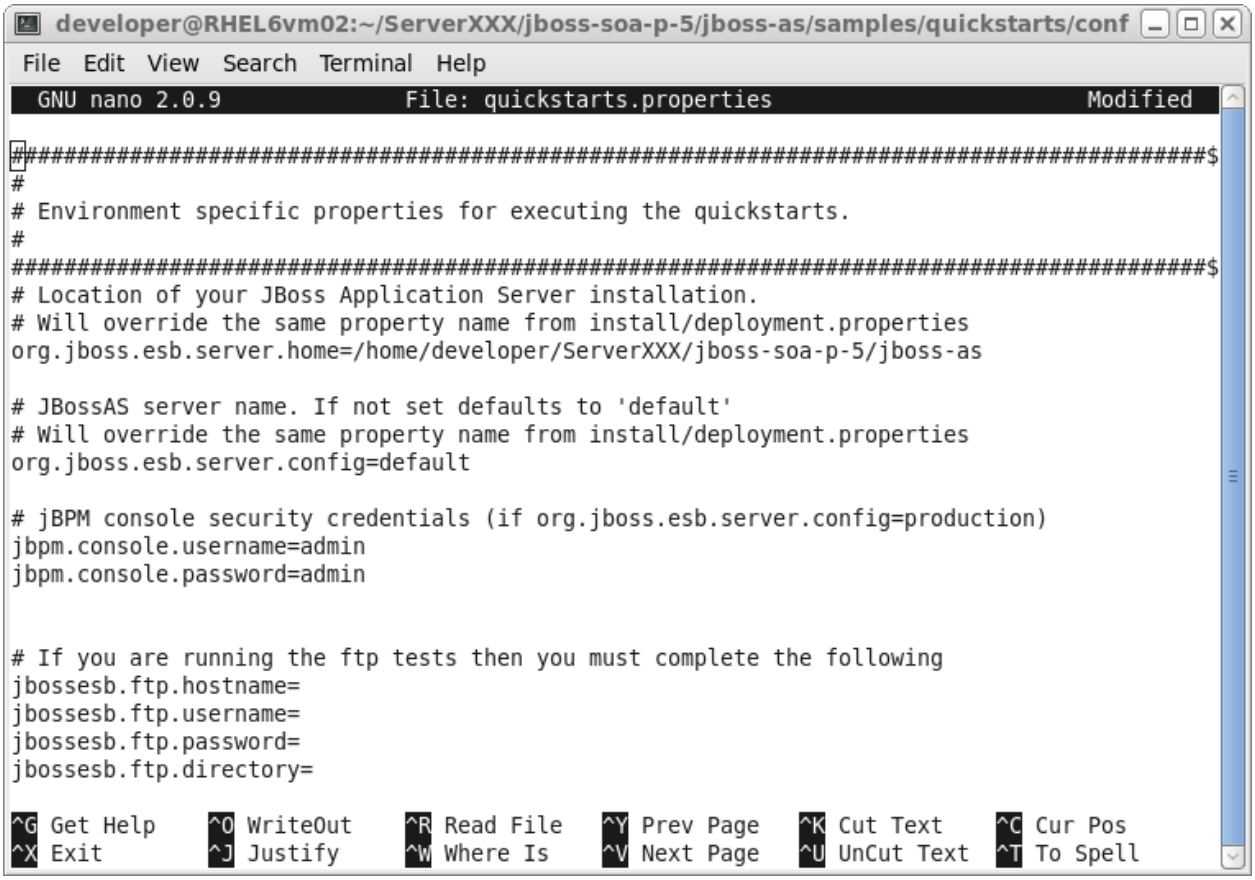


Illustration 9: Updated quickstart.properties

Deploy a Quick Start

Now that you have the runtime environments setup correctly. Change to the helloworld directory and run your first quickstart

```
cd ../helloworld
```

```
ant deploy
```

As shown in the below screen shot:

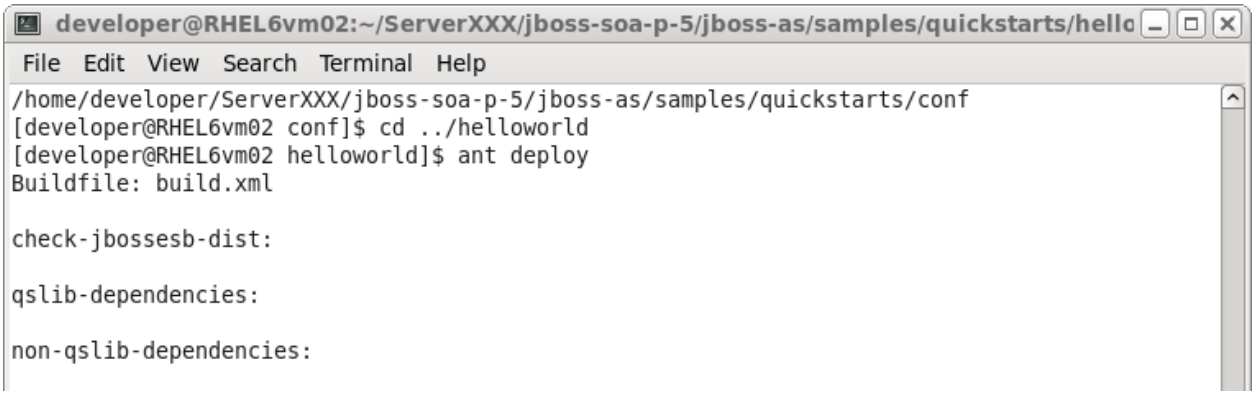


Illustration 10: Start helloworld deployment

When it is done building you should see a BUILD SUCCESSFUL message as below:

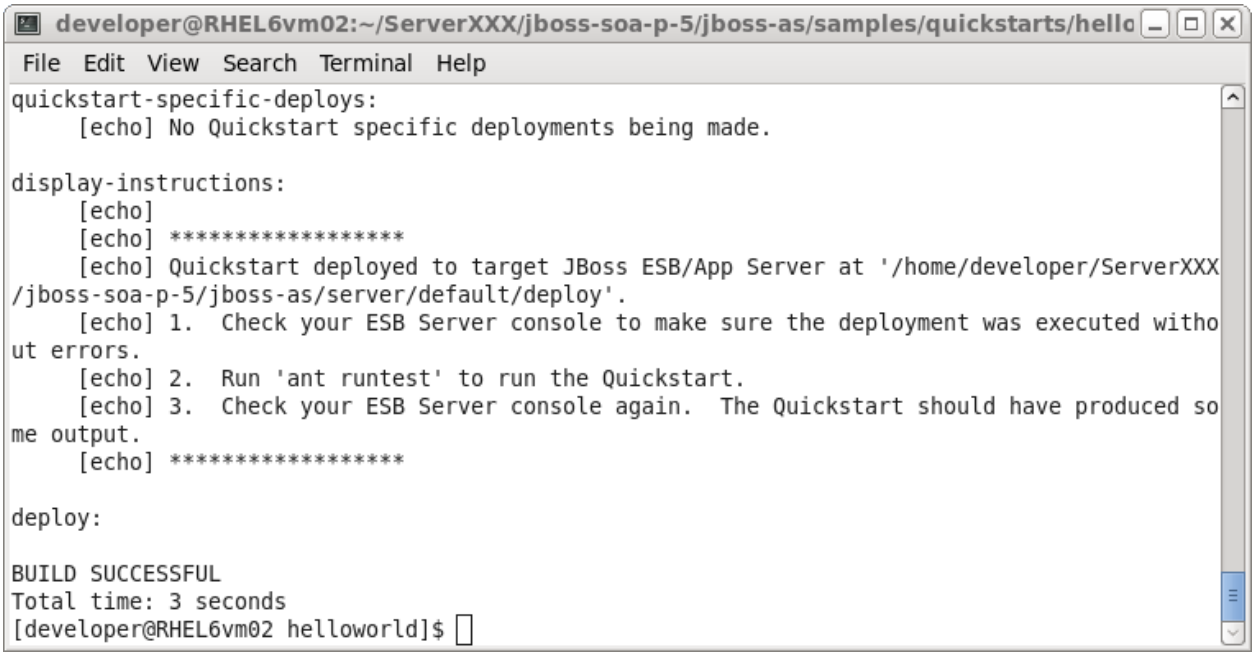


Illustration 11: helloworld Deployed

In the console where you started SOA-P, you should see a message about “Starting ESB Deployment....”, as below:

developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/bin

File Edit View Search Terminal Help

```
22:17:35,639 INFO  [SettingsFactory] Optimize cache for minimal puts: disabled
22:17:35,639 INFO  [SettingsFactory] Structured second-level cache entries: disabled
22:17:35,639 INFO  [SettingsFactory] Statistics: disabled
22:17:35,639 INFO  [SettingsFactory] Deleted entity synthetic identifier rollback: disabled
22:17:35,639 INFO  [SettingsFactory] Default entity-mode: pojo
22:17:35,639 INFO  [SettingsFactory] Named query checking : enabled
22:17:35,643 INFO  [SessionFactoryImpl] building session factory
22:17:36,256 INFO  [SessionFactoryObjectFactory] Not binding factory to JNDI, no JNDI name configured
22:17:36,268 INFO  [NamingHelper] JNDI InitialContext properties:{}
22:17:45,302 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_esb] started, fullSize=2000
00, pageSize=2000, downCacheSize=2000
22:17:45,317 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_gw] started, fullSize=20000
0, pageSize=2000, downCacheSize=2000
22:17:45,386 INFO  [EsbDeployment] Starting ESB Deployment 'Quickstart_helloworld.esb'
```

Illustration 12: Log shows helloworld deployment

Run a Quick Start

Now run:

```
ant runtest
```

You should see the following output in the helloworld directory:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts/hello
File  Edit  View  Search  Terminal  Help

quickstart-specific-dependencies:

classpath-dependencies-as4:

classpath-dependencies-as5:

quickstart-specific-checks:

dependencies:

compile:

runtest:
     [echo] Runs Test JMS Sender
     [java] Connection Started

BUILD SUCCESSFUL
Total time: 3 seconds
[developer@RHEL6vm02 helloworld]$
```

Illustration 13: Run test with helloworld

In the console where you are running the soa-p instance you should see the following:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/bin
File  Edit  View  Search  Terminal  Help
22:17:36,256 INFO  [SessionFactoryObjectFactory] Not binding factory to JNDI, no JNDI name configured
22:17:36,268 INFO  [NamingHelper] JNDI InitialContext properties:{}
22:17:45,302 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_esb] started, fullSize=2000
00, pageSize=2000, downCacheSize=2000
22:17:45,317 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_gw] started, fullSize=20000
0, pageSize=2000, downCacheSize=2000
22:17:45,386 INFO  [EsbDeployment] Starting ESB Deployment 'Quickstart_helloworld.esb'
22:20:17,761 INFO  [STDOUT] &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
22:20:17,762 INFO  [STDOUT] Body: Hello World
22:20:17,762 INFO  [STDOUT] &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
22:20:17,763 INFO  [STDOUT] Message structure:
22:20:17,763 INFO  [STDOUT] [Hello World].
```

Illustration 14: helloworld output

Exploring What Happened

So we ran a few commands and what was the net net of all of those things? A few things: first what did the helloworld quickstart do for us? By running ant deploy we packaged up an .esb archive, essentially a zip file not unlike a war or

ear file, and copied it to the running JBoss SOA-P instance. We'll look at that in a second. By running ant runtest we started up a client that would onramp/put a message on to the JMS queue that was running on the JBoss SOA-P instance outputting the [STDOUT}... message you see in the above screen shot.

What do the contents of the .esb archive file created look like, just run this command from the same place as you ran the ant command:

```
jar tf build/Quickstart_hellowrold.esb
```

You should see the following output:

Illustration 15: Contents of helloworld.esb

What makes up the above?

So there are three Java classes. The two in the test directory are actually only client classes used to send the JMS message and not needed in the ESB deployment. The MyJMSListenerAction is a custom action that prints out what we saw in the server console. The jbm-queue-service.xml file defines the JMS queues that get deployed. deployment.xml declares that the ESB services depend on those queues, and jboss-esb.xml defines the services. That's it! Please feel free to look over this example in more detail. You can even change the source code in the quickstart directory and "ant deploy" will recompile and re-deploy.

Exploring Hot Deployment

So with the above new found knowledge lets make a few changes to the above files:

1. Open up the jboss-esb.xml file and repeat the SystemPrintln action2 a few times as shown below. This is towards the bottom of the file. It should look like this when done. Remember that cut and paste is your friend and make sure each action has a unique name.

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts/helloworld
File  Edit  View  Search  Terminal  Help
  GNU nano 2.0.9                File: jboss-esb.xml                        Modified

                />
            </listeners>
            <actions mep="OneWay">
                    <action name="action1"
                         class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"
                         process="displayMessage"
                         />
                     <action name="action2" class="org.jboss.soa.esb.actions.SystemPrintln">
                       <property name="printfull" value="false"/>
                     </action>
                     <action name="action3" class="org.jboss.soa.esb.actions.SystemPrintln">
                       <property name="printfull" value="false"/>
                     </action>
                     <action name="action4" class="org.jboss.soa.esb.actions.SystemPrintln">
                       <property name="printfull" value="false"/>
                     </action>
                     <!-- The next action is for Continuous Integration testing -->
                     <action name="testStore" class="org.jboss.soa.esb.actions.TestMessageStore"/>
            </actions>
        </service>
      </services>

</jbossesb>

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Illustration 16: Repeat the SystemPrintln actions

2. Edit the MyJMSListenerAction.java file adding in your name or something else to the output. The path is src/org/jboss/soa/esb/samples/quickstart/helloworld. You can see the path below:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts/helloworld
File  Edit  View  Search  Terminal  Help
/home/developer/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts/helloworld
[developer@RHEL6vm02 helloworld]$ nano -w src/org/jboss/soa/esb/samples/quickstart/helloworld/MyJMSL
istenerAction.java
```

3. Make a change to the file, something like this:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/samples/quickstarts/helloworld
File  Edit  View  Search  Terminal  Help
  GNU nano 2.0.9 File: .../soa/esb/samples/quickstart/helloworld/MyJMSListenerAction.java


  public Message displayMessage(Message message) throws Exception{

                  System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&");
                  System.out.println("<Your Name> was here");
                  System.out.println("Body: " + message.getBody().get()) ;
                  System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&");
                  return message;

        }


}

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Illustration 17: Edit custom action

4. Run ant deploy to copy the new files out to the server. Wait for the JBoss SOA-P instance to show that the new archive is deployed and then run ant runtest again to see the new output as shown below:

```
developer@RHEL6vm02:~/ServerXXX/jboss-soa-p-5/jboss-as/bin
File  Edit  View  Search  Terminal  Help
10:05:51,717 INFO  [EsbDeployment] Stopping 'Quickstart_helloworld.esb'
10:05:51,978 INFO  [EsbDeployment] Destroying 'Quickstart_helloworld.esb'
10:05:51,978 WARN  [ServiceMessageCounterLifecycleResource] Calling cleanup on e
xisting service message counters for identity ID-8
10:05:52,024 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_gw]
 stopped
10:05:52,025 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_esb
] stopped
10:05:52,089 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_esb
] started, fullSize=200000, pageSize=2000, downCacheSize=2000
10:05:52,099 INFO  [QueueService] Queue[/queue/quickstart_helloworld_Request_gw]
 started, fullSize=200000, pageSize=2000, downCacheSize=2000
10:05:52,121 INFO  [EsbDeployment] Starting ESB Deployment 'Quickstart_helloworl
d.esb'
10:06:25,331 INFO  [STDOUT] &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
10:06:25,331 INFO  [STDOUT] <Your Name> was here
10:06:25,332 INFO  [STDOUT] Body: Hello World
10:06:25,332 INFO  [STDOUT] &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
10:06:25,333 INFO  [STDOUT] Message structure:
10:06:25,333 INFO  [STDOUT] [Hello World].
10:06:25,333 INFO  [STDOUT] Message structure:
10:06:25,333 INFO  [STDOUT] [Hello World].
10:06:25,333 INFO  [STDOUT] Message structure:
10:06:25,333 INFO  [STDOUT] [Hello World].
```

Illustration 18: Running with additional actions

The last thing you can do from the quickstart is undeploy the running instance. You can do that by running the command:

```
ant undeploy
```

As seen below:

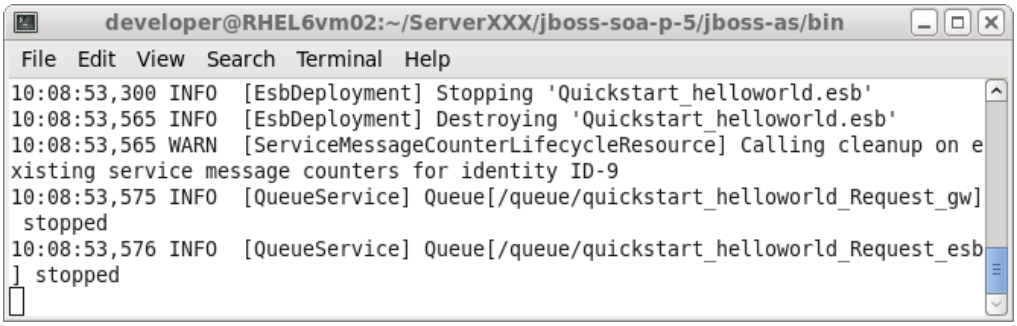


Illustration 19: Undeploy the quickstart

Congratulations on deploying, running, updating, and undeploying your first SOA-P quickstart and first SOA-P deployment! You will likely find the other quickstarts to be a wealth of excellent examples as you start creating more complex ESB services.

Lab Number 3: Installation of JBDS

Get the File

In the ${USER_HOME}Downloads/JBDS directory you will find the JBDS installer, which for Linux will look something like this:

```
jbdevstudio-product-eap-linux-gtk-x86_64-4.0.0.v201102161941R-H180-GA.jar
```

Running the Installer

The next step is to run the installer. The command to do that is very simple:

```
cd ${User_Home}/Downloads/JBDS
```

type in java -jar and j and the tab key to bring up the correct file

```
java -jar ${The_File_Available_For_Linux_Above}
```

A command prompt will be used for this and on Linux it looks like this

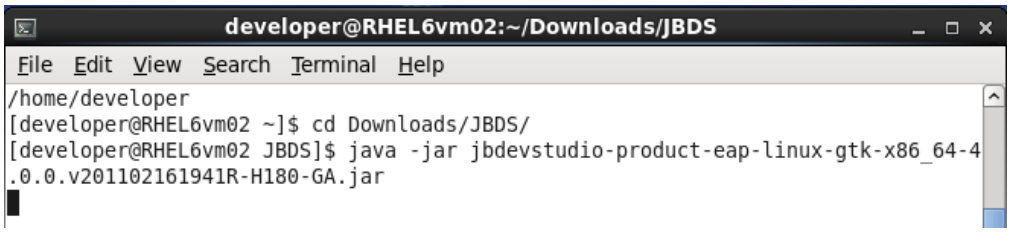


Illustration 20: Running JBDS Installer

Running this command will then get you to the next step in the process: the visual installer. It will look something like this:

Illustration 21: JBDS Installer Welcome Screen

Select the Next Button and this will get you to this screen:

Illustration 22: Accept License

Please select the “I accept....” radio button to enable the next button. Click the next Button.

The next screen prompts you on where to install JBDS. Make the name unique and don't just select the default. Make sure you write down or remember this path. Please make sure it does not have spaces in it, as older versions had problems running with spaces in the installed path and it should be fixed, but you never know. Also if you already have an existing copy of JBoss Developer Studio installed please raise your hand. Also please put in something unique in the name (like your initials) so that someone else after you will able to install this lab without any problems.

Illustration 23: Select the Installation Path

Make any changes needed to the installation path and then please select Next. You will be prompted to make this directory if it does not exist. If you do not see a prompt like this, select previous and make the installation path unique.

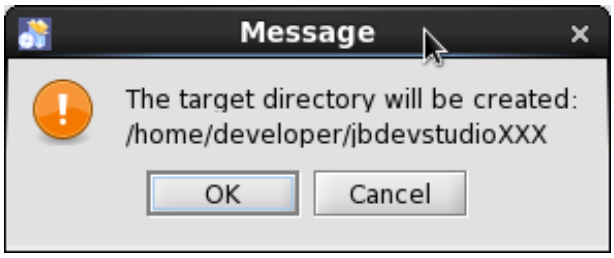


Illustration 24: Confirm Installation Directory

Press OK to confirm that the directory can be created.

Next you will be prompted to use the JDK that was used from your path/java_home properties. No changes are needed with this setting.

Illustration 25: Select Default JVM

Please select Next

The next box will prompt you to install and make available the embedded EAP instance. Please leave this section alone and click Next. Later, we will show you how to add a SOA-P instance that you already installed. You also could add another JBoss EAP/AS instance, but for the purposes of this lab this is not required and not recommended.

Illustration 26: Select existing platforms

You could choose to click the add button and find the SOA-P instance, however we will show you how to do that in the next lab. You could also click find, and hunt down the the installed instance that way.

Select Next

The next screen just shows you what will be installed:

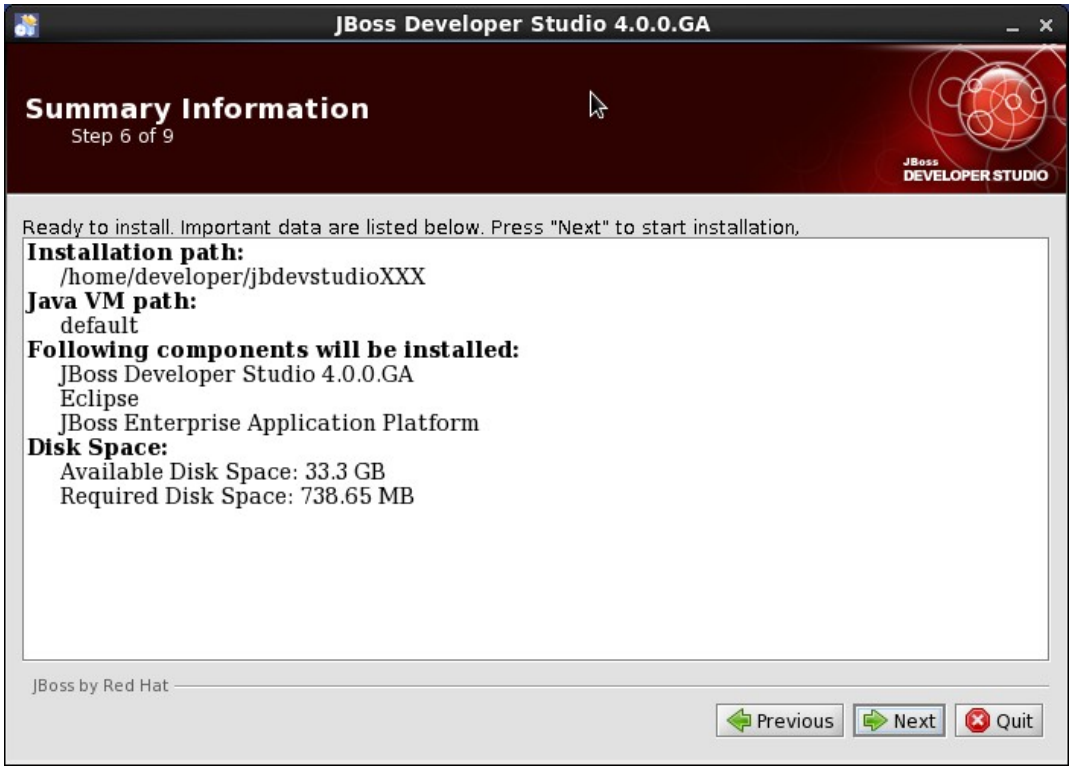


Illustration 27: Confirm installation settings

Click Next

The installer will run for a few minutes, expanding and copying all the necessary files.

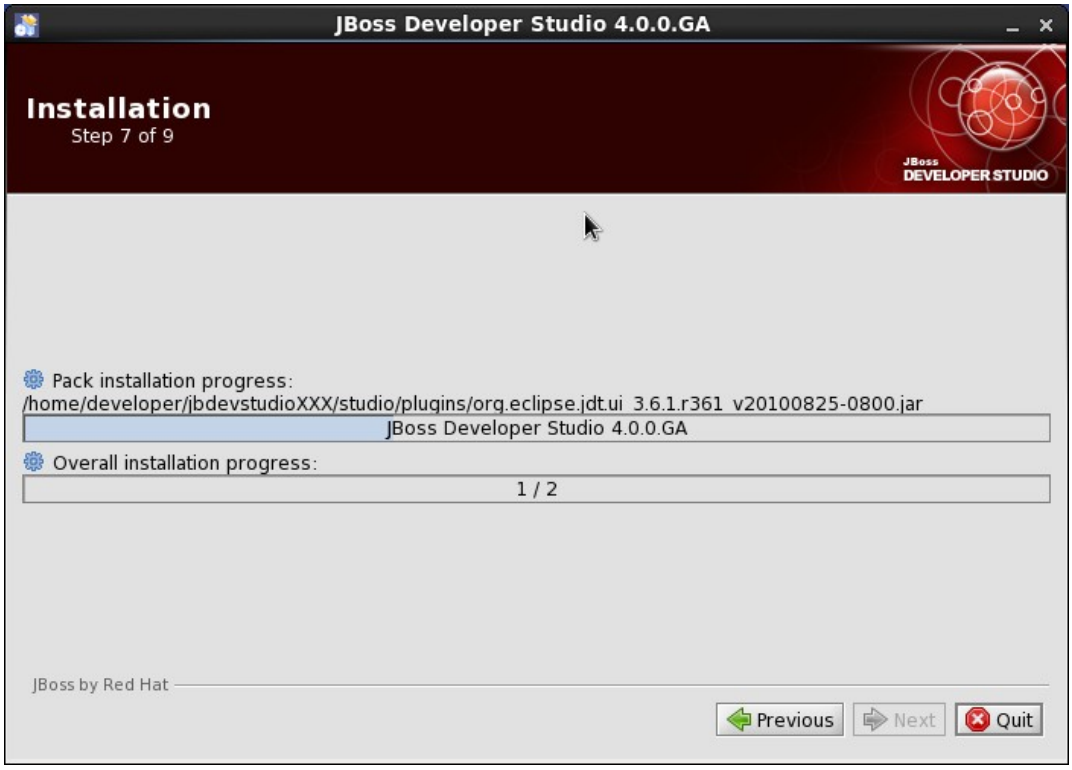


Illustration 28: Running installation

Wait for Next to be available to you once the installer is completed.

The next step will ask where to place short cuts and the like. The defaults are fine for this lab.

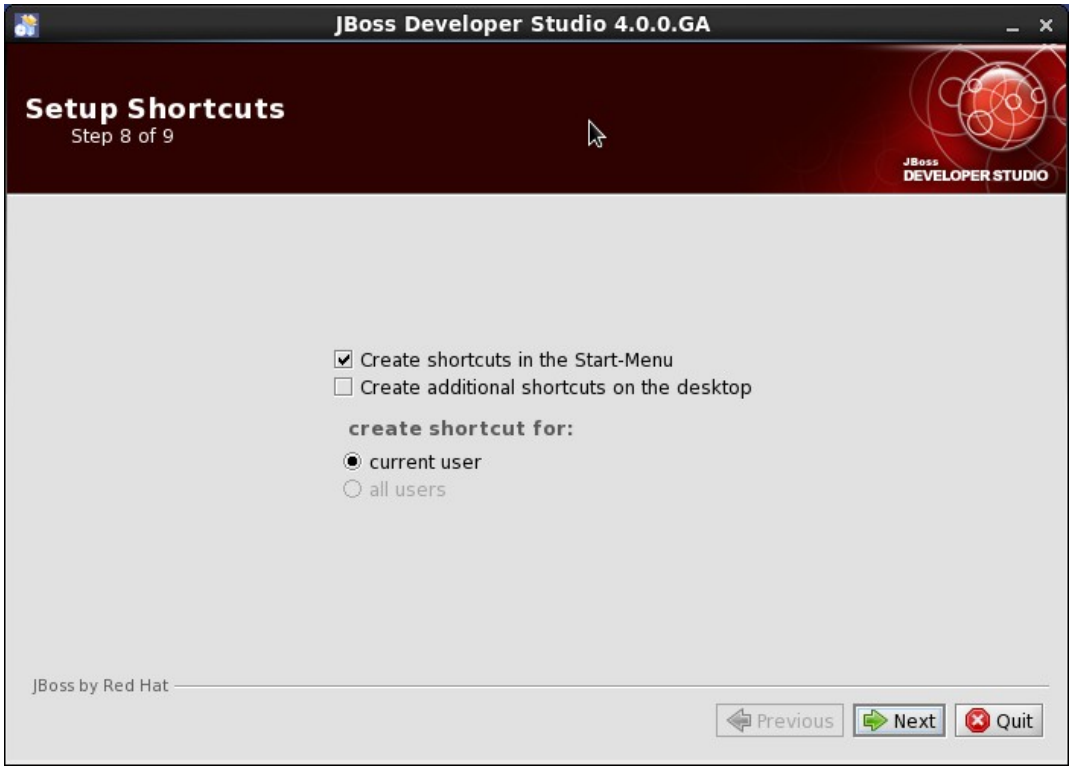


Illustration 29: Shortcut locations

Select Next.

Congratulations you have installed JBDS 4.0.

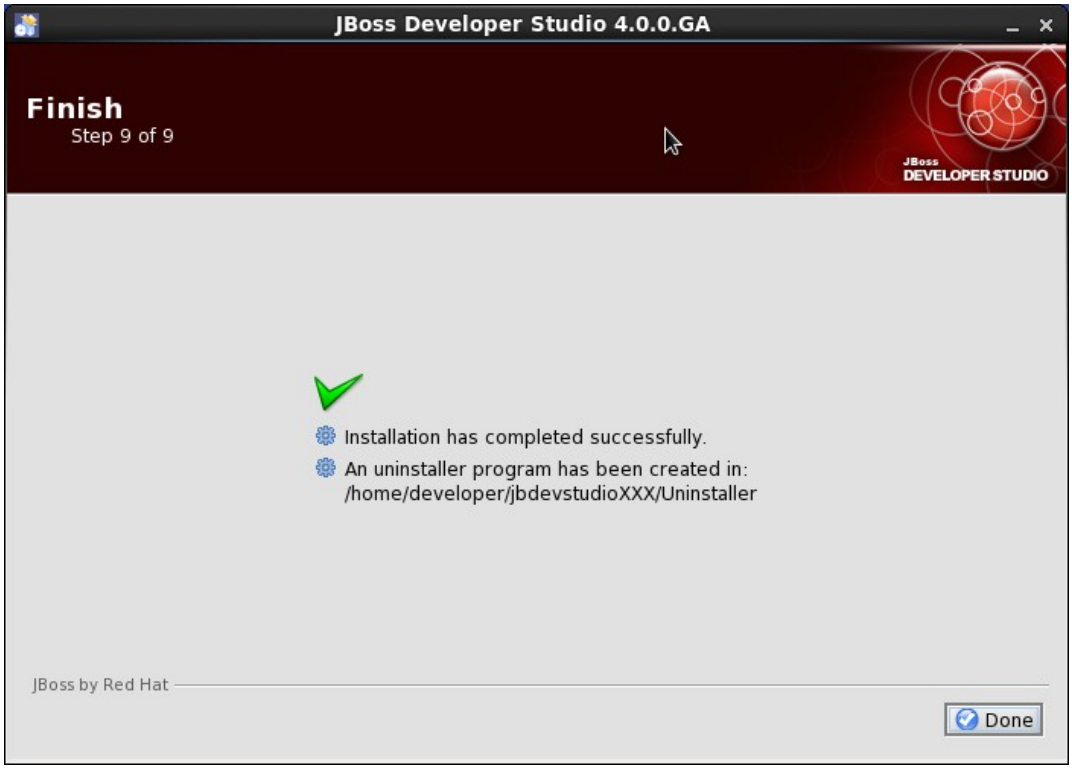


Illustration 30: Installation complete

Please Select Done.

Congratulations you are now done installing JBDS 4.0 on your local workstation. You have now completed this lab and we will next create a new Seam Project using the already running HSQLDB.

Lab Number 4: Configure SOA-P in JBDS

Start JBDS

To start JBDS you will have to find the JBDS instance we just installed. There should be shortcuts in your programs menu or maybe even on your desktop depending on what you selected above when installing. On Linux it will be installed in Applications -> Programming -> JBoss Developer Studio 4.0.0.GA as seen below. You might need to restart your machine to see this:

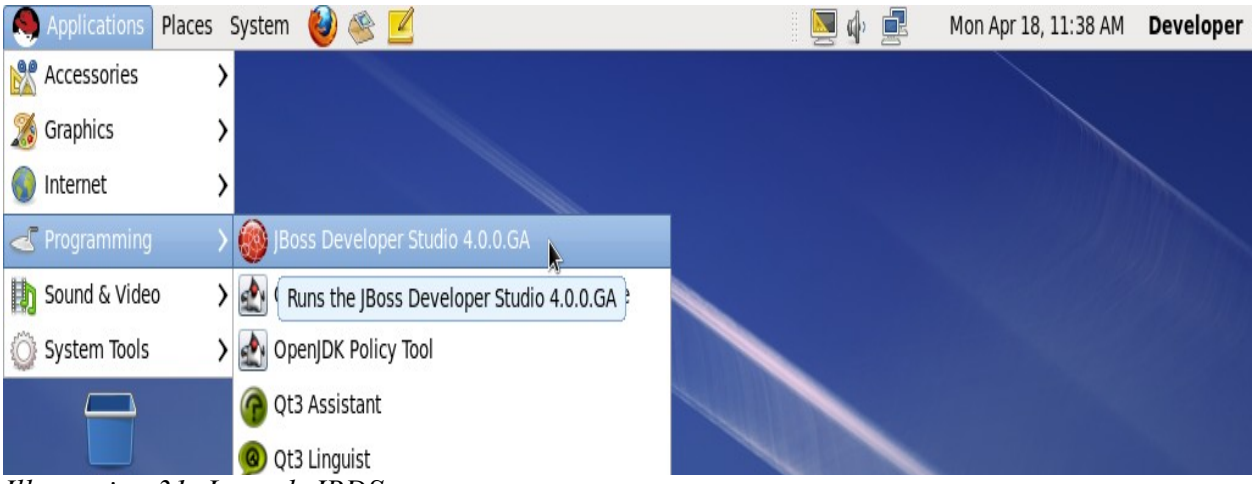


Illustration 31: Launch JBDS

Select a Work Space

Please when selecting a workspace select a new directory or one that does not exist yet:

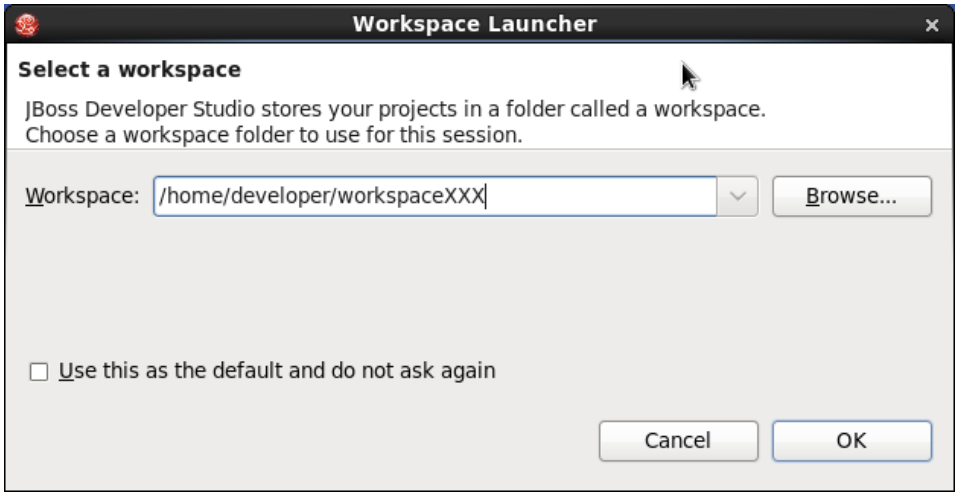


Illustration 32: Select workspace

Select OK and wait for JBDS to fully open

JBDS Start Page

You will then be brought to the main landing page. This page comes up every time you create and open a new Work Space.

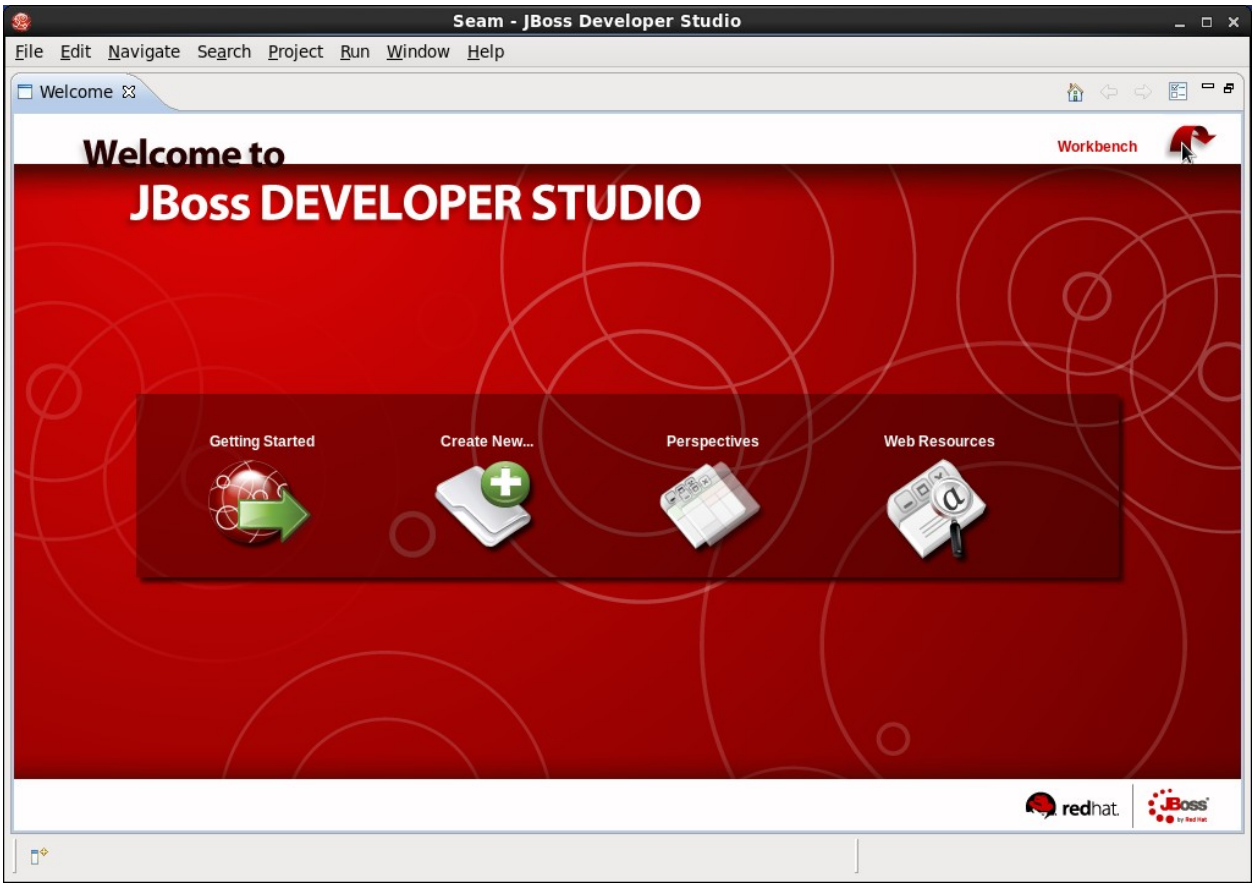


Illustration 33: JBDS initial screen

Select the Workbench Icon above which will bring up JBDS as shown below:

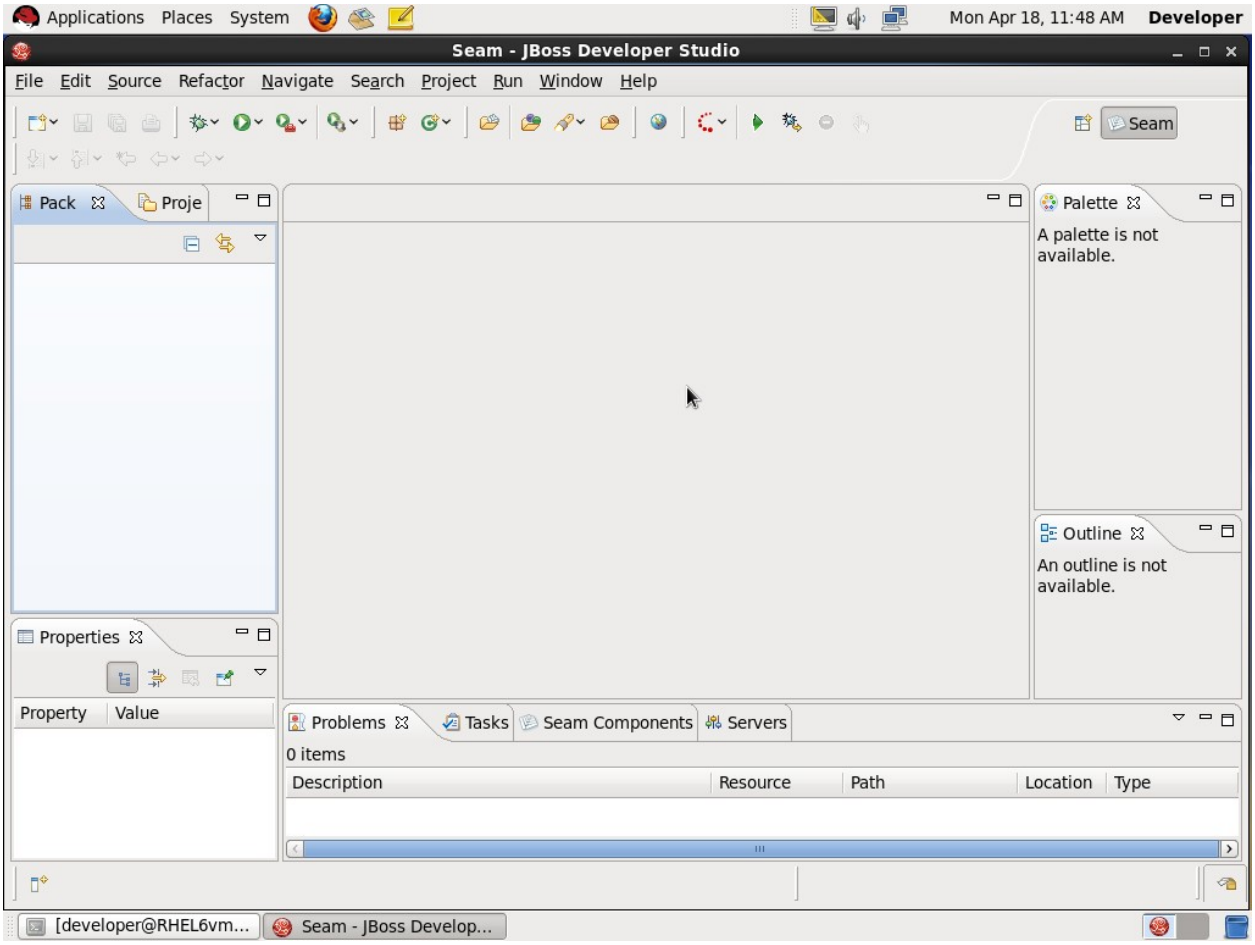


Illustration 34: JBDS initial perspective

`Change the Perspective`

This is not the correct perspective, so open Window -> Open Perspective -> Other as shown:

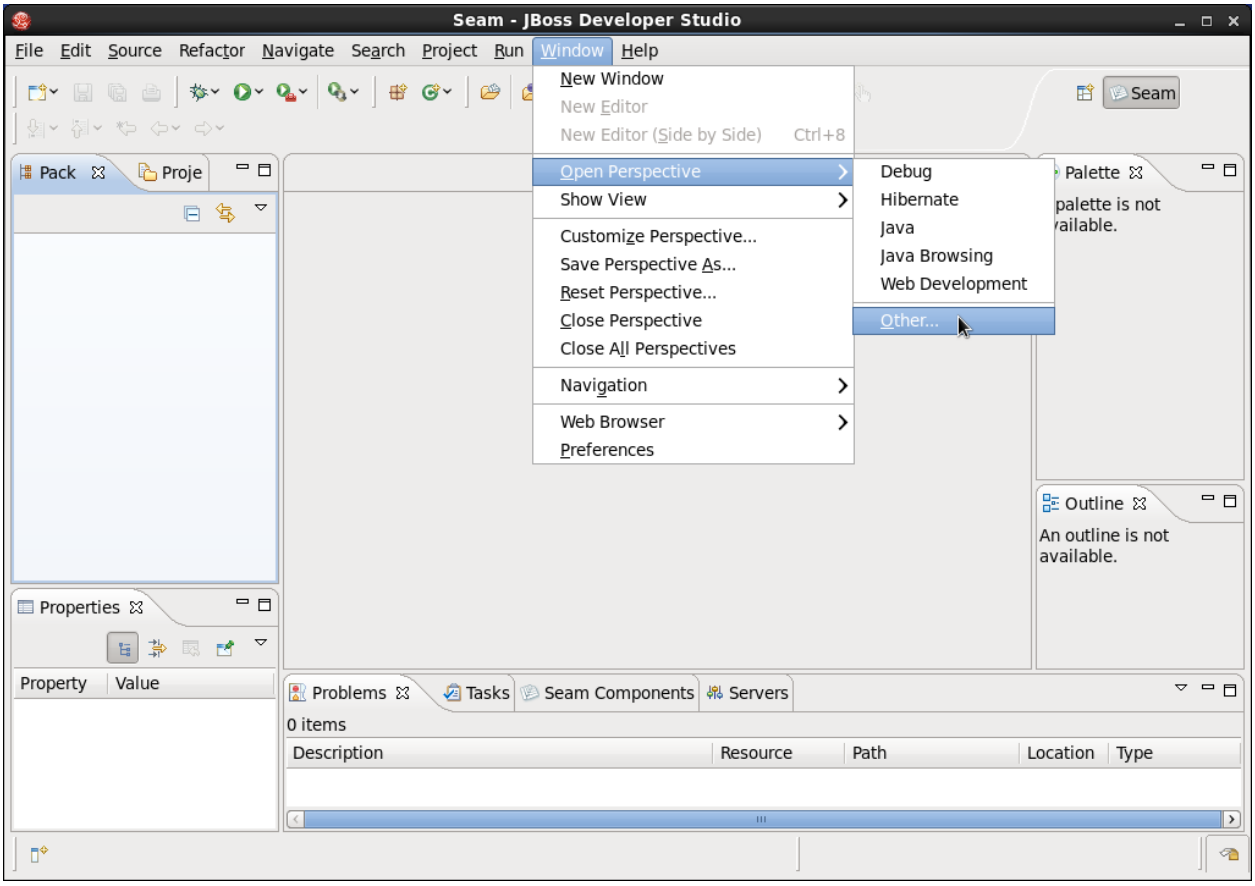


Illustration 35: Changing perspective

Select JBoss AS from the perspective list as shown:

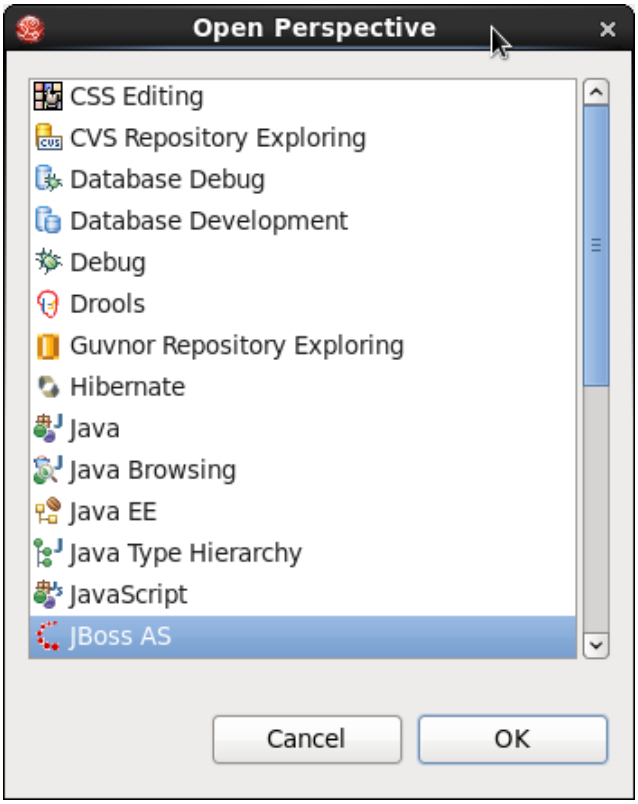


Illustration 36: Select JBoss AS perspective

That will bring you to a screen that looks like this:

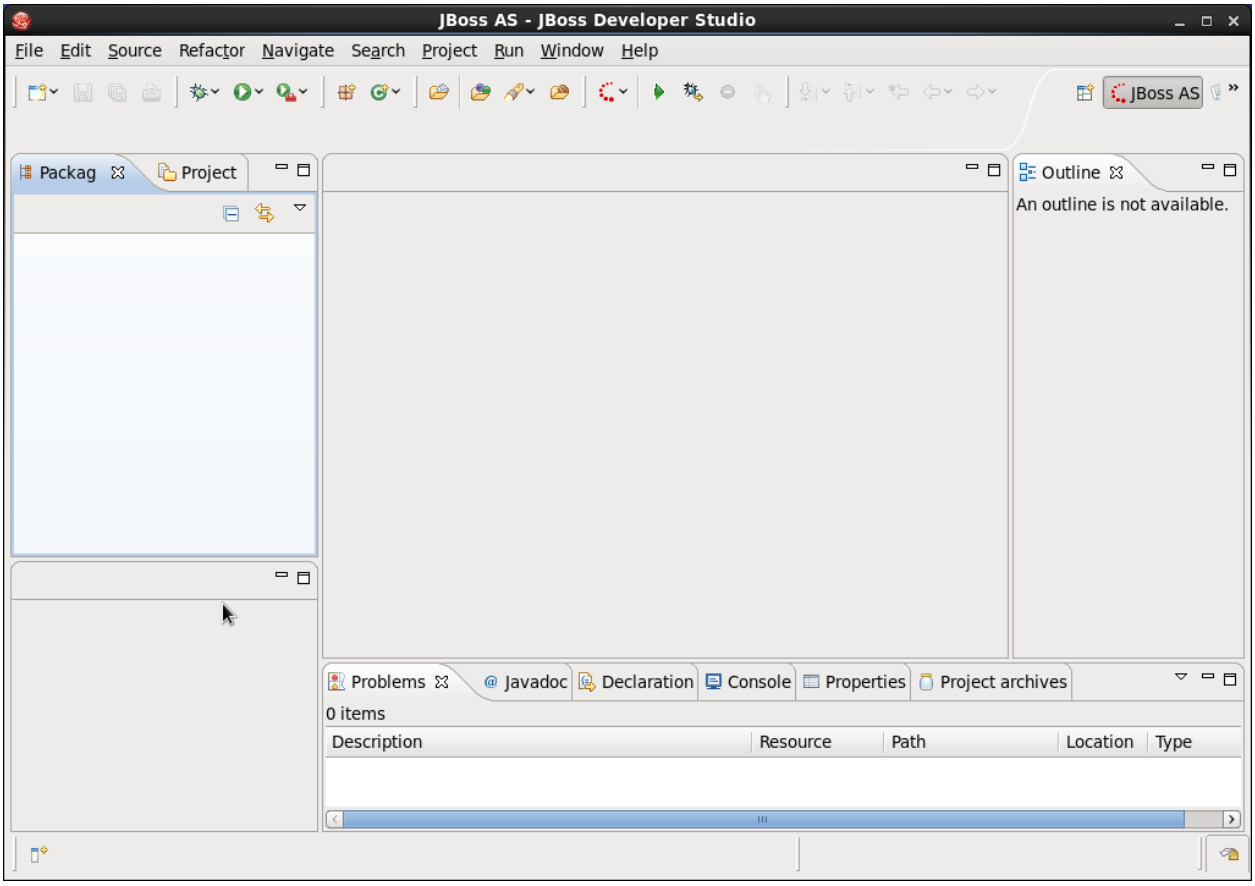


Illustration 37: JBoss AS Perspective

Create a New Server

Create a new server instance using the menus as shown below:

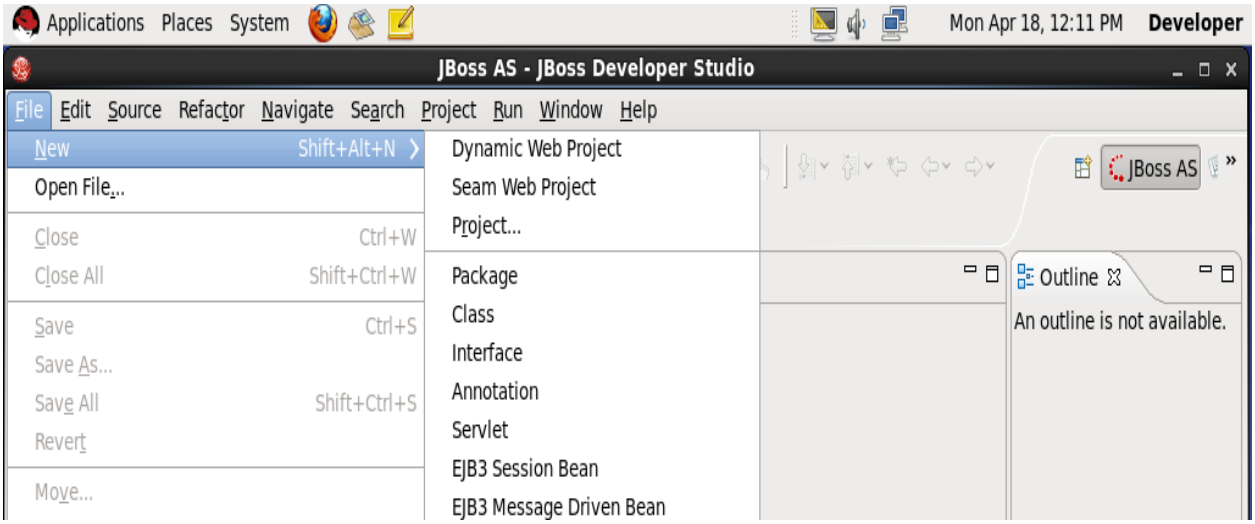


Illustration 38: Select a wizard

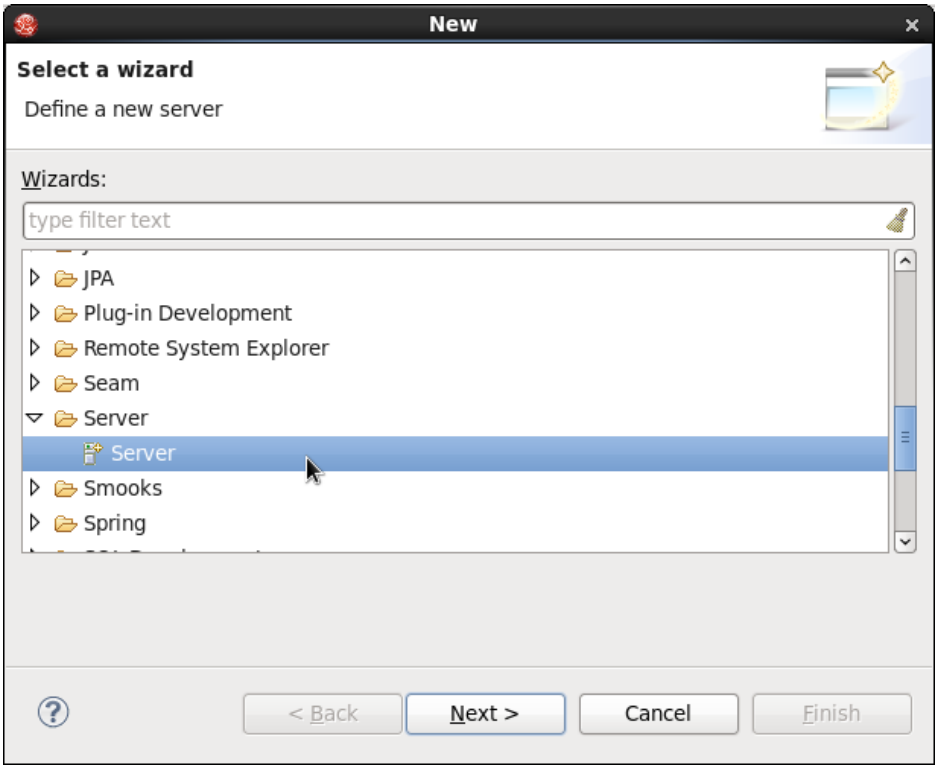


Illustration 39: Select server wizard

Select Next, which will open up a dialogue box. Please select JBoss Enterprise Application Platform 5.x and change the server name to soa-p Runtime Server as shown:

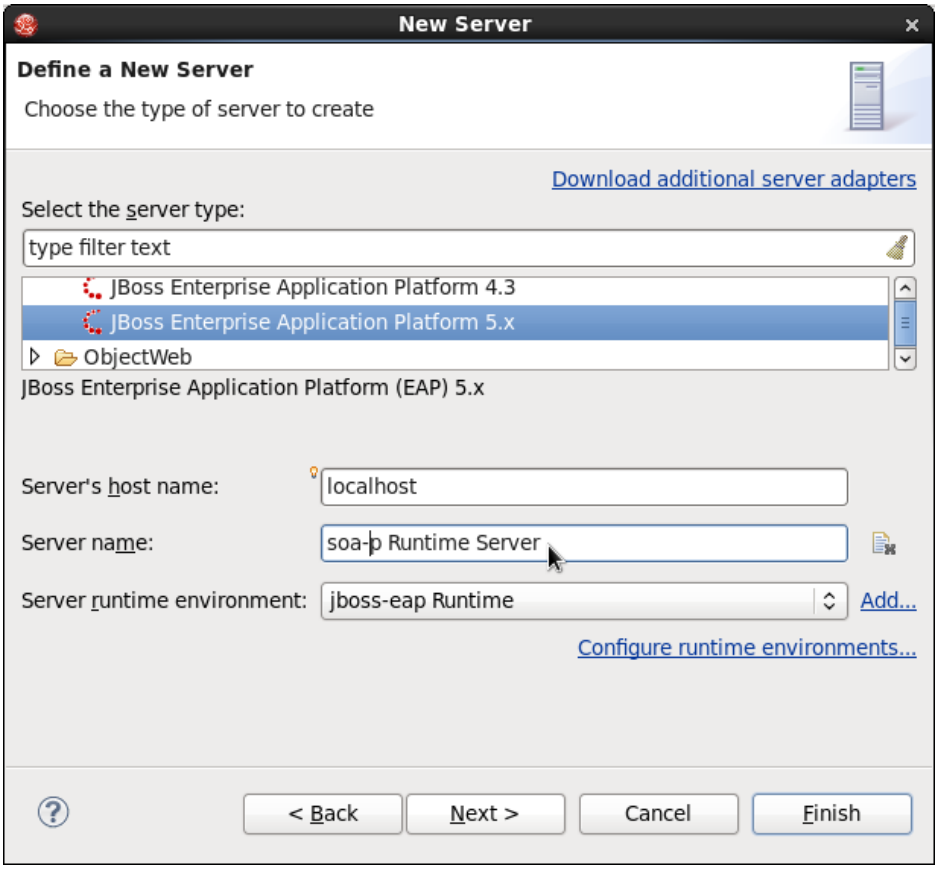


Illustration 40: Define new server

Please click "Add..." to the right of the jboss-eap Runtime.

Please select the Browse button and navigate to the SOA-P instance we installed in the previous lab. Also note that the default, minimal and other configurations are made available in the bottom part of the dialogue box. Make selections as shown:

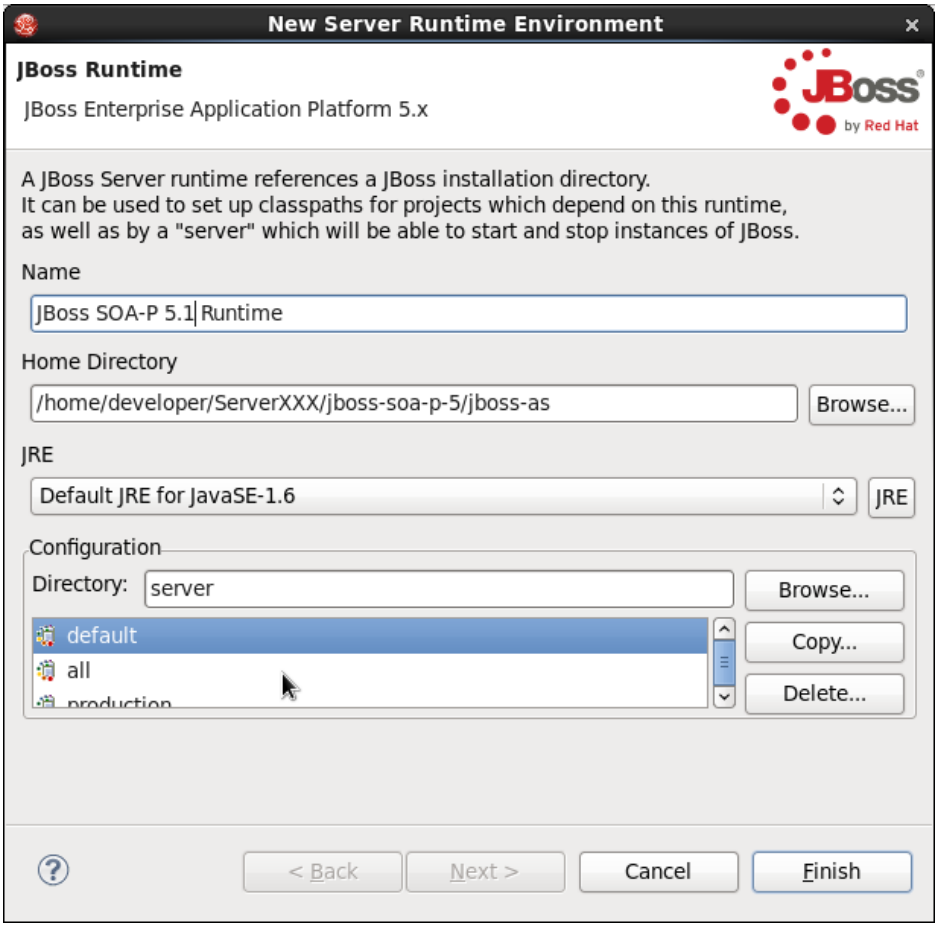


Illustration 41: Configure runtime

Please click on Finish

This brings you back to the New Server Dialogue

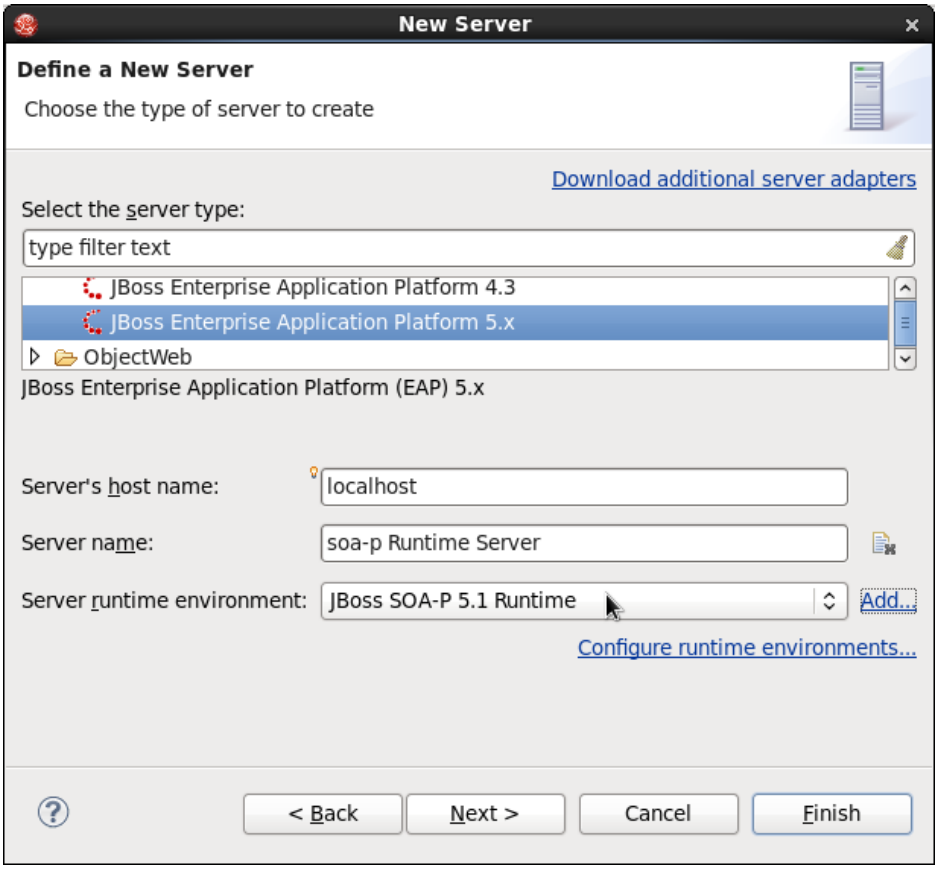


Illustration 42: Finish runtime configuration

Please select Next. Notice that the JBoss SOA-P 5.1 Runtime is now in the server runtime environment.

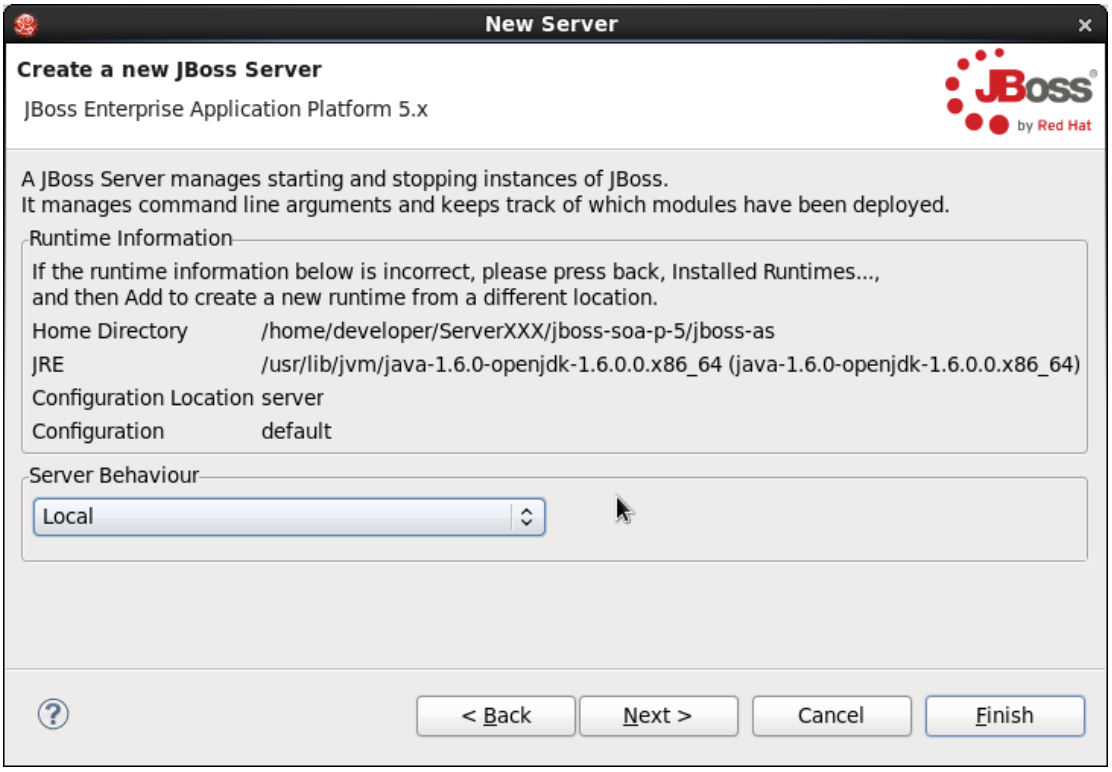


Illustration 43: Create the new server

Select Next

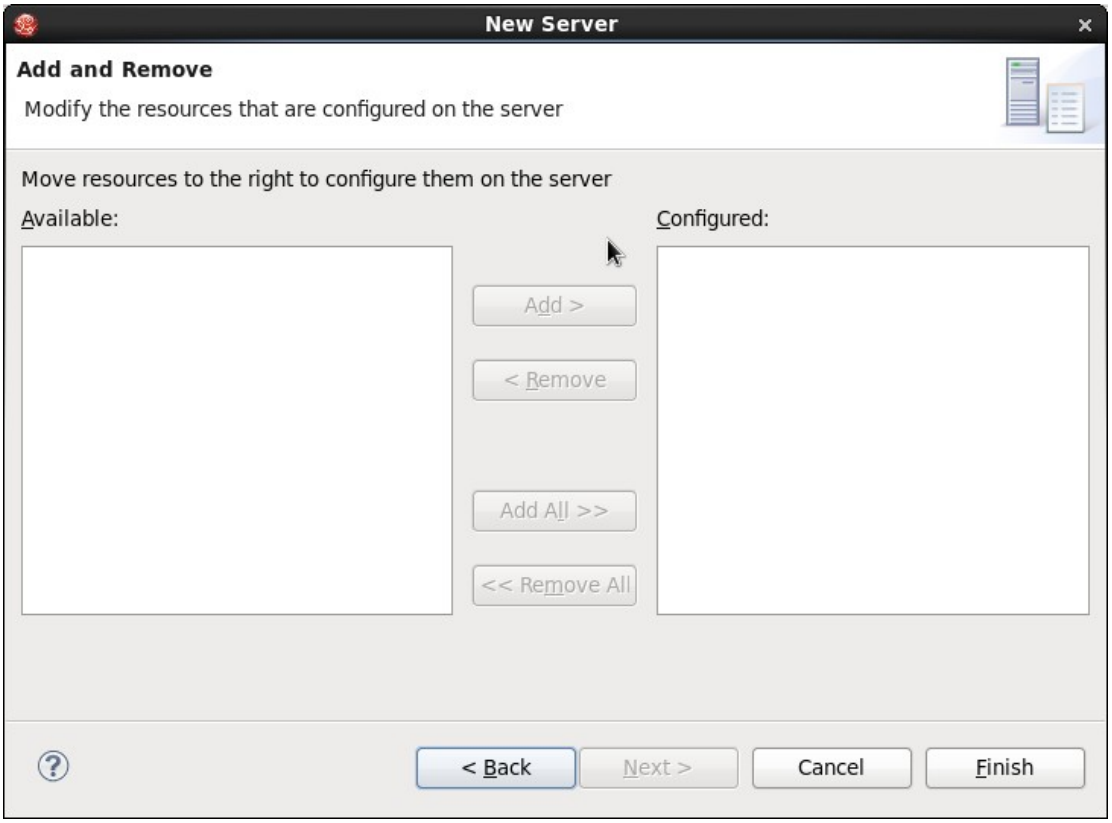


Illustration 44: Finish creating server

Select Finish

To see the newly added server, you need to open the server view. Select Window->Show View->Other... from the main menu as shown below:

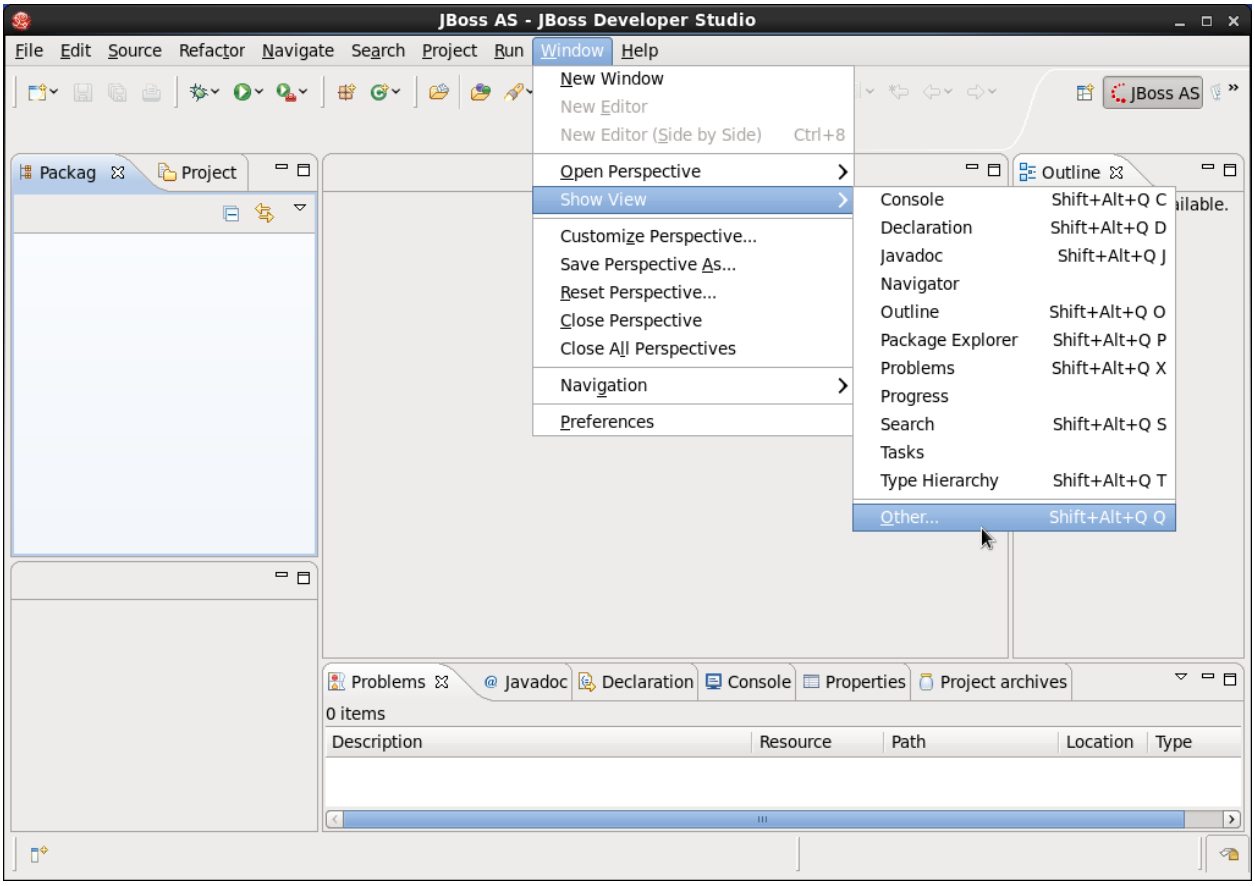


Illustration 45: Open server view

When the dialog appears, select Server->Servers as shown below:

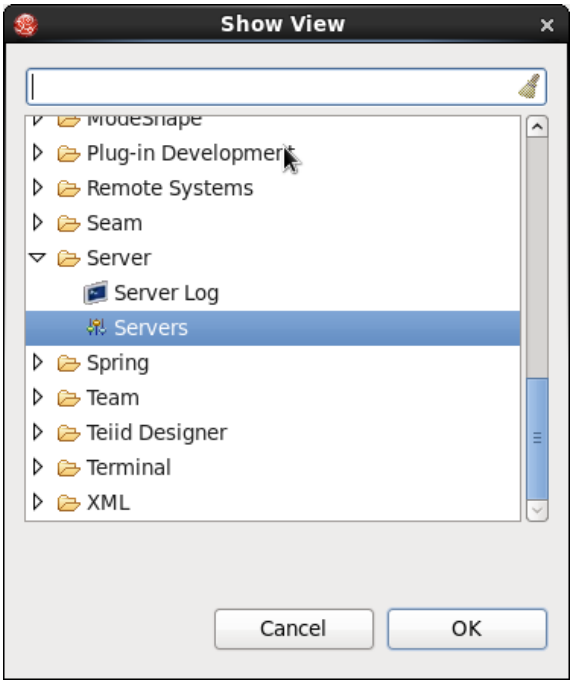


Illustration 46: Select server view

When the “Servers” tab appears near the bottom on the JBDS window, drag it over to the left-hand corner. You now have added in a new JBoss Server instance as shown below:

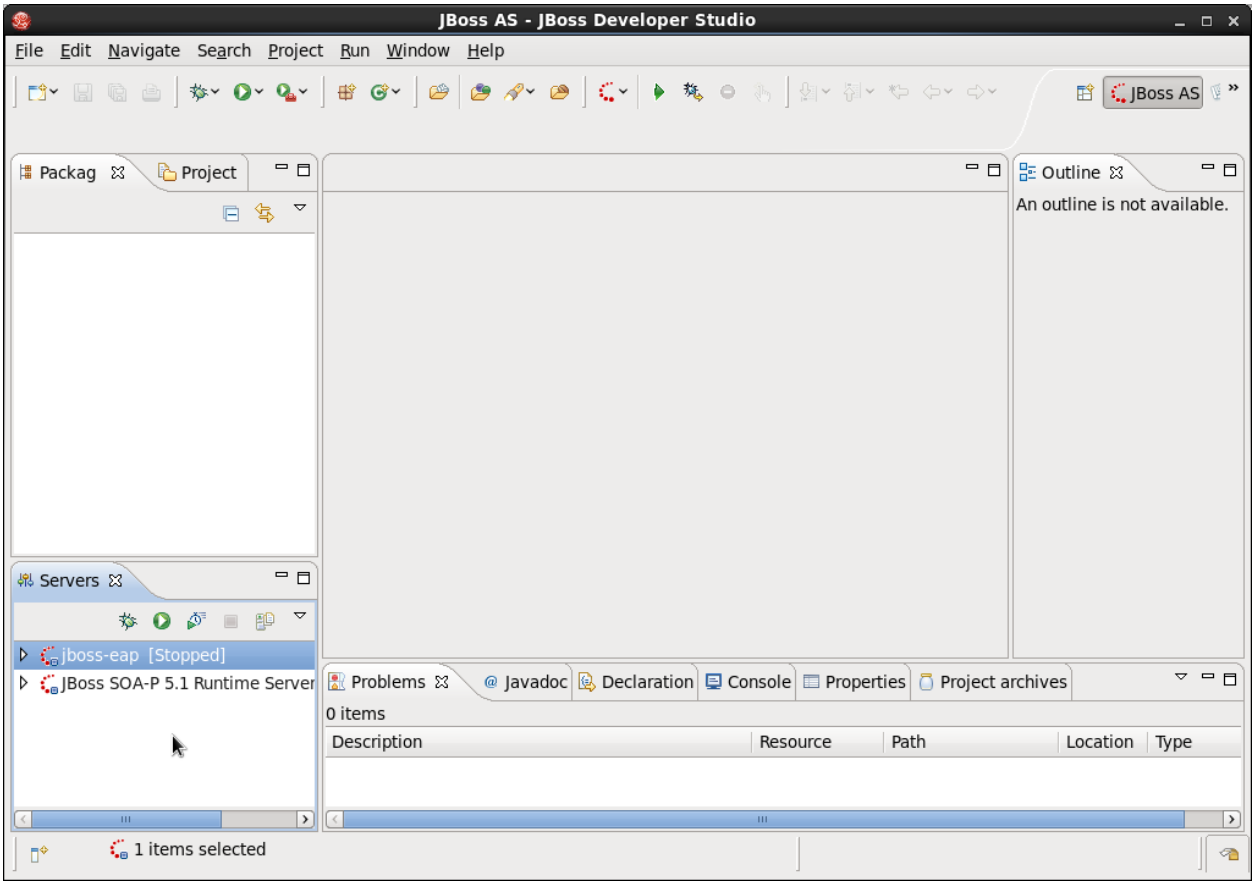


Illustration 47: Running server view

Start the Newly Created Server

Make sure you have have shut down the earlier running JBoss SOA-P instance via control-c and then right click on the JBoss SOA-P 5.1 Runtime and select “Start”. Notice that the console will start scrolling by as it did in the shell earlier.

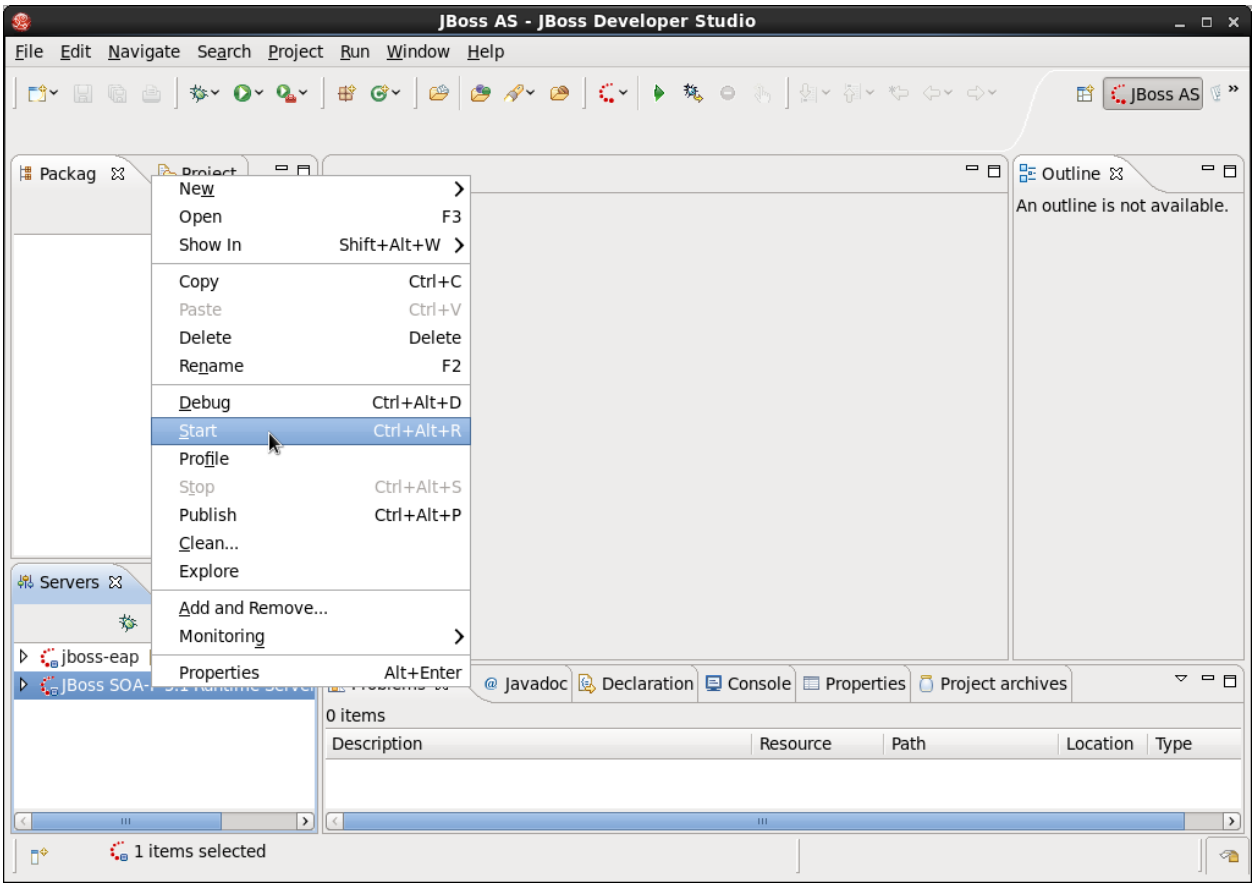


Illustration 48: Launch SOA-P server

When the server is done starting you can double click on the console tab to open it up full screen, and double click it again to minimize it as it was:

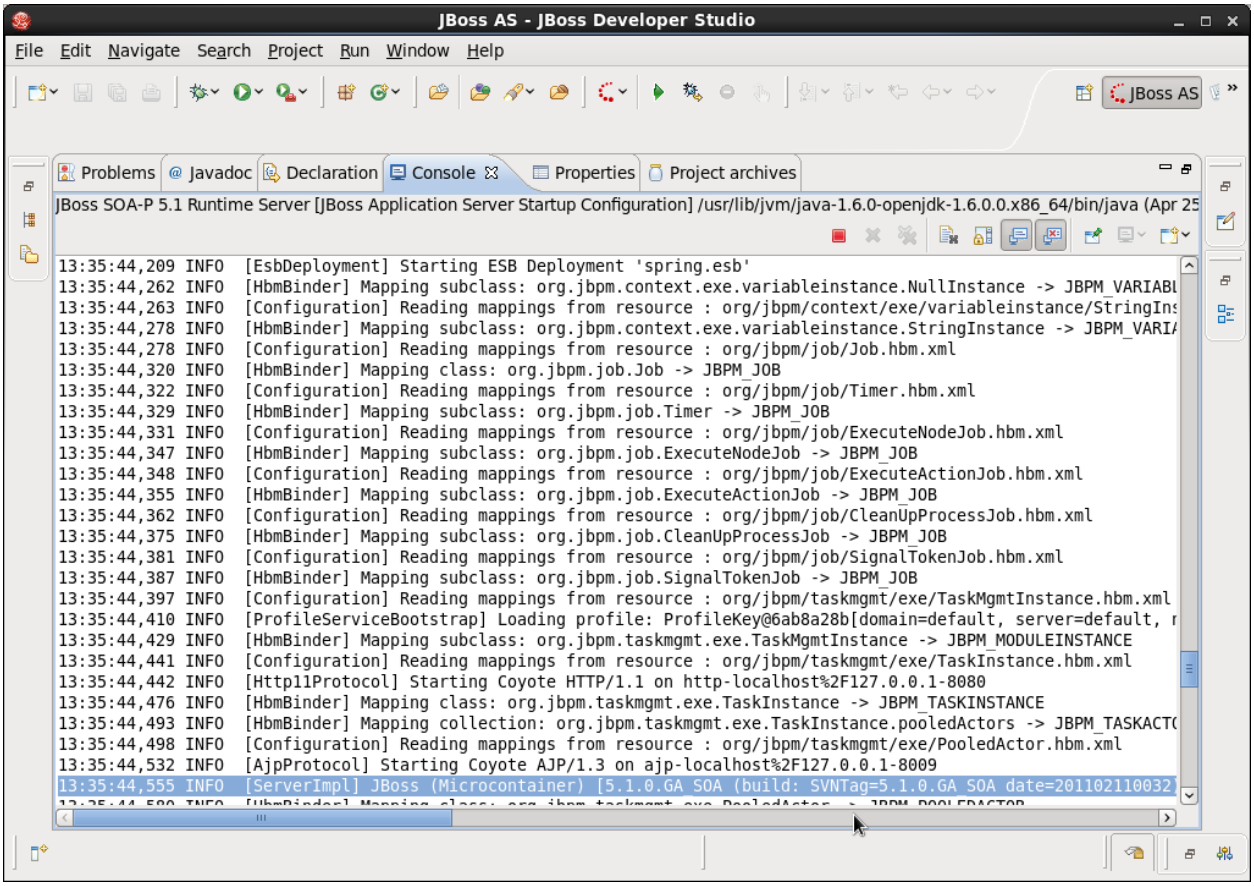


Illustration 49: Console log

If you do not see the console output please raise your hand and lets get this working for you. Congratulations you have completed this lab, installed JBoss SOA-P into JBDS, and now have it started from within the IDE. Now we can create our first ESB Project.

Lab Number 5: Creating Our First ESB Project

New ESB Project

The first ESB project we are going to create is going to be a simple project with one ESB service. This ESB service will poll a directory looking for a file with a given suffix. When a matching file is placed in the poll directory, it will read the contents of the file and send them as a message on the ESB.

To create our ESB project, please open File-> New -> Other This will give us this dialogue:

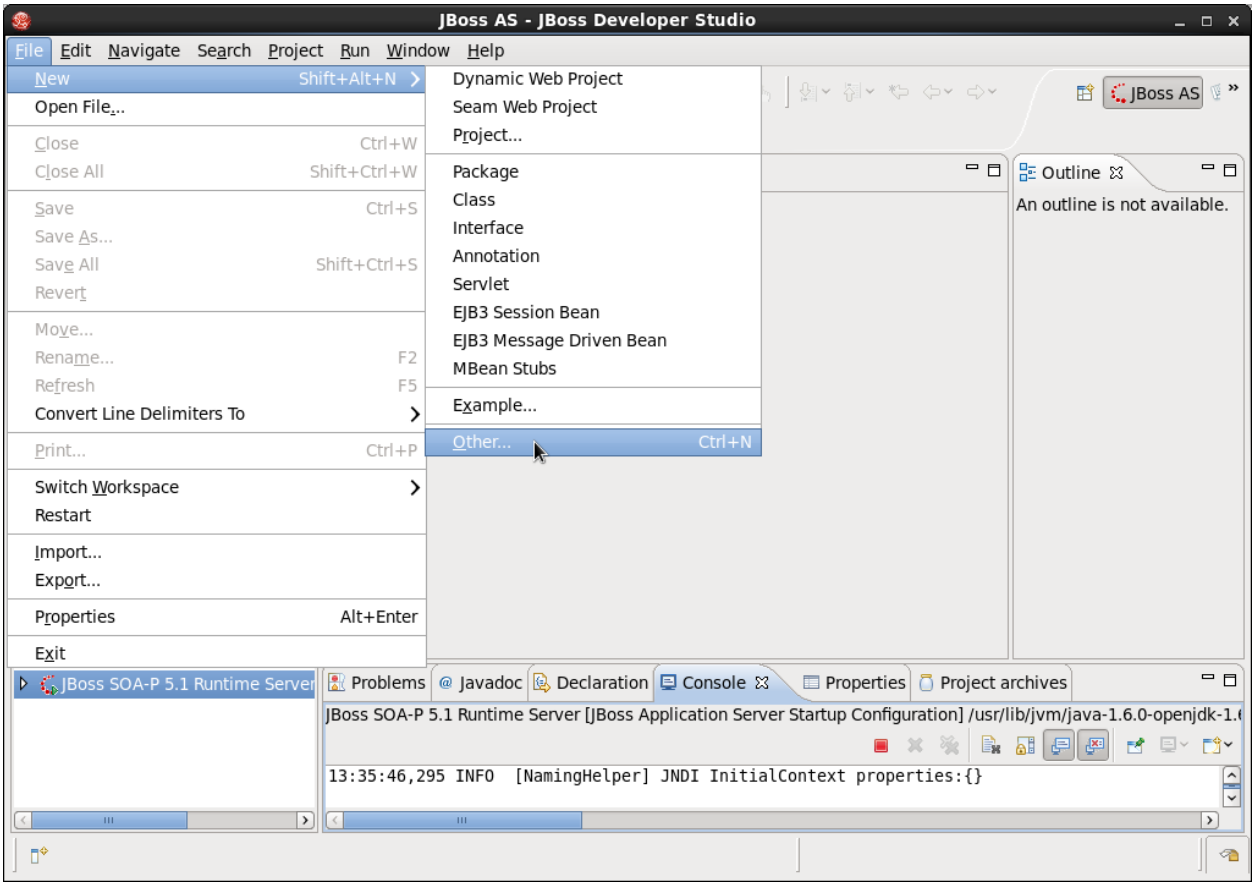


Illustration 50: Open wizard selection dialogue

In the “Select a wizard” dialogue, select “ESB Project” as shown below:

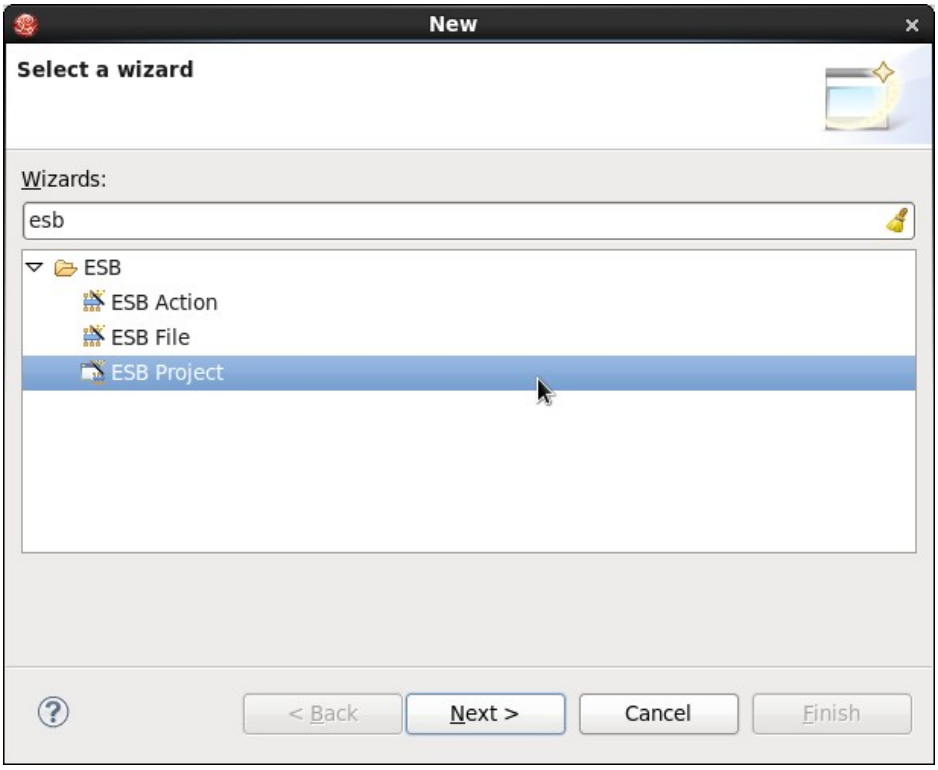


Illustration 51: Launch ESB project wizard

Then select Next which will bring up the next dialogue. Please type in the name “FilePollerProject” in the Project Name box, change the EAP runtime to JBoss SOA-P 5.1 Runtime and notice the JBoss ESB Version Number is 4.9 as shown below:

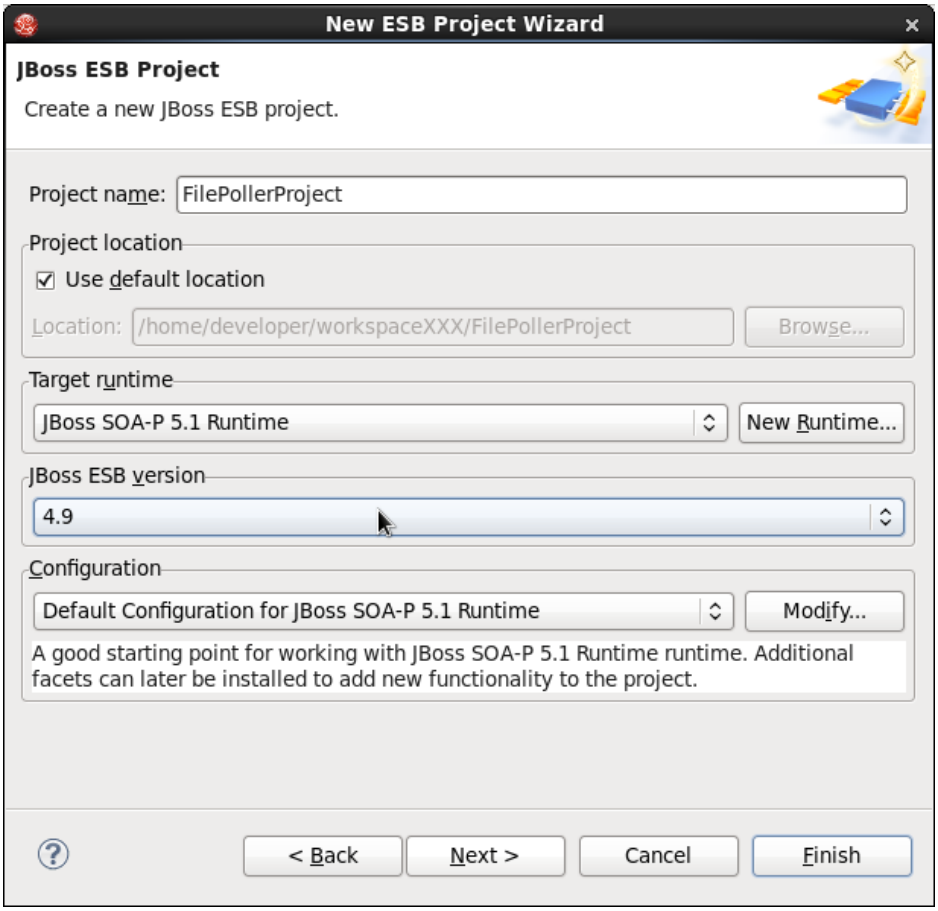


Illustration 52: Create ESB project

Click Finish

Artifact Editor

You should see that the project was created and the ESB configuration editor was automatically opened. This is actually the editor for the jboss-esb.xml file which you can find in the project by navigating to “Project Name -> esbcontent - > META-INF -> jboss-esb.xml” in the project explorer on the left of the JBDS window. It is highlighted in the screenshot below:

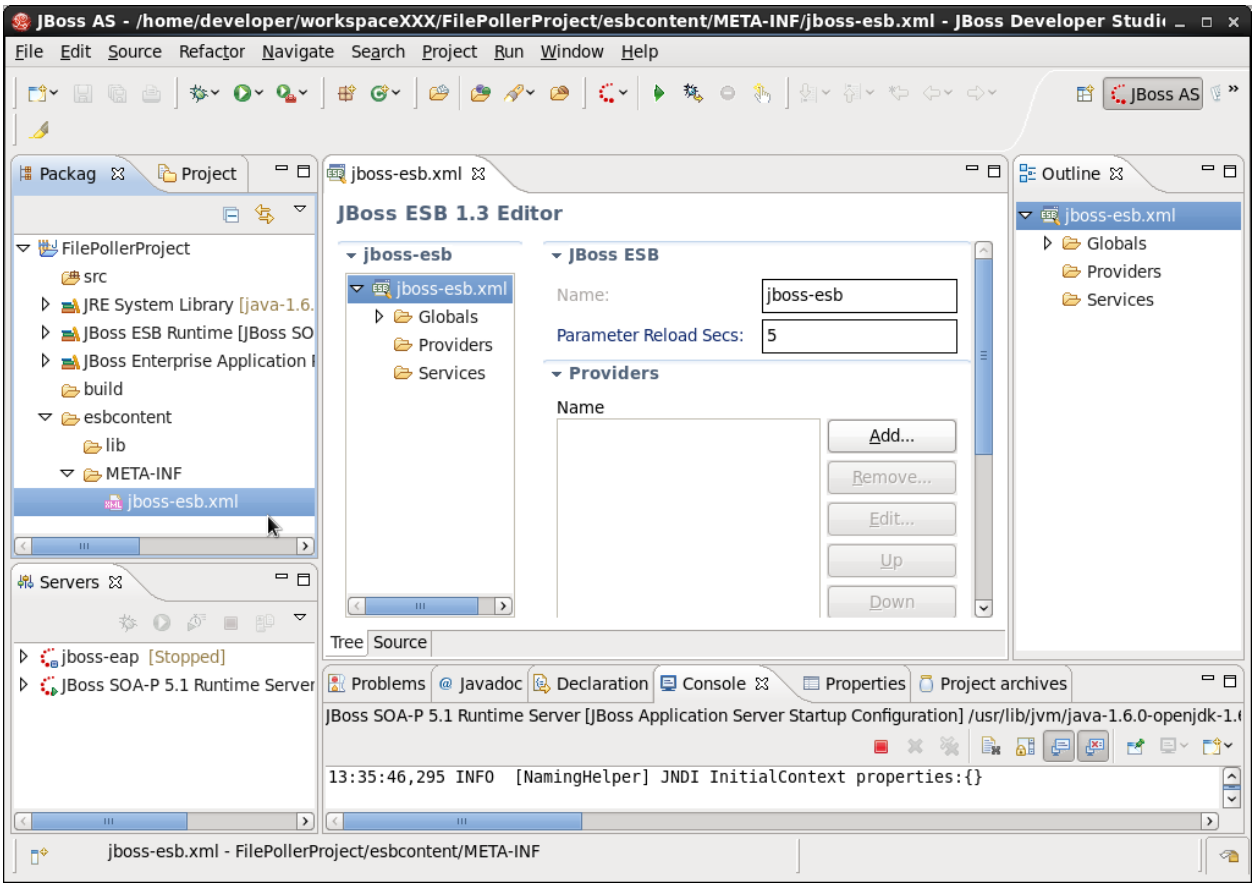


Illustration 53: Artifact Editor

Your First Provider

Now we need to create a gateway "provider". This is how we get messages onto the ESB. There are providers for HTTP, JMS, FTP, File, Email, JCA, Database, and a host of others – including the ability to create your own custom provider. We are going to create a fs-provider (File System) gateway. So, right-click on the "Providers" folder in the JBoss ESB Editor and select "New -> FS Provider...". You should see a dialog like this:

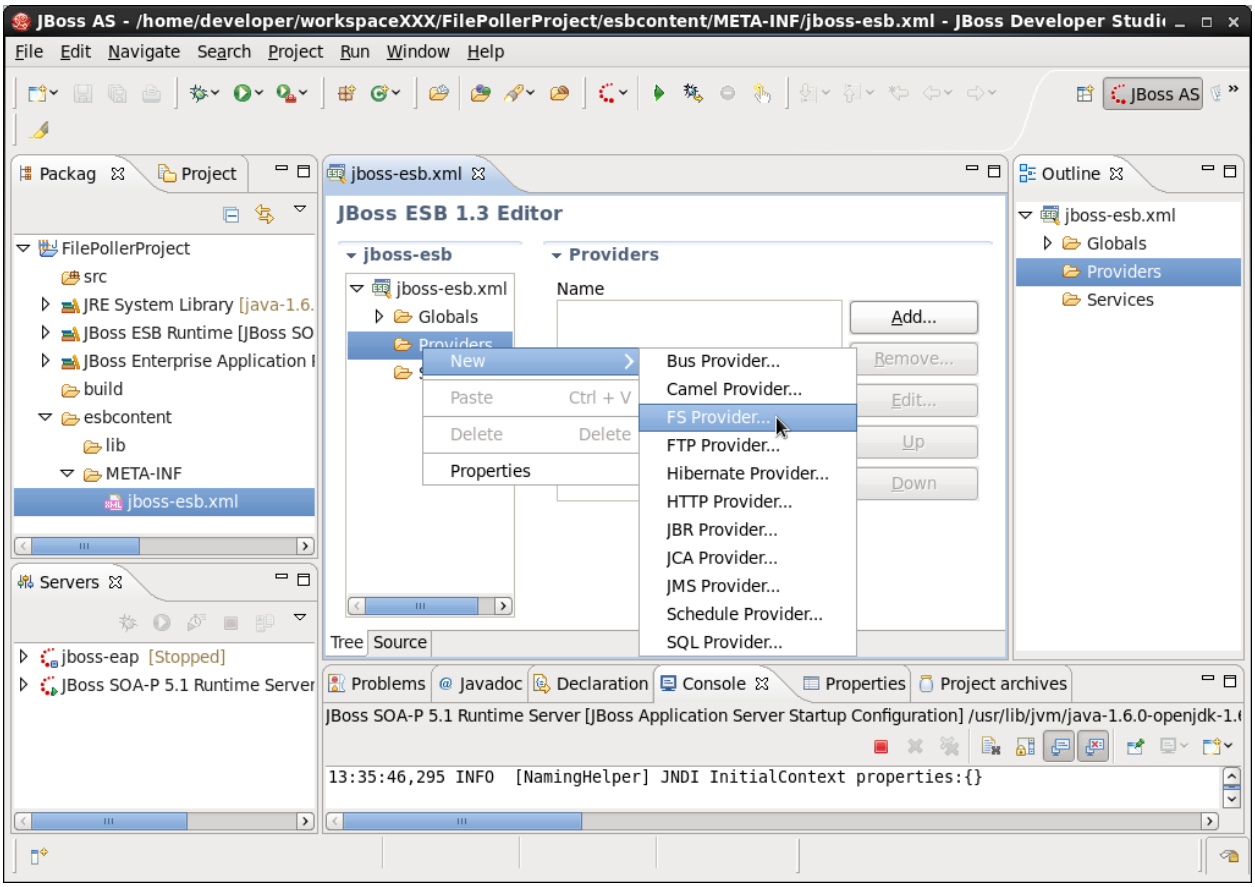


Illustration 54: Create new FS provider

Selecting FS Provider gets you this dialogue. Type in "File Poller Gateway":

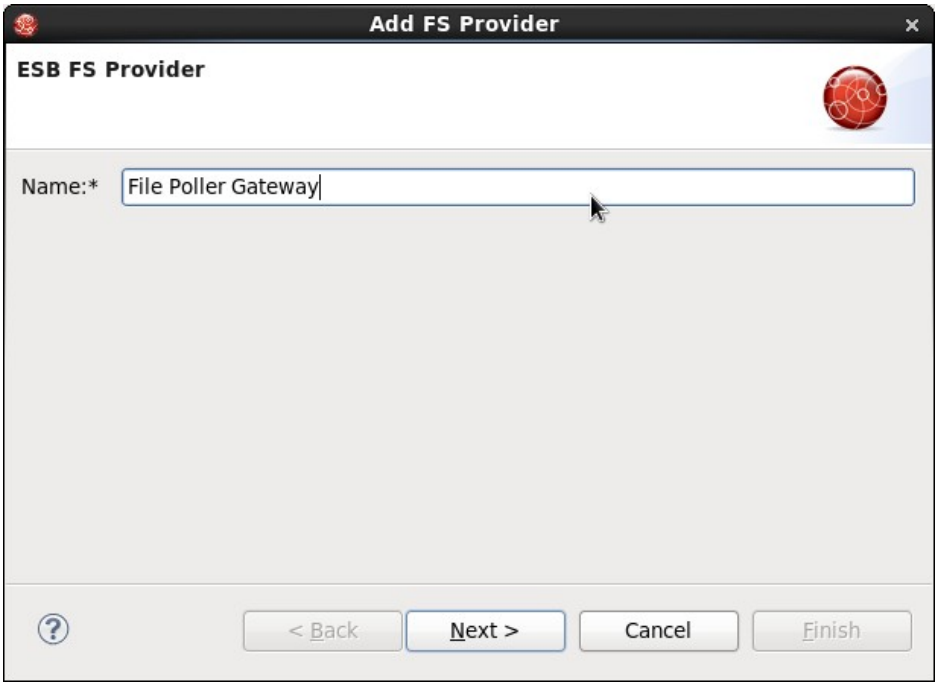


Illustration 55: Name FS provider

Select Next, that will get you to this Dialogue. Type in “File Poller GatewayID”:

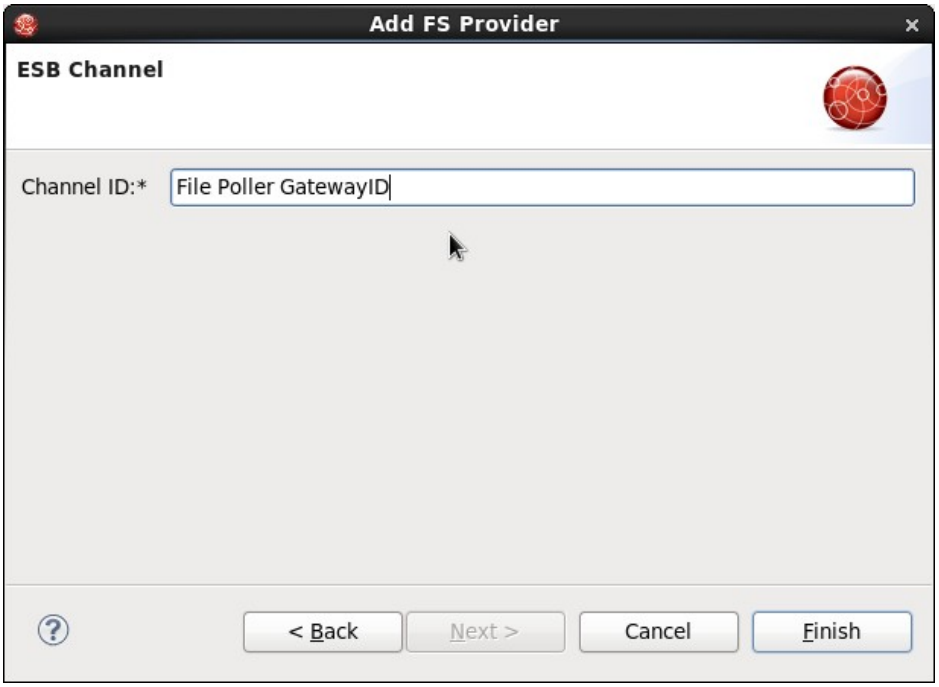


Illustration 56: Set channel ID

t

The Channel ID is the identifier that will be used later when a service declares that it wants to get messages from this gateway. I named the Channel ID the same as the provider name “File Poller Gateway”. Click “Finish”.

Now the file gateway provider has been added to our ESB configuration. However, we need to specify a filter so that the file gateway will know what files to look for. So, expand the File Poller Gateway tree as shown below and select the “Filter” node. This will allow us to configure a host of options. In this case, we only want to specify the directory to be polled and the suffix of the files we will be looking for. I am polling “/home/student/tmp” and looking for files that have an “esbfile” suffix:

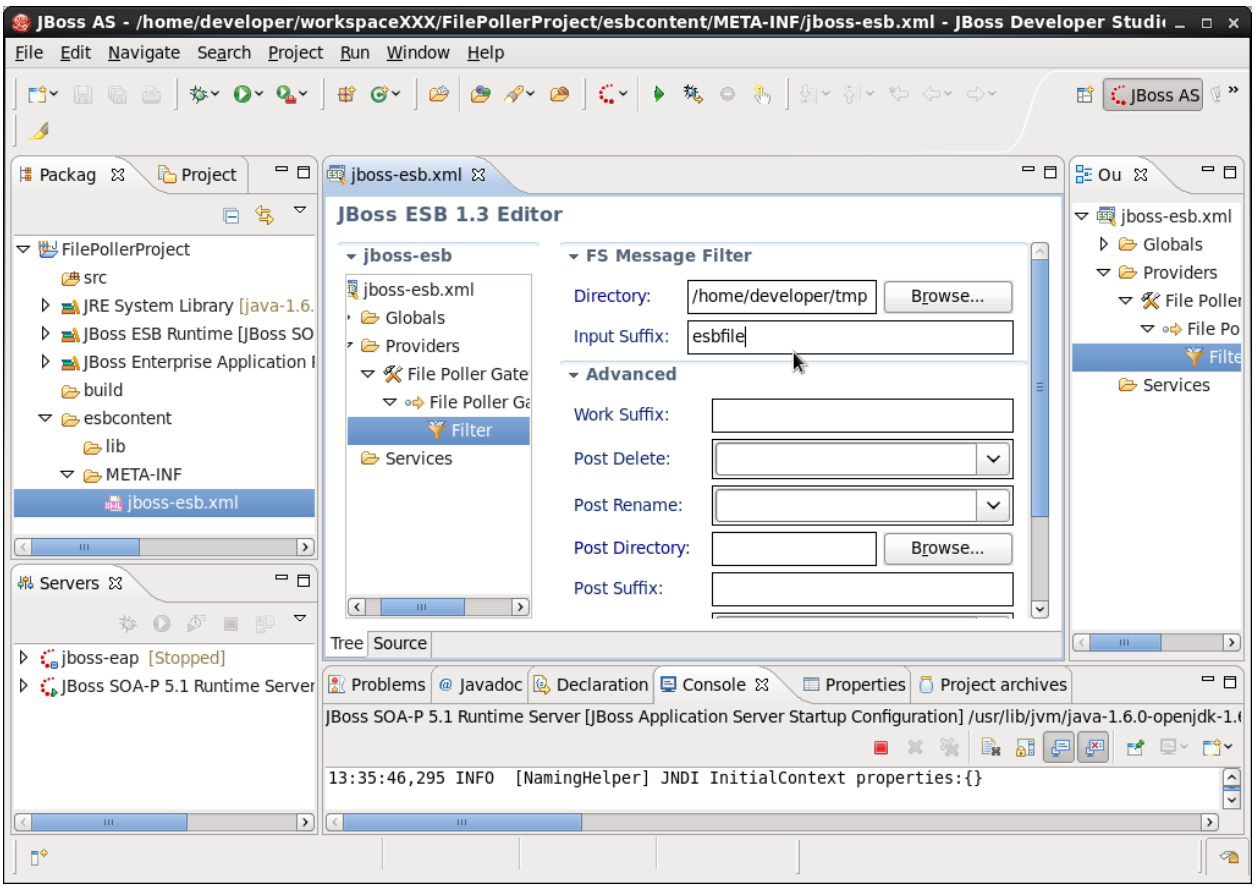


Illustration 57: Set message filter

Your First Service

Now we need to create a service that will consume messages from the provider we just created. Right-click on Services and select Add Service as shown below:

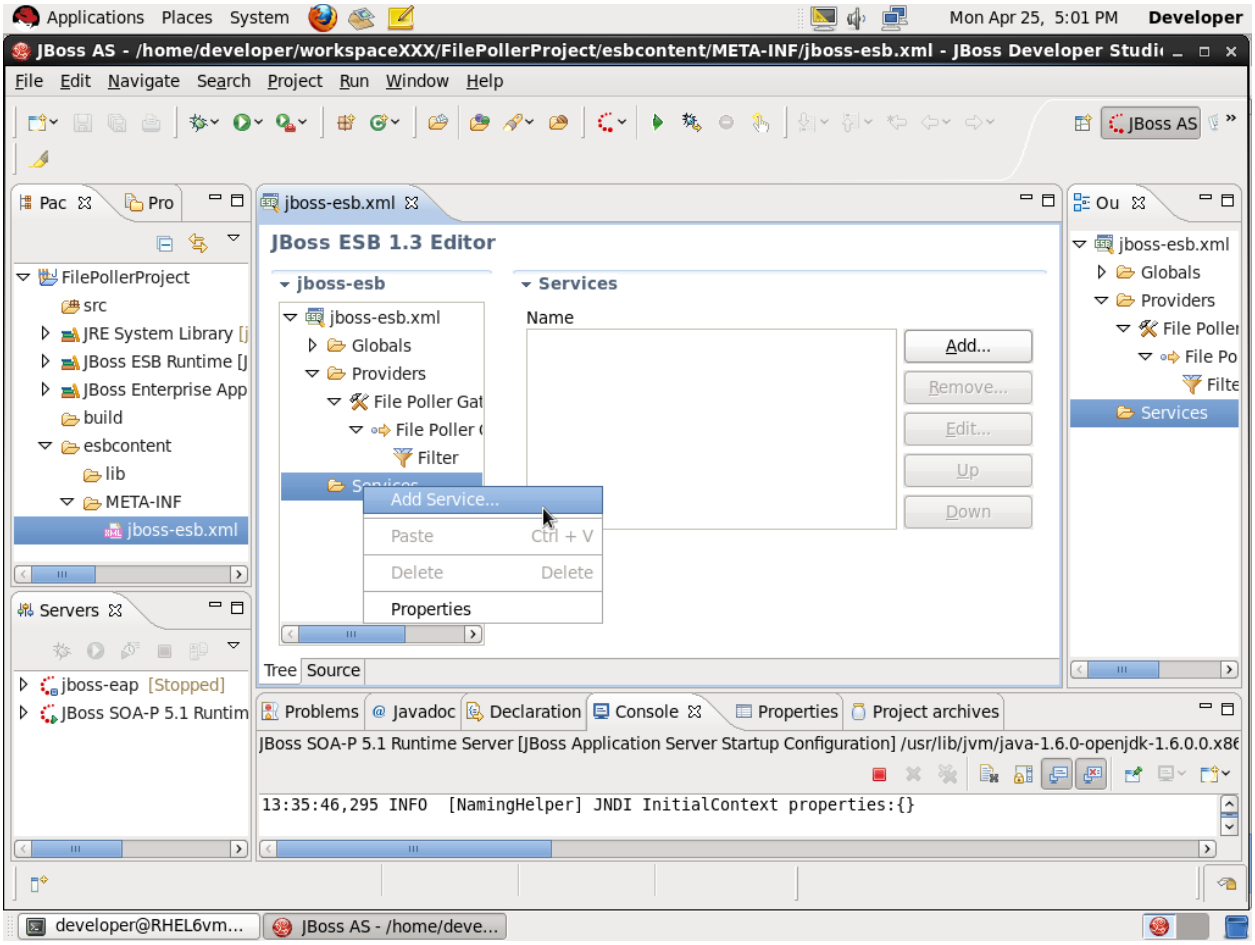


Illustration 58: Add service

Fill in the dialogue as shown below:

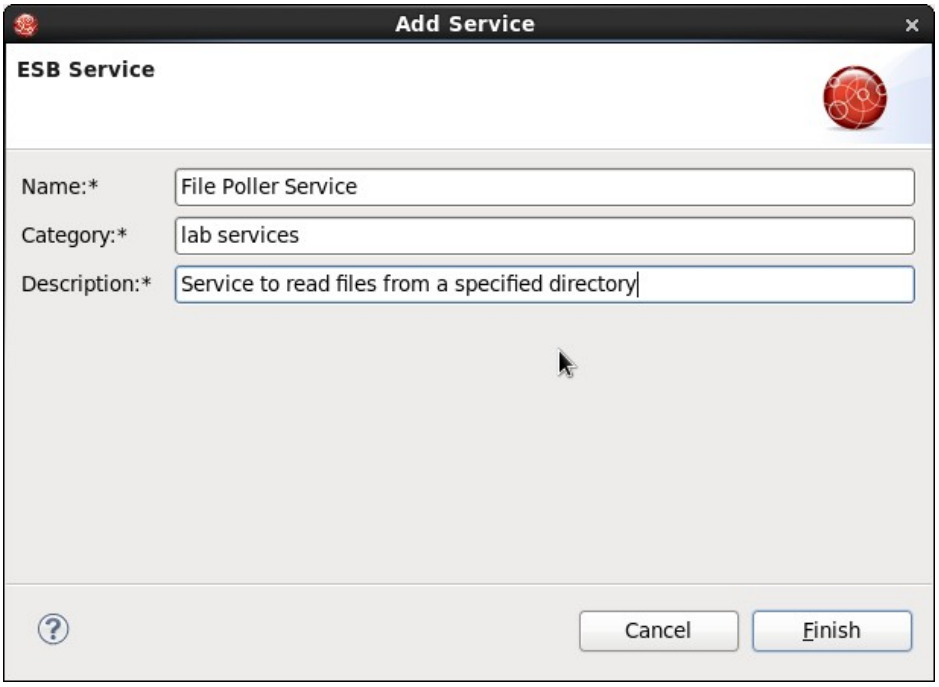


Illustration 59: Configure service

Click Finish

Lastly for the inVM scope select GLOBAL (this is simply an optimization):

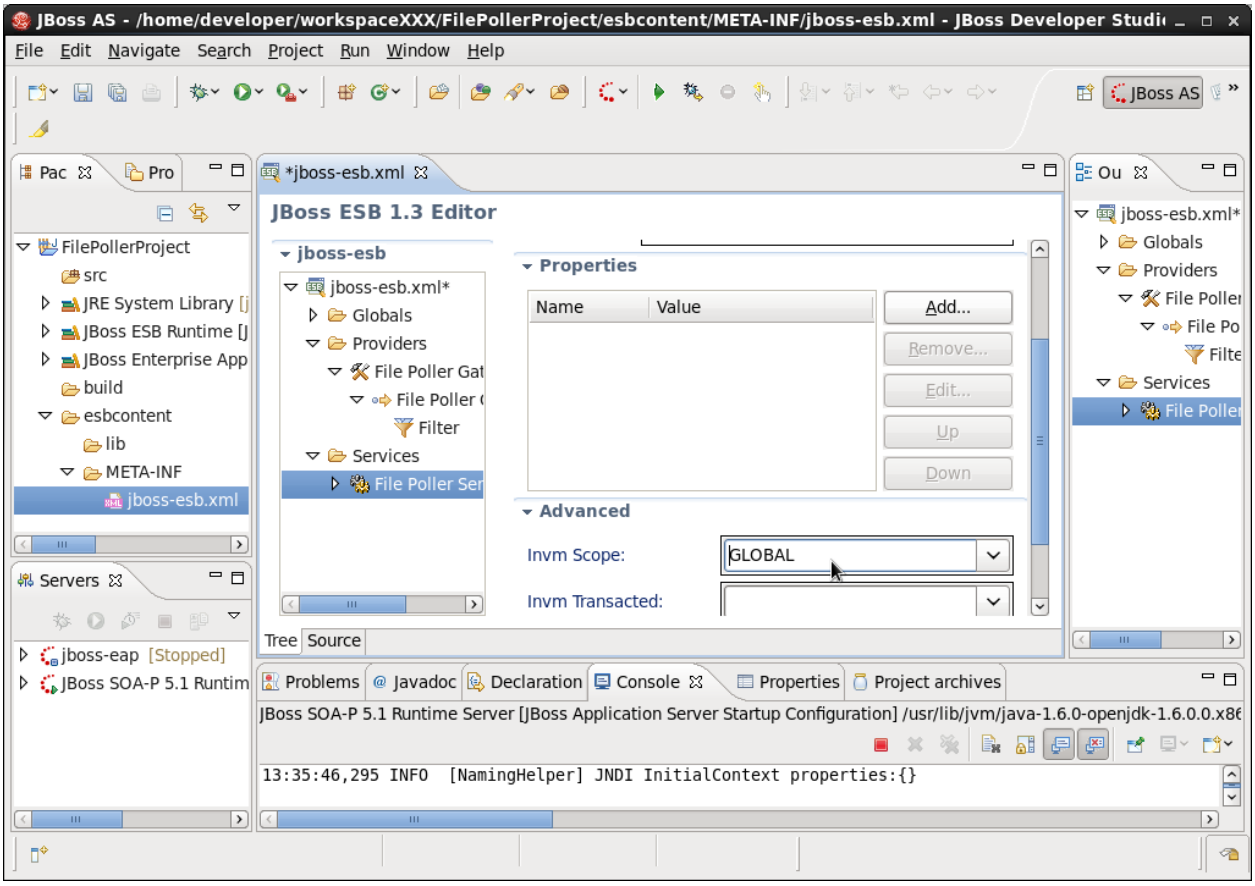


Illustration 60: Optimize service

We should see the service selected as above. The only thing we need to do here is select “GLOBAL” for the InVM scope (highlighed in the above screenshot). Basically, the InVM scope is an optimization that allows the gateway provider we created to send messages directly to the service without having to go through an additional non-gateway provider.

We now have our provider setup and we have a service created. It's now time to tell our service that we want it to receive any messages that come into the provider we created. This might initially seem an unnecessary step. But, not all services have just one provider. For example, we might have a service that accepted a SOAP document from a file system, FTP, JMS, and HTTP. In this case, we would still only have to create one service but could link it to four different gateway providers.

To link this service to the gateway provider we created, right-click on the “listeners” folder under the service we created and choose “New | FS Listener” as shown below:

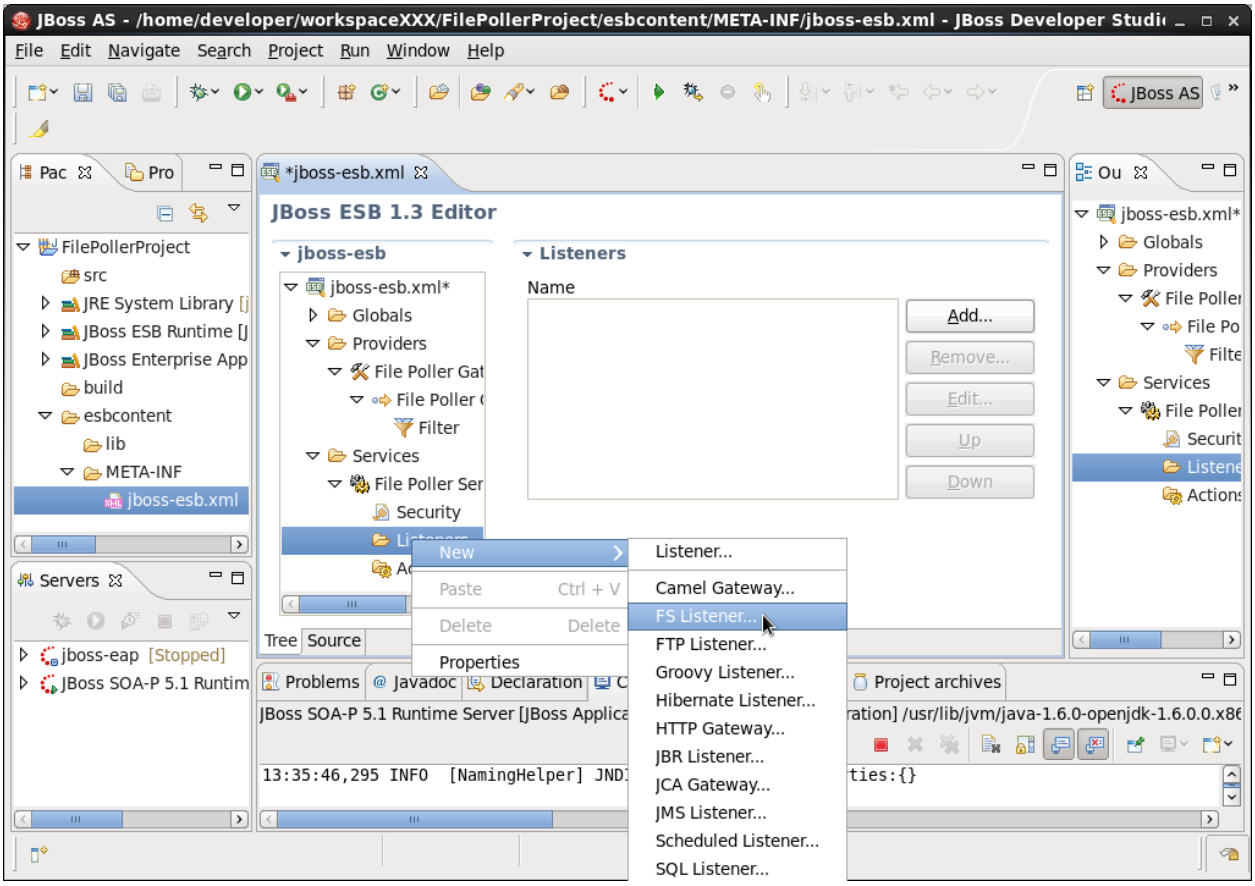


Illustration 61: Create FS listener

Type in the File Gateway Listener and select the File Poller GatewayID from the drop down:

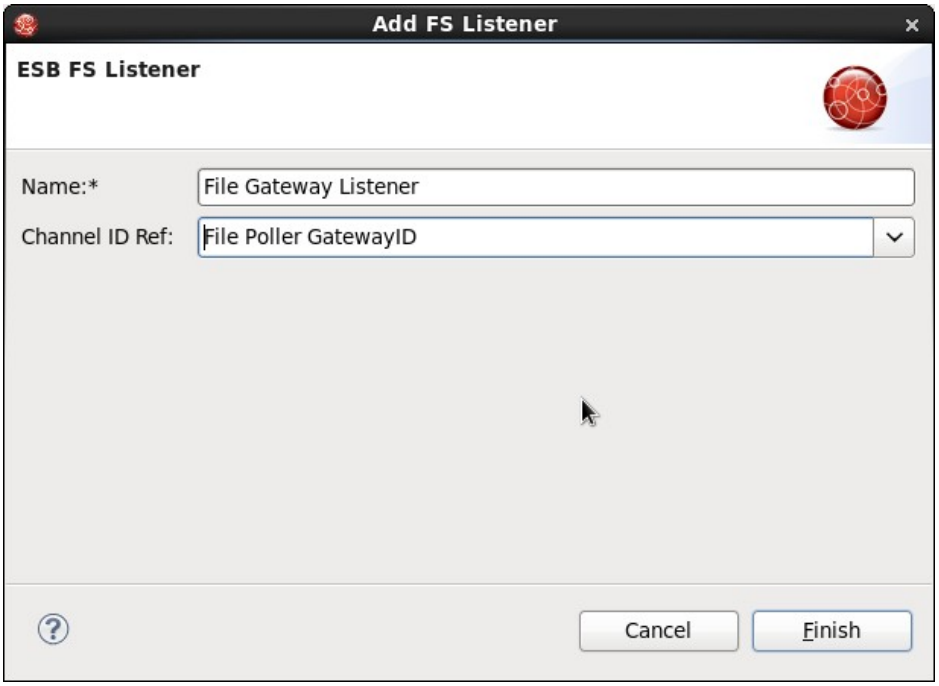


Illustration 62: Configure listener

Select Finish

Set the “Is Gateway” field to True as shown below:

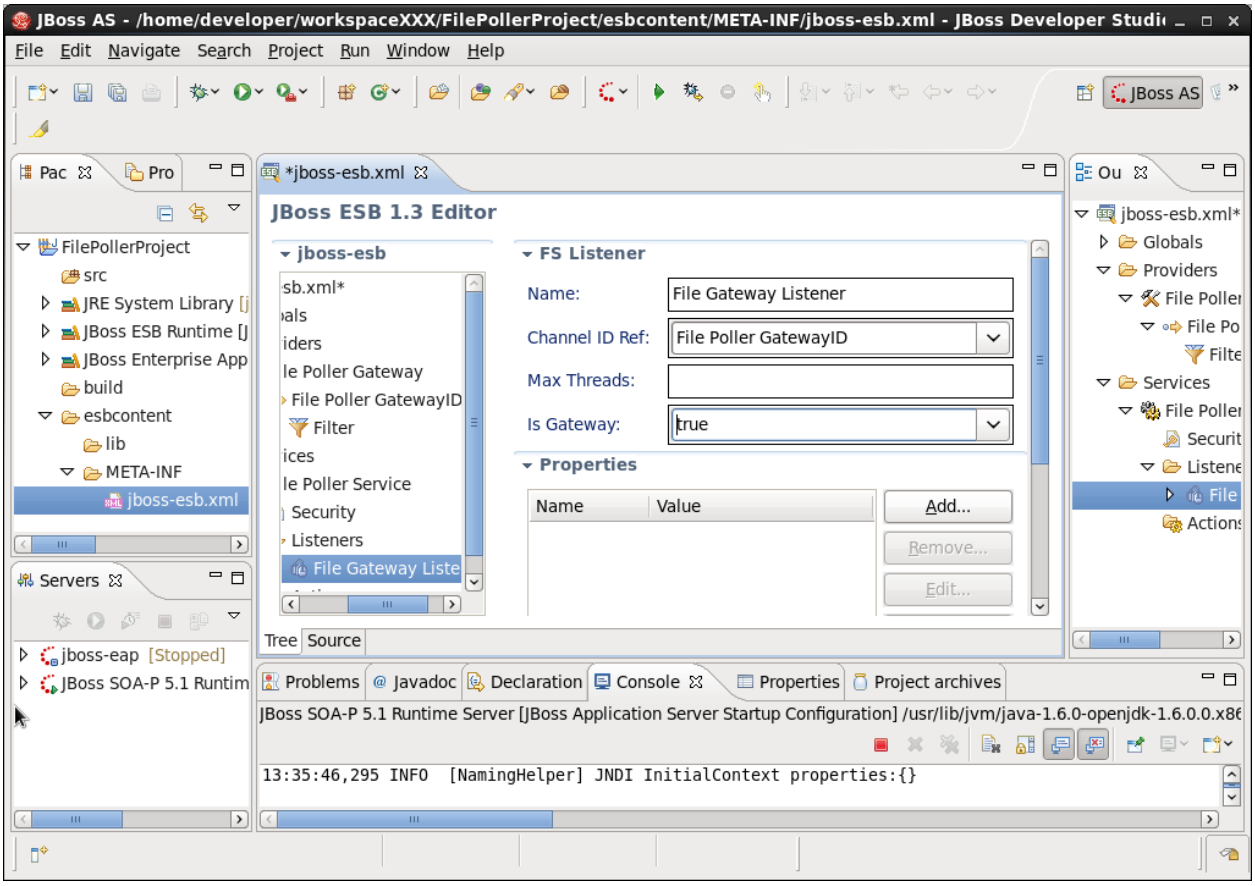


Illustration 63: Configure listener as gateway

Now we have the service pointing to our gateway provider. Make sure that you set the “Is Gateway” to “true” as highlighted above. This means that the listener is a “Gateway” and can therefore accept messages of any type (binary, ASCII, XML, etc.). If we did not specify that this was a gateway, it could only accept JBoss ESB specific messages.

Your First Action

The ESB service is actually created now and would work. But, since we don't have any actions specified on the service, we wouldn't see much when the service was invoked. So, we're going to add a “Print Ln” action to our service that will print the contents of whatever message we send to this service. To do this, right-click on the “Actions” folder under our service and select “New -> Miscellaneous -> System Println...” as shown below:

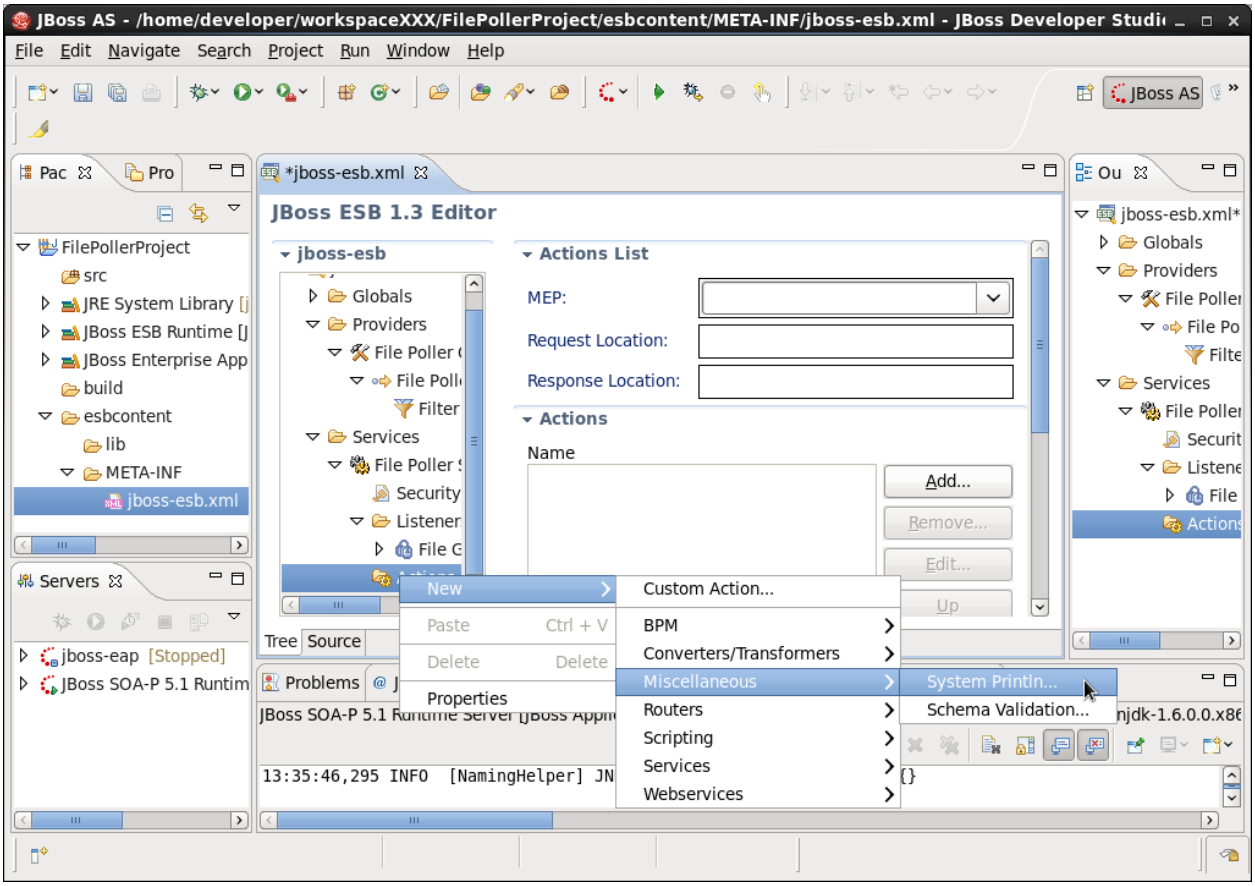


Illustration 64: Create action

Fill in the dialogue as shown below. All actions need to be uniquely named, and the message can be whatever you would like:

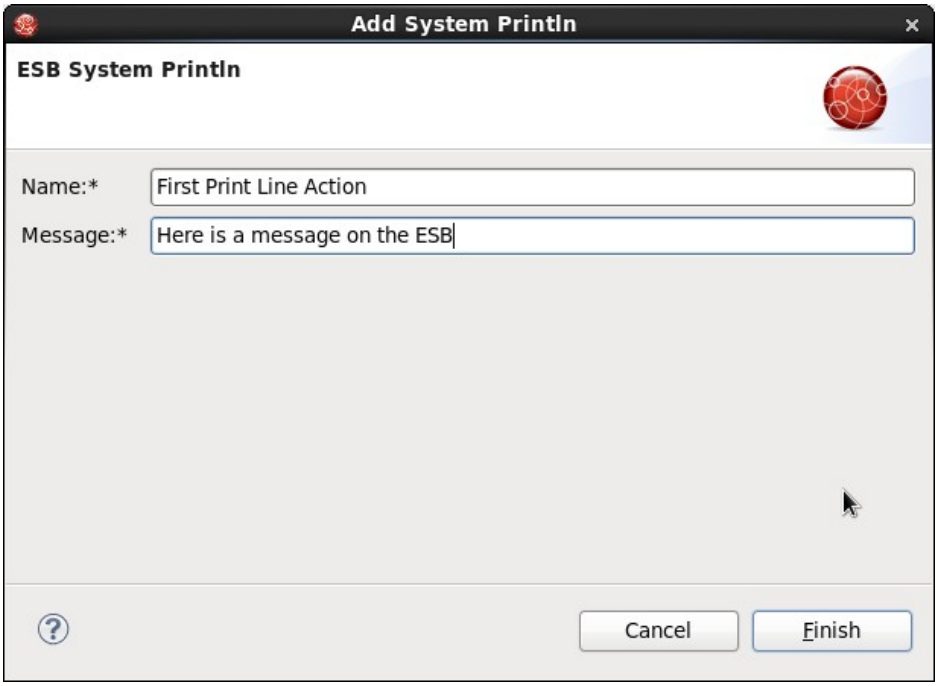


Illustration 65: Configure action

Click Finish

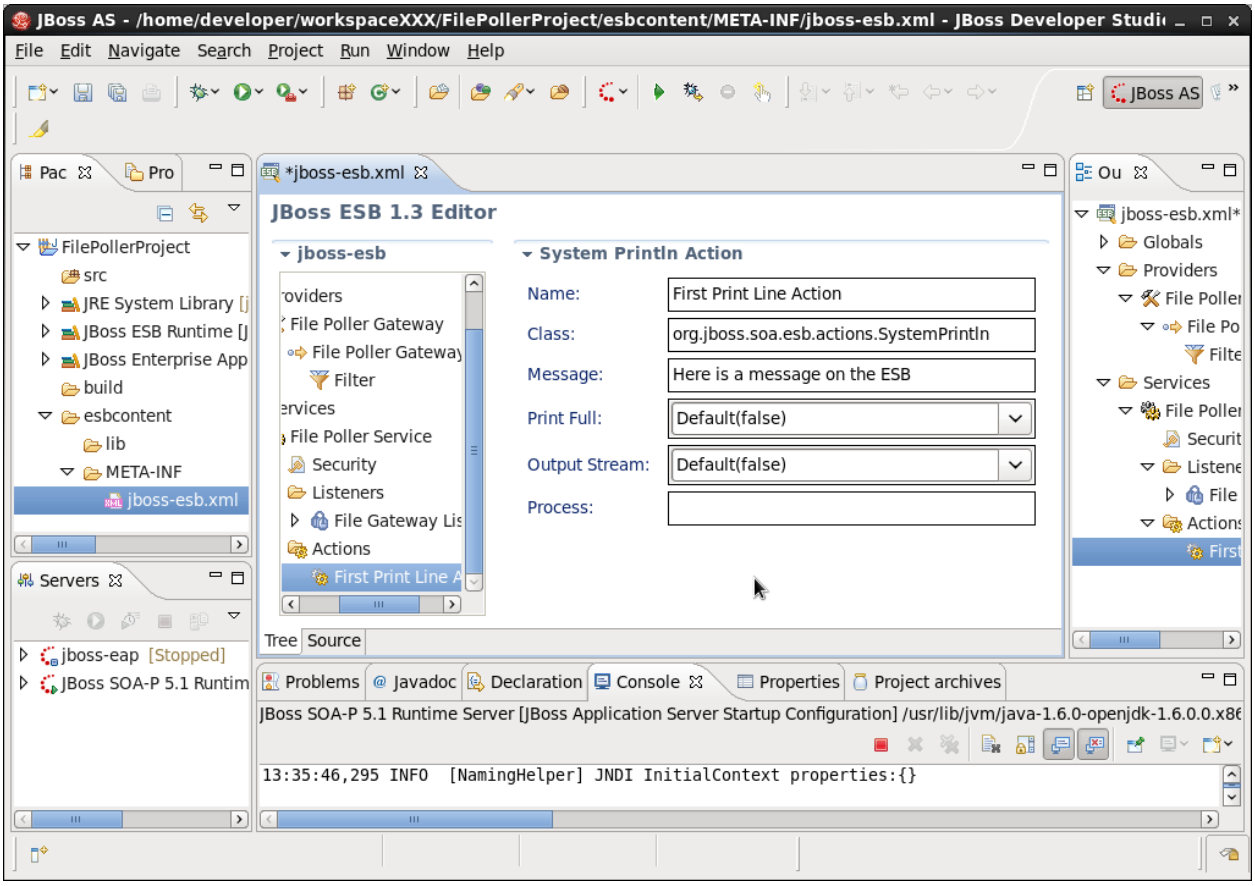


Illustration 66: Newly created action

We are all done at this point, you can save the file by clicking the Diskette Icon in the upper left corner, Click FIle -> Save, or Cntrl-S.

Deploy the Project

Now it is time to deploy our project to our embedded SOA-P Instance. The directory that we wish to scan must first exist. In a separate terminal window, type the commands:

```
cd ${USER_HOME}
mkdir tmp
```

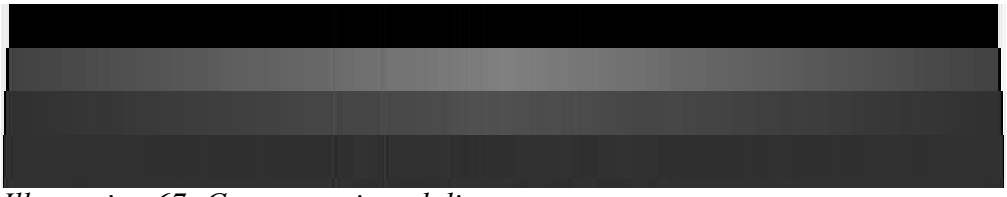


Illustration 67: Create monitored directory

In JBDS, right-click on the JBoss SOA-P 5.1 Runtime Server as shown below:

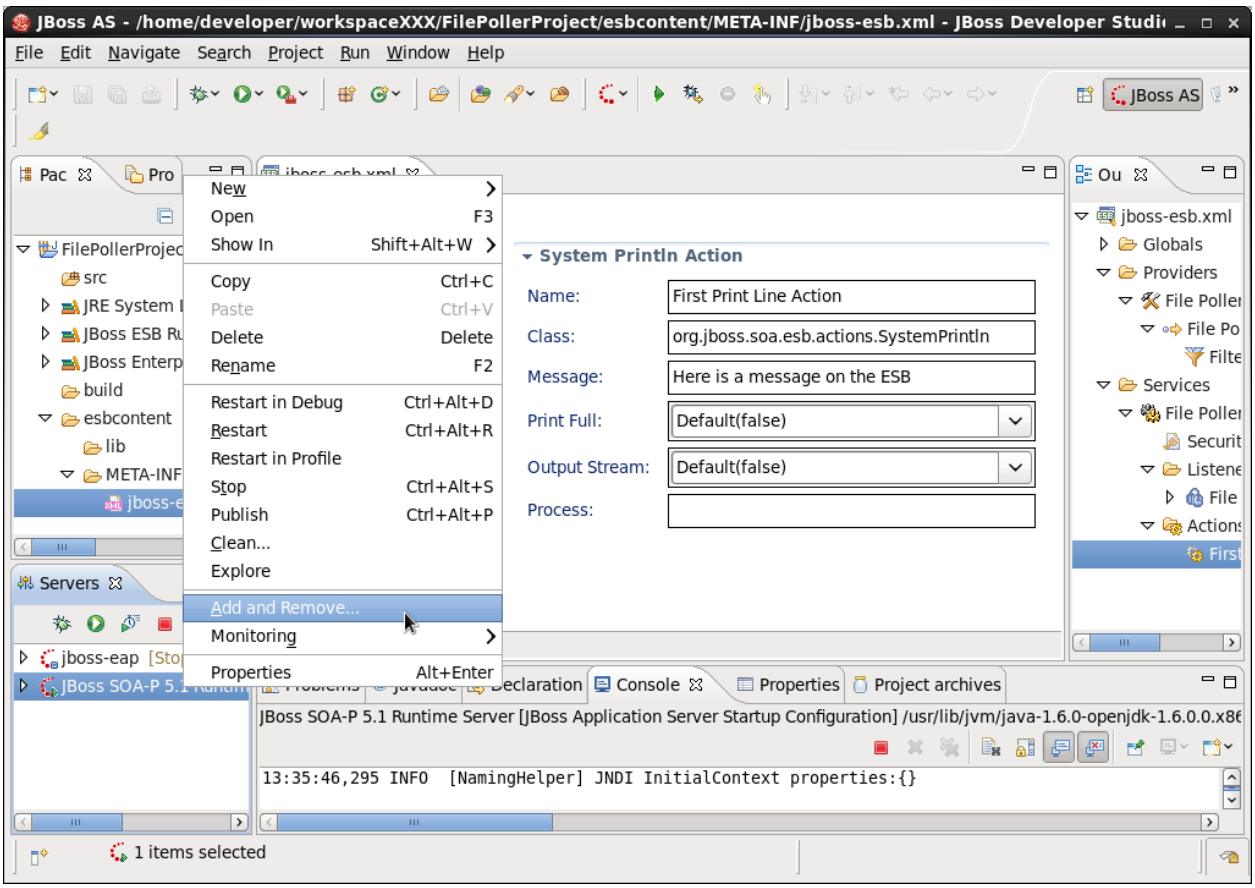


Illustration 68: Prepare to deploy project

Now select the Project on the left hand side and click “Add” as shown, moving it to the right hand side:

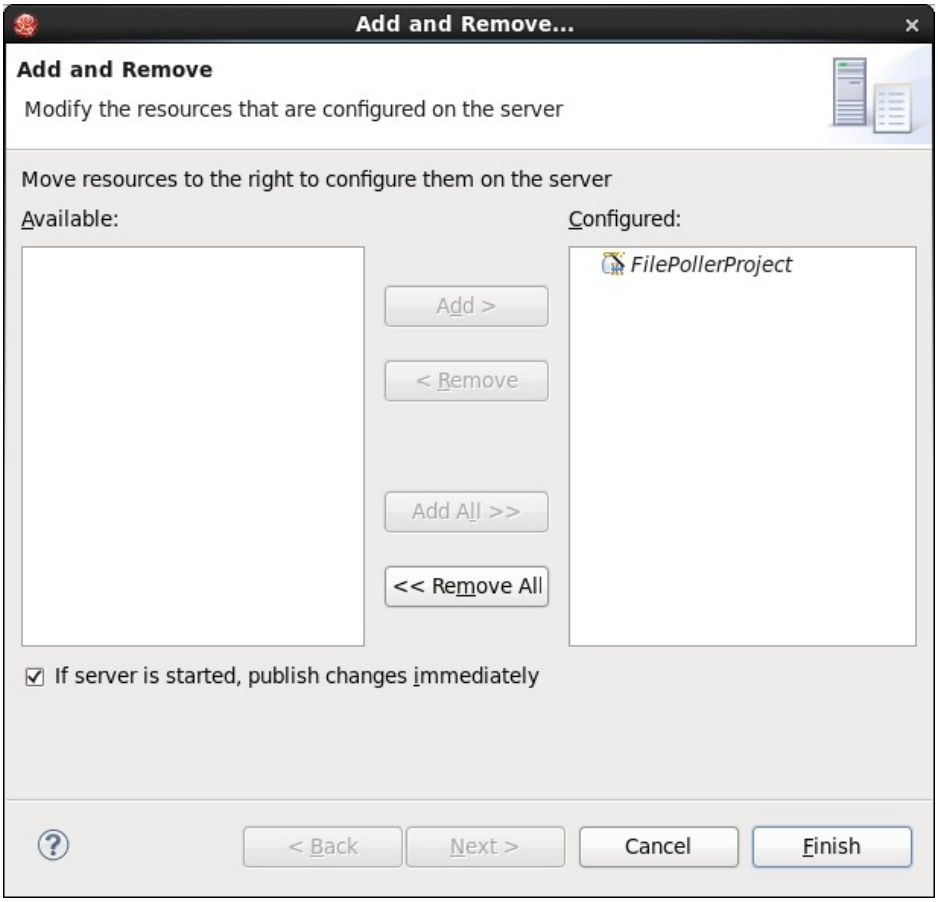


Illustration 69: Deploy the project

Click Finish

You should notice that in the console the FilePollerProject.esb has been deployed as shown:

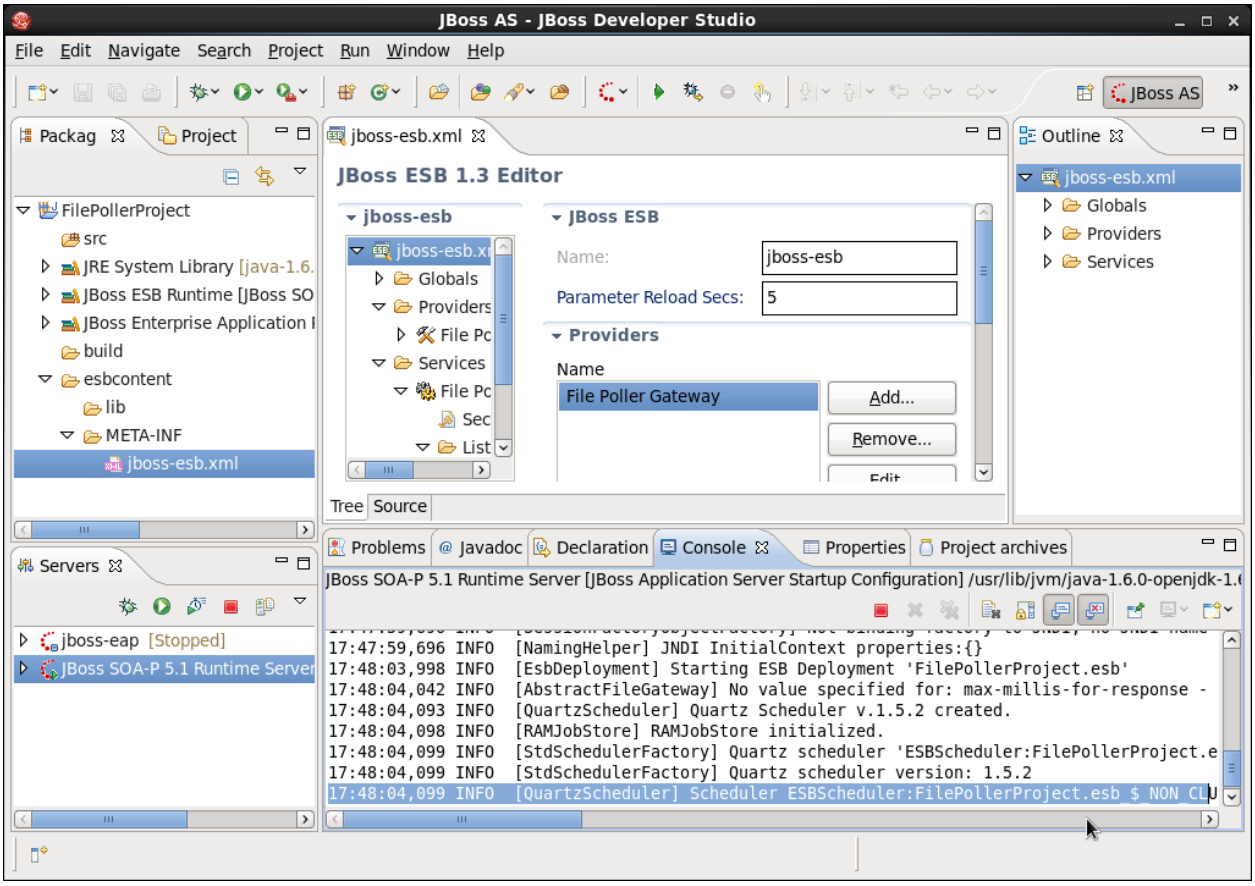


Illustration 70: Log showing ESB deployment

You'll see in the console that when we added our project to the server, JBDS created a ".esb" archive (simliar to a .war archive) and deployed it to our SOA-P server. Now it's time to test it!

Test the Service

So, we need to create a file with the suffix we specified in the directory we specified. You can do this with any editor like vi, Notepad, emacs, etc. I'm going to just “echo” some text to a file below like this:

Illustration 71: Create test data

Then notice your output below:

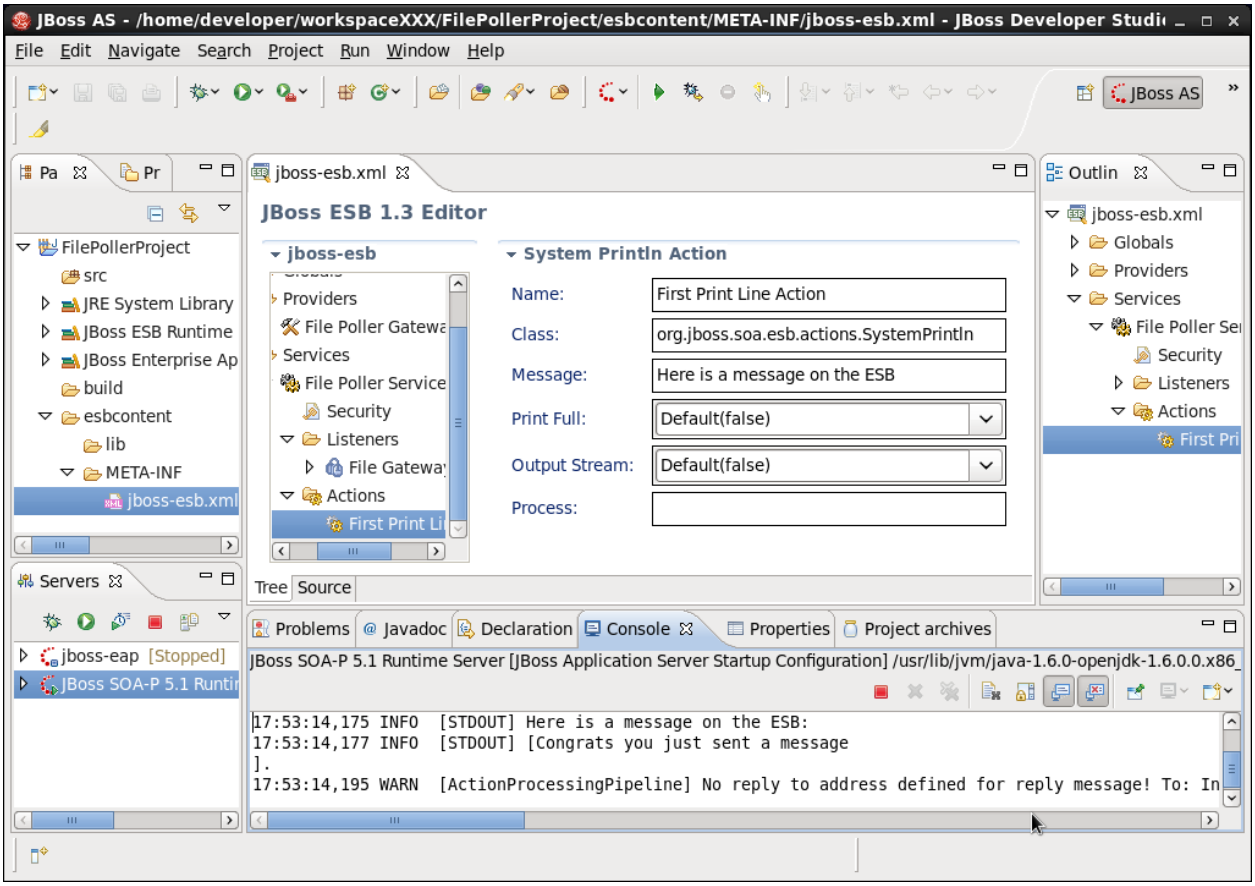


Illustration 72: Test the ESB service

Congratulations you have now successfully submitted a file and used a file based poller to ingest a message into the SOA-P. If you do not see the message as above, or are having other issues, please raise your hand. This lab is now completed.

Lab Number 6: Adding a Custom ESB Action

Your First Custom Action

ESB services have listeners and actions. Thus far, we have created a service with a file poller listener and a single System Println action. Now, we're going to add a custom Java action. An ESB action has full access to the ESB message. It can read the message, modify the message, forward to other services, call out to external systems, anything that can be done from a Java class.

To create a custom Java action, we first need to create the Java class for it. Open up the jboss-esb.xml file and right click on Actions and highlight New -> Custom Action as shown:

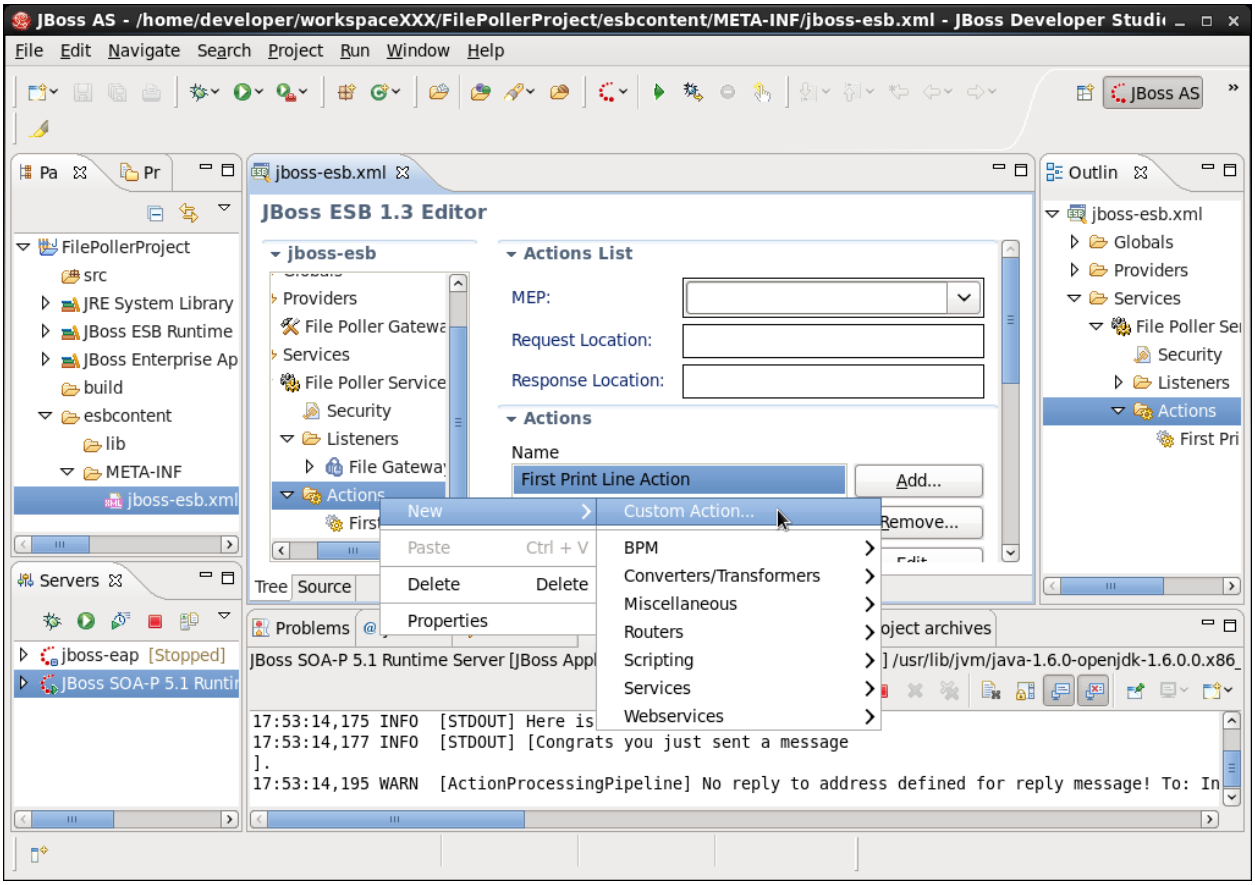


Illustration 73: Create custom action

Fill in the dialogue as shown:

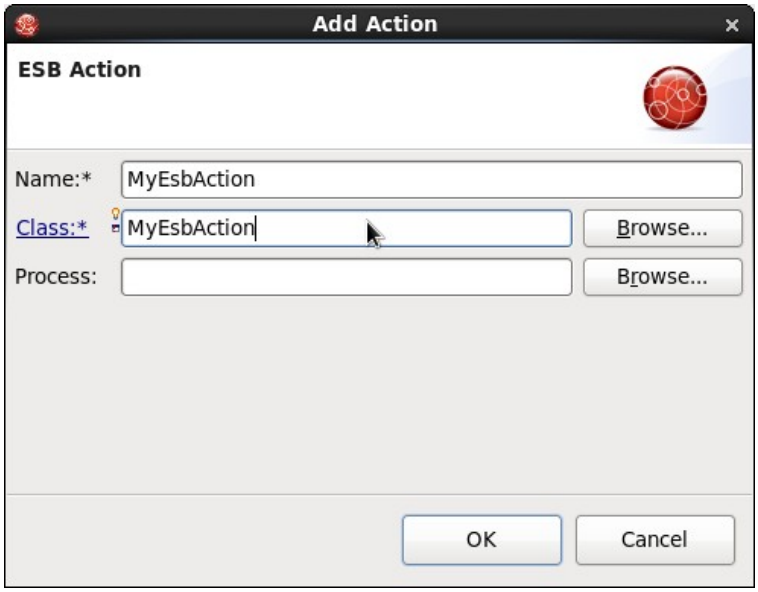


Illustration 74: Configure custom action

Click OK

This will bring you back to the jboss-esb.xml file editor as shown:

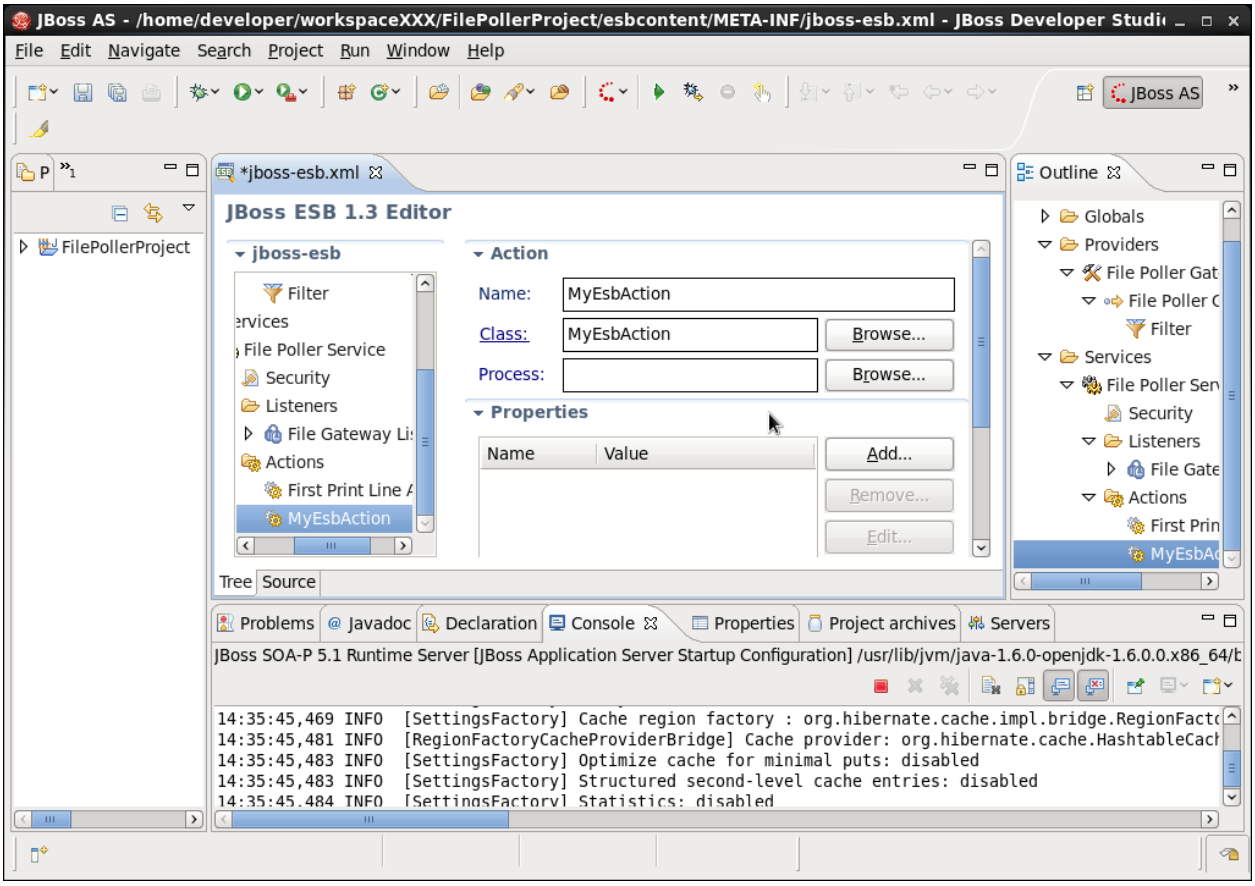


Illustration 75: Newly created action

Double-click on the underlined Class to the left of the MyEsbAction. This will bring up the new java class wizard filled in as shown. Update the package to com.jboss.lab:

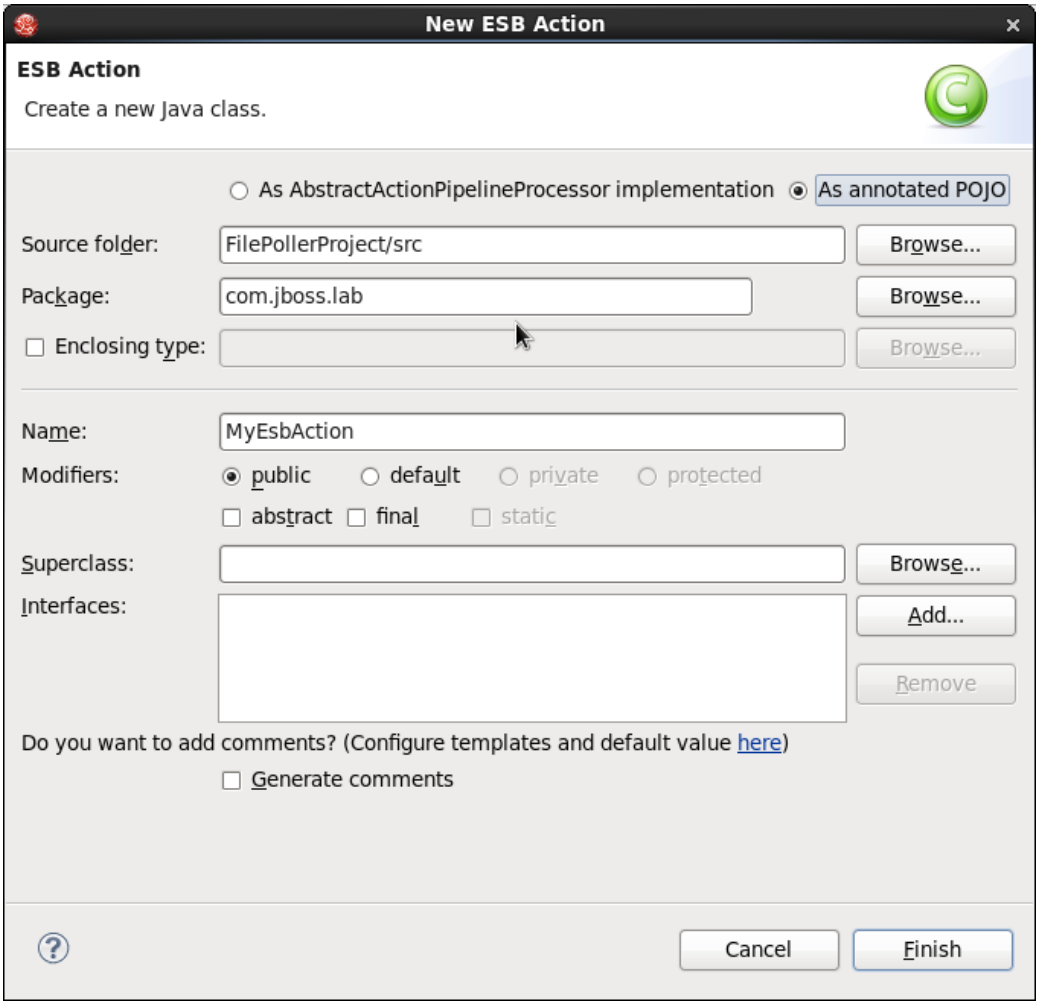


Illustration 76: Modify new action

Add Custom Code

Click Finish. Expand the src/com.jboss.lab package under the FilePollerProject and open the newly created java file:

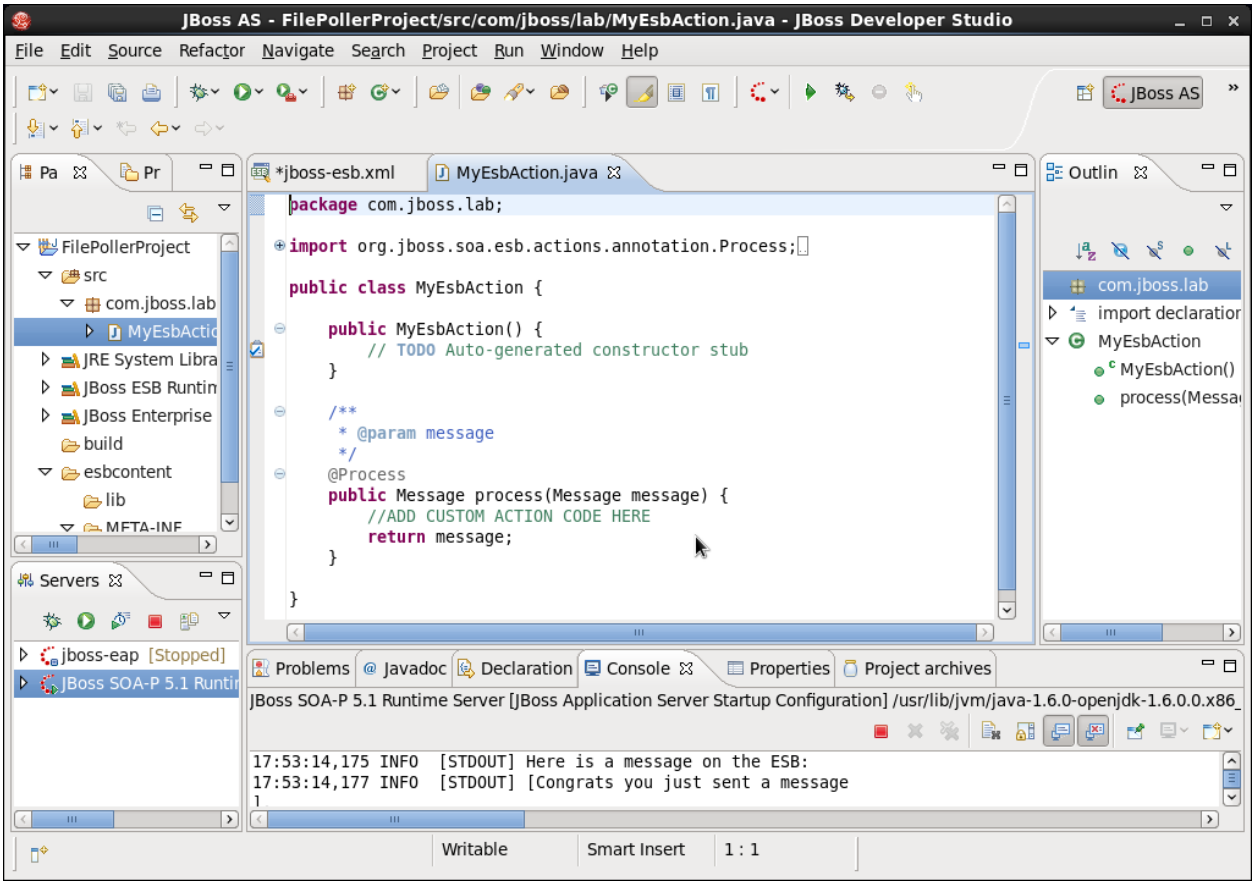


Illustration 77: Auto-generated source code

As you can see this has created an annotated POJO class, with stubbed methods for the constructor and process method. Notice the process method was created by default, and the "@Process" annotation indicates that this is the method SOA-P will invoke in an Action. In the process method add in the following code like this:

```
String messageString = new String((byte[])message.getBody().get());

System.out.println("Default message body is: " + messageString);

message.getBody().add("****" + messageString + "****");

return message;
```

So your file ends up looking like this:

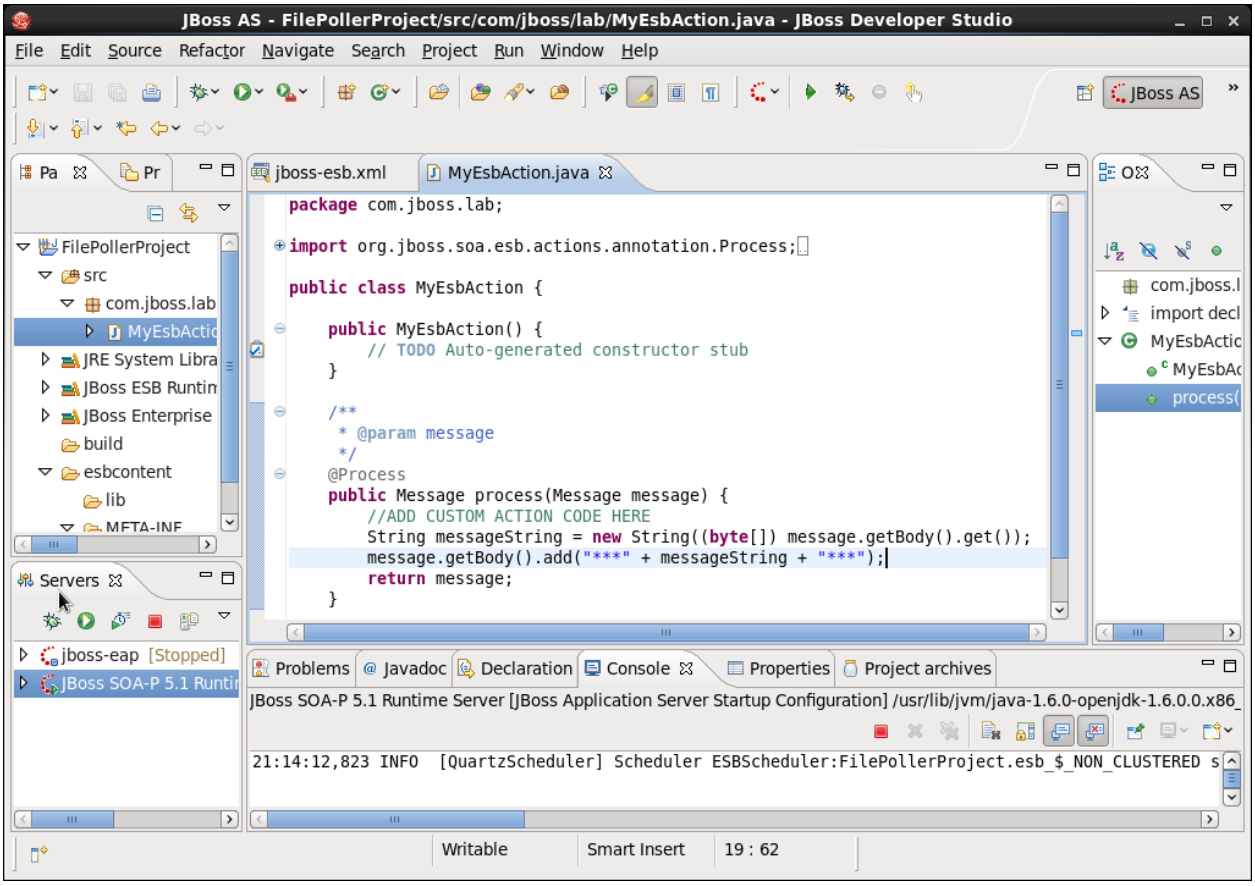


Illustration 78: Edited source file

Note that this action is printing out the current contents of the message and then modifying the message. We are casting the default message body to a byte[] and storing in a String so that it will print correctly. Without that, it would just print the address of the byte array (default toString() implementation).

By default, our new action will be added below the Println Action. So, click on the new custom action and drag it above the Println Action so that it will be executed before the Println Action. In this way, we should be able to see if our custom action really modified the ESB message, as shown below:

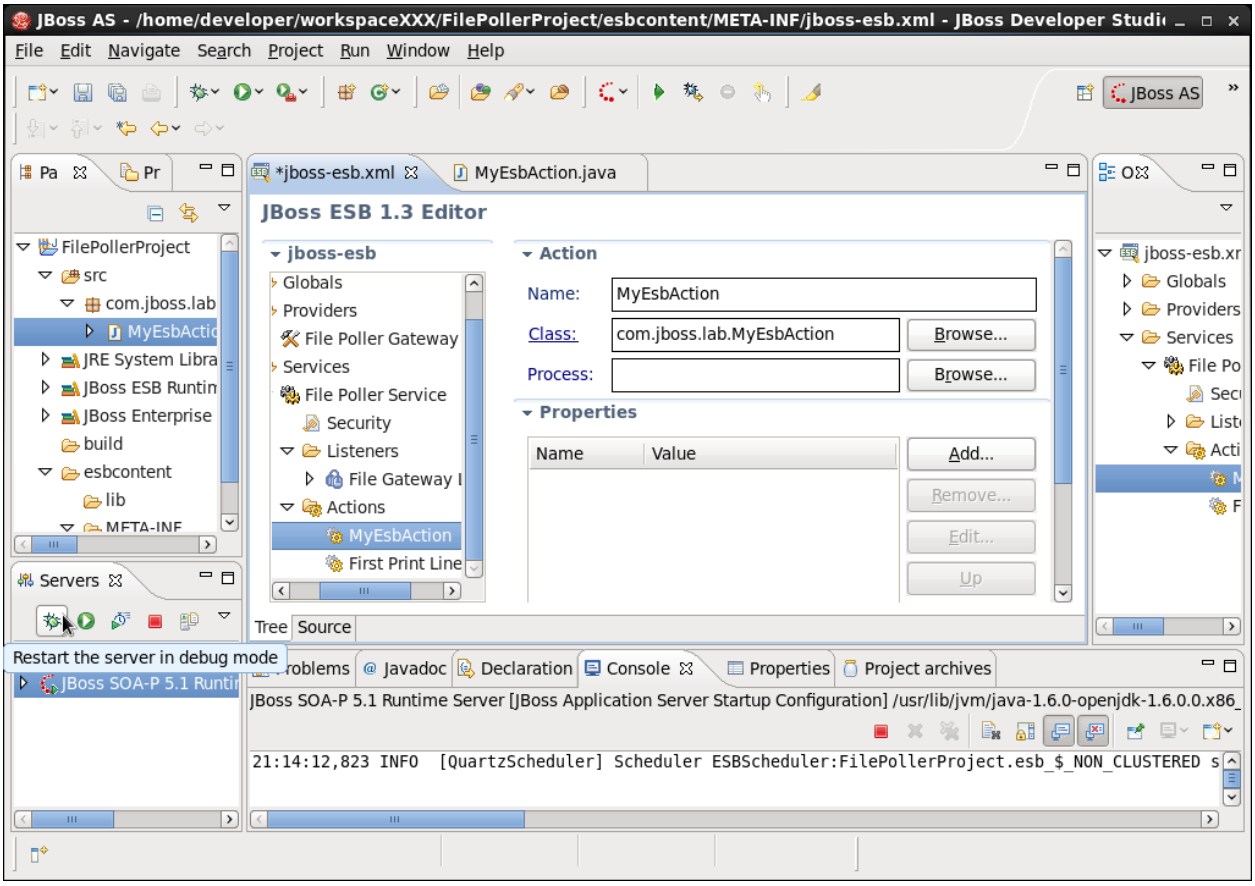


Illustration 79: Change action order

Publish Your Changes

Now we just need to save the project “File | Save All”. You will see that the FilePollerProject.esb is automatically re-deployed as shown:

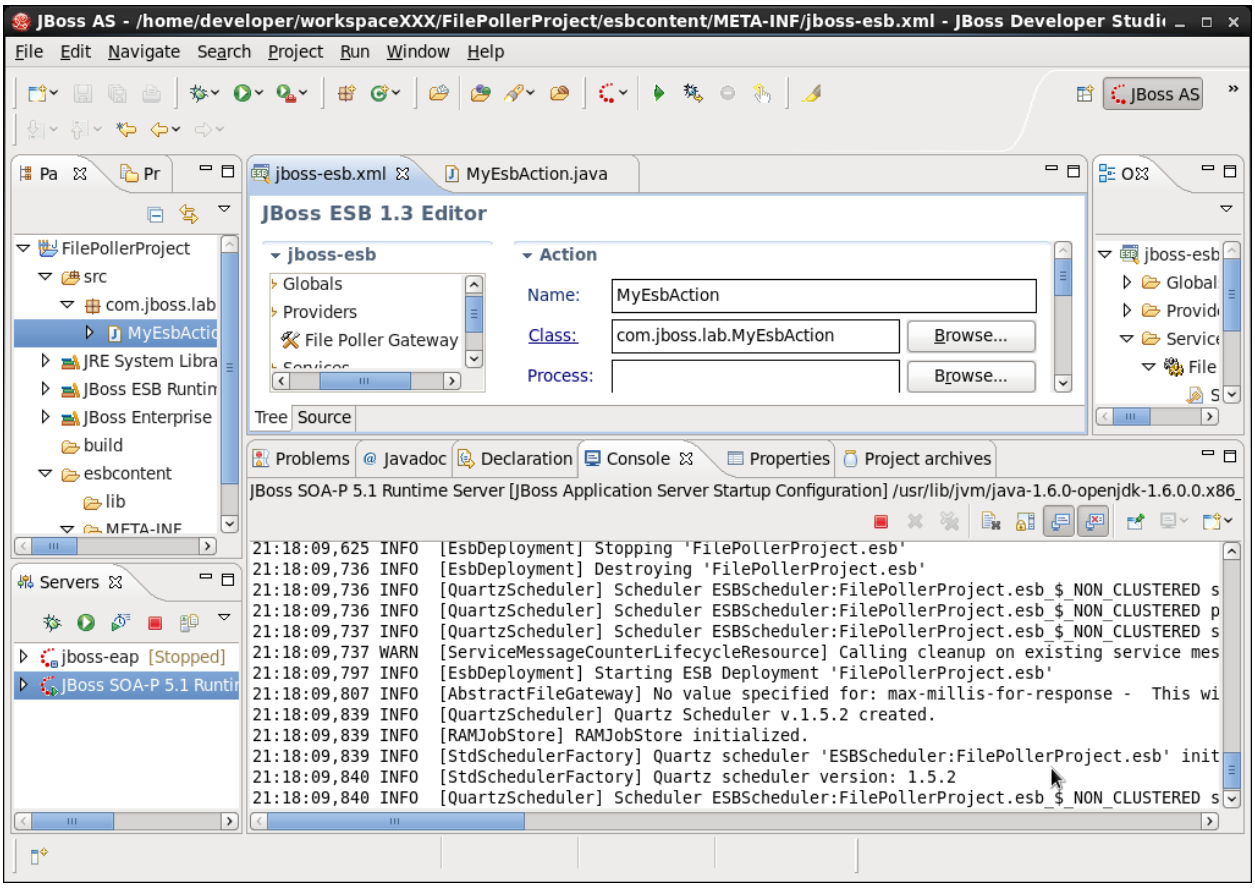


Illustration 80: Redeploy application

Echo the message again in the window as before:

Illustration 81: Send another message

See that the message was altered before the Println Action was called:

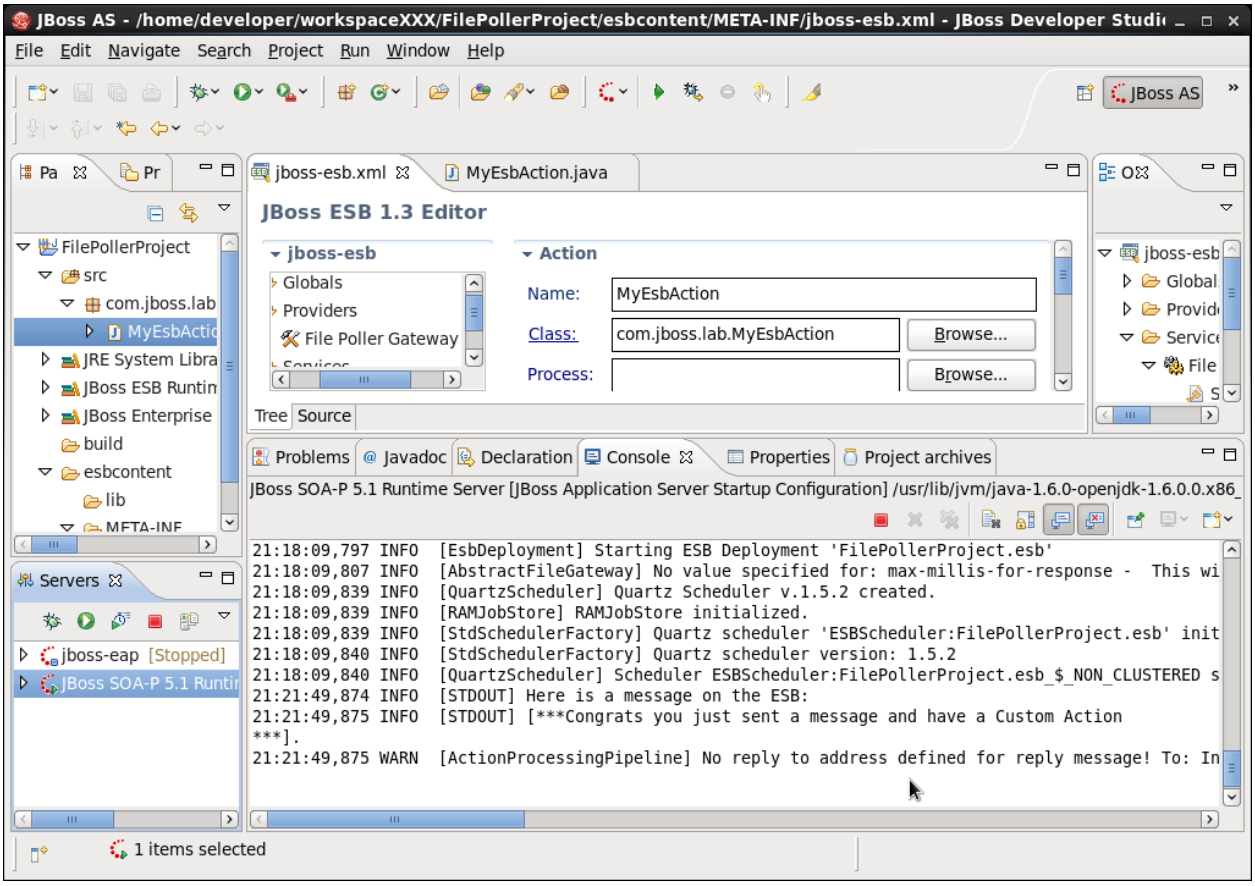


Illustration 82: Confirm that message was modified

Sure enough, we see the "Default message body is" message from our custom action. Then, we see that our second action (System Println) printed out a message that is surrounded by "***" - which was done by our custom action. Congratulations, you've now created a custom action and added it to your ESB service! You could modify this action to do anything that Java can do – which gives you a lot of flexibility. That is a very important concept: you can modify this Action to do `ANYTHING` that Java can do.

You have now successfully completed this lab.

Lab #7: Installing soapUI for WS Testing

Install soapUI

SoapUI is a wonderful tool for working with web services. In the simplest case, it provides a GUI to parse a WSDL, generate an editable test client, and invoke the web service. In more complex cases, SoapUI can be used to create elaborate and mult-threaded load test scenarios. For this lab we will only be using it for the most simple case.

First, we need to install SoapUI. Please find the zip file for your platform (Linux, Mac, or Windows) and extract it as shown below. It will be located in the ${USER_HOME}/Downloads/soapUI Directory and unzip it as shown:

Illustration 83: Unzip soapUI

Start soapUI

We should be able to start SoapUI by running the start script as shown below. On non Windows platforms, you made need to change the permissions on the start script so that it is executable as shown:

Illustration 84: Launch soapUI

That should bring up the soapUI graphical user interface as shown below:

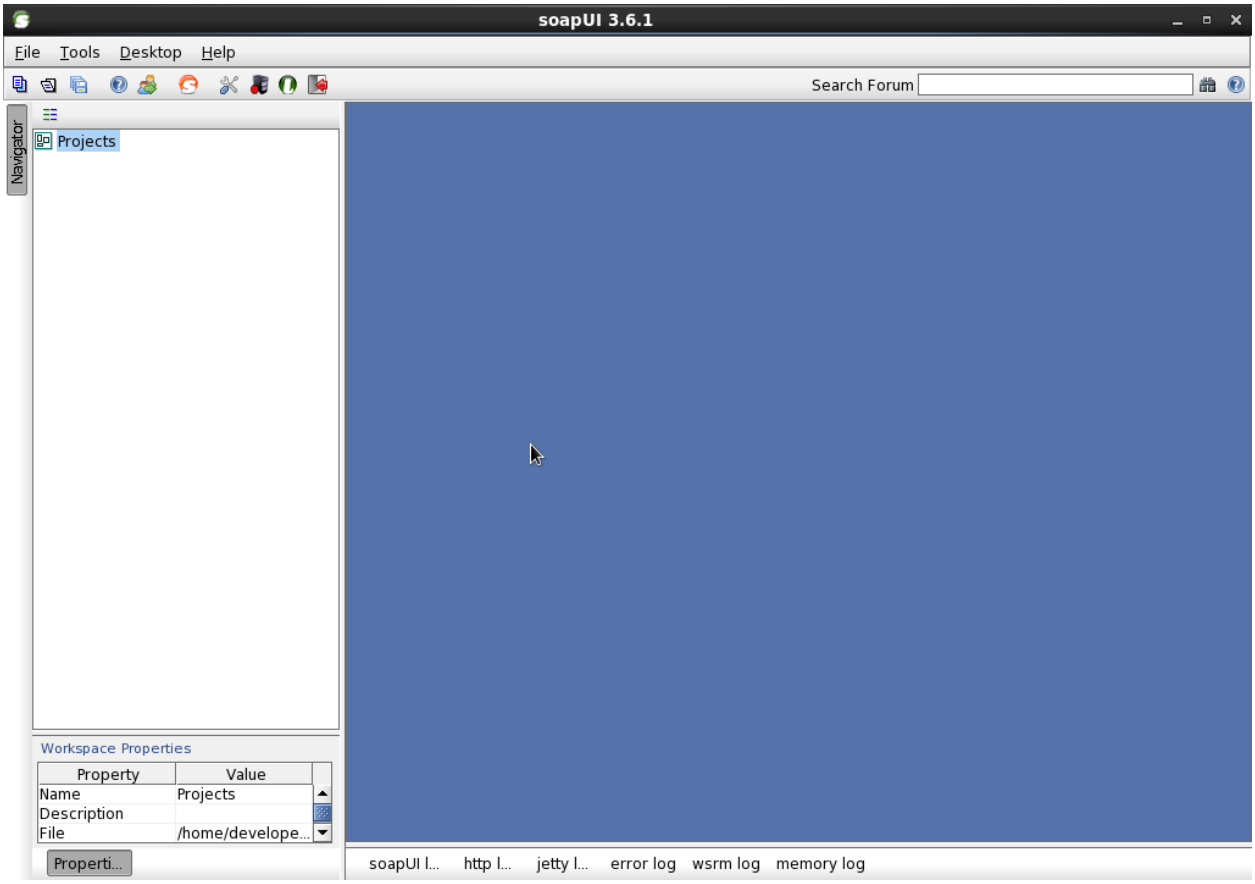


Illustration 85: soapUI interface

At this point you have completed this lab, if you do not have soapUI installed please raise your hand. Feel free to close the window by clicking the “X” in the upper right corner of the browser.

Lab Number 8: Create a Simple JSR 181 Web Service

One of the most common use cases for an ESB is to mediate web services. The client sends a request to the ESB, the ESB may do specialized security, transformations, routing, or a host of other actions, and then the ESB routes to a back-end web service implementation.

In this lab, we will create a simple JSR 181 web service and show invoking it from SoapUI. In the next lab, we'll proxy this web service with the ESB and show invoking the proxy via SoapUI.

Creating a JSR 181 web service is quite easy with JBDS. First, we'll create a new Dynamic Web Project "File -> New -> Dynamic Web Project".

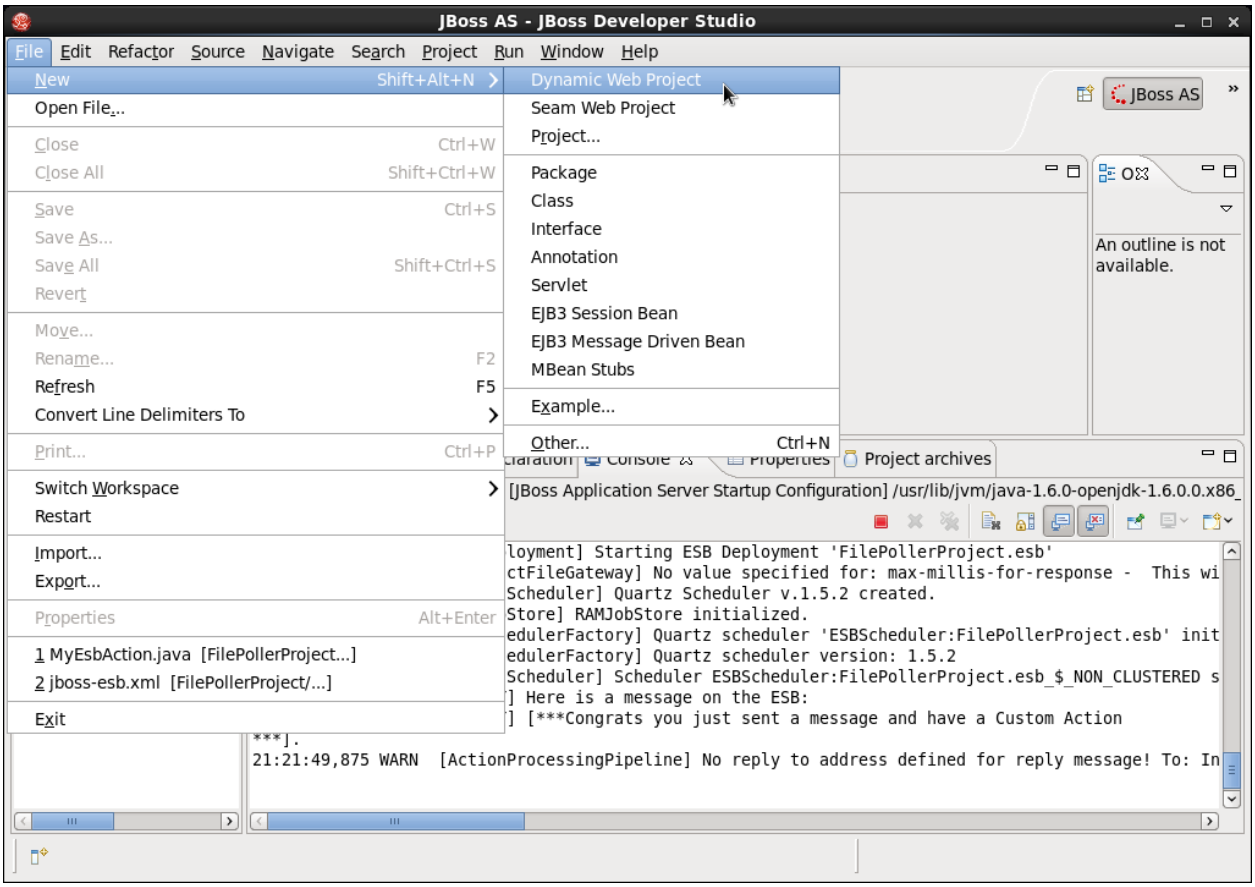


Illustration 86: Create new dynamic web project

Create JSR-181 Annotated Class via Wizard

Name the project MyWebServiceProject and make sure the target runtime is JBoss SOA-P 5.1 Runtime and the configuration is "Default Configuration for JBoss SOA-P 5.1 Runtime" as shown:

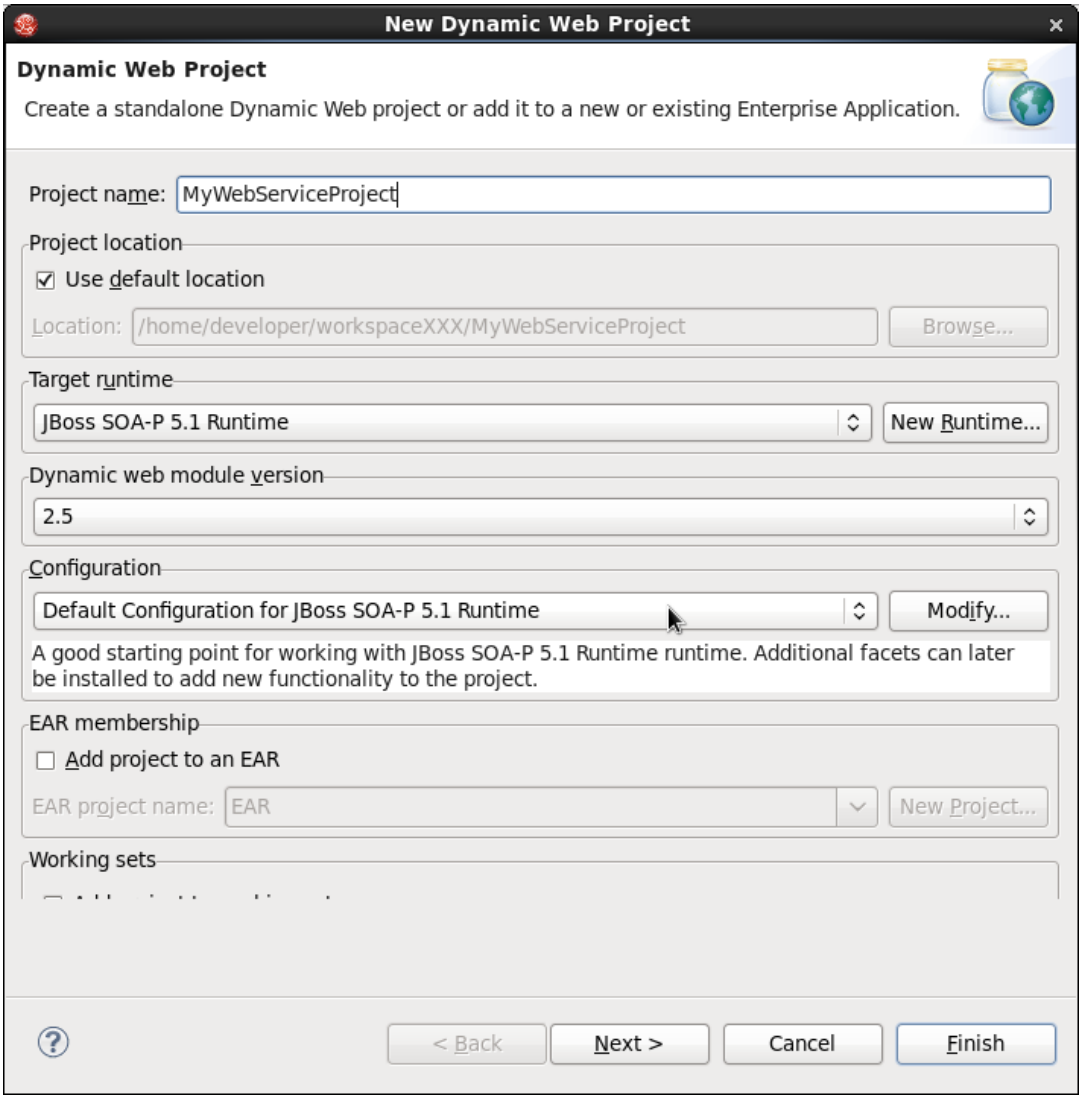


Illustration 87: Configure new project

Click Finish. You can switch to the Java EE perspective if desired, but I prefer to stay in the JBoss AS perspective. So, click "No":

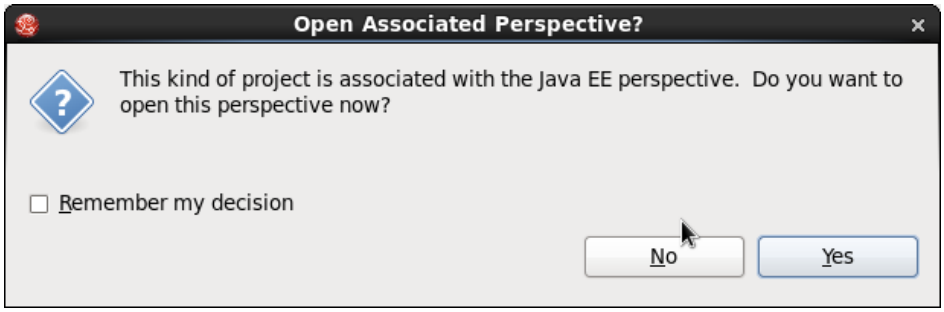


Illustration 88: Decline perspective change

That brings you to a screen that looks about like this once you minimize and maximize the two projects as shown:

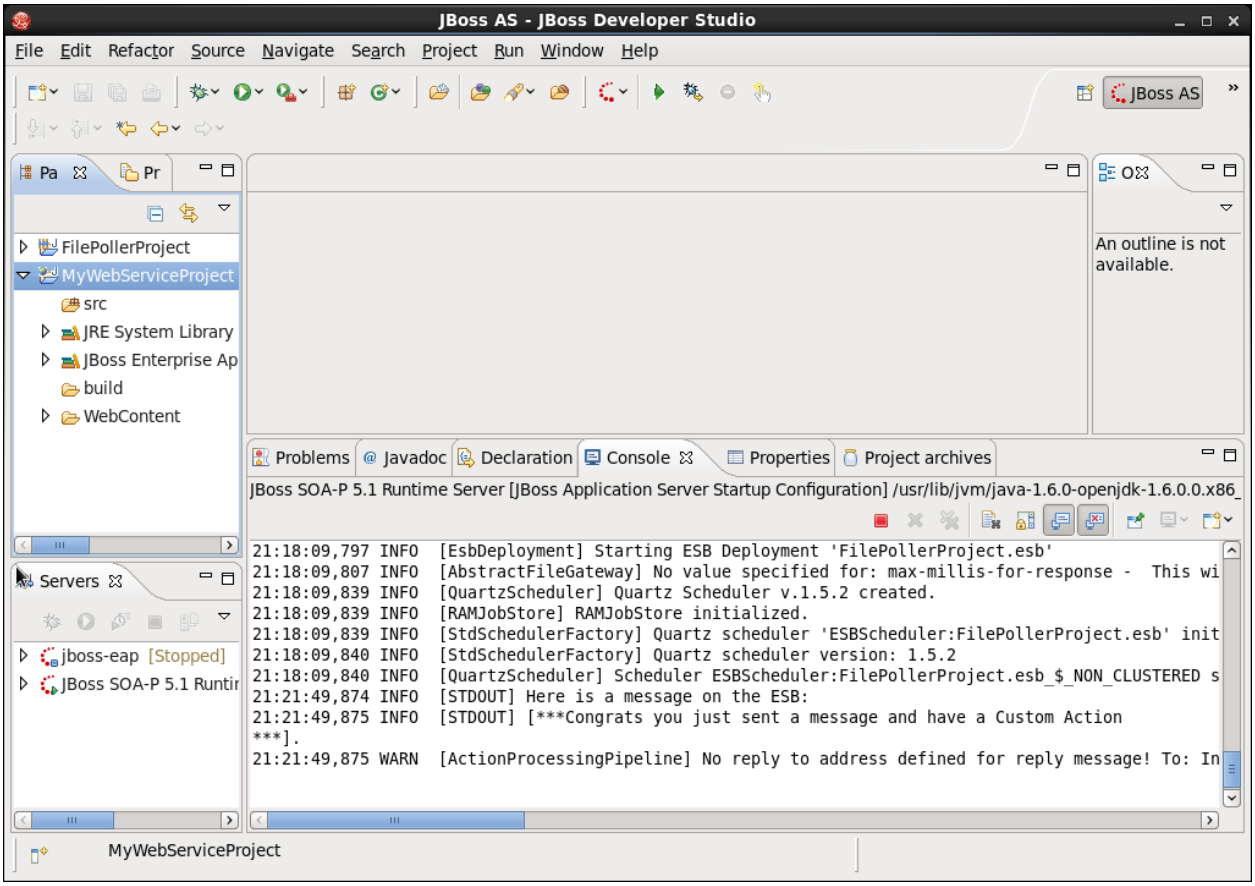


Illustration 89: Newly created project

Next we will have to create a JSR-181 web service. Thankfully JBDS provides a wizard that sets up all of this for us, select New -> Other as shown:

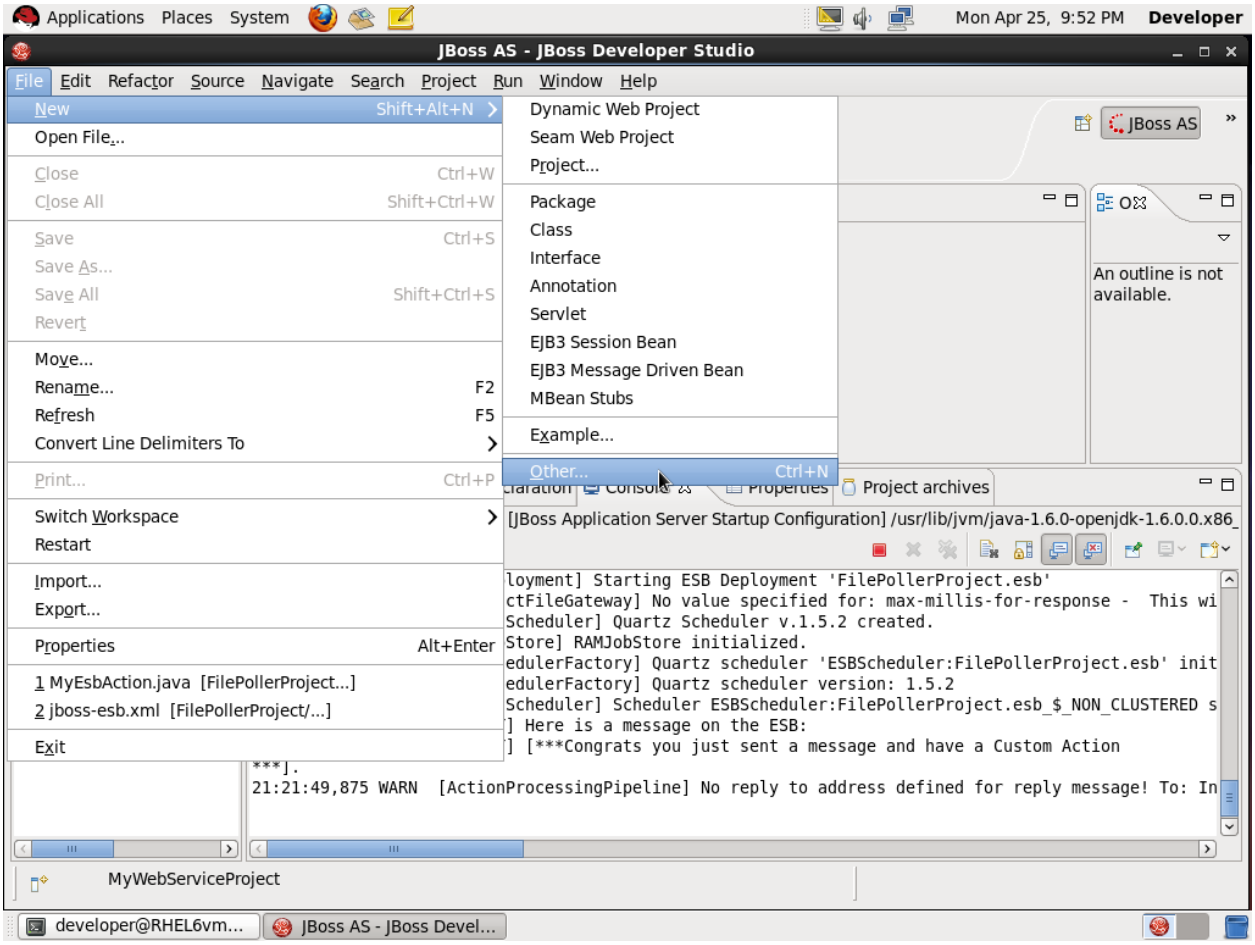


Illustration 90: Launch wizard for web service

This will bring up a dialogue where you can type in Create in the top search box and this will bring up a Wizard to fill in the attributes for JSR-181 annotated web service, as shown:

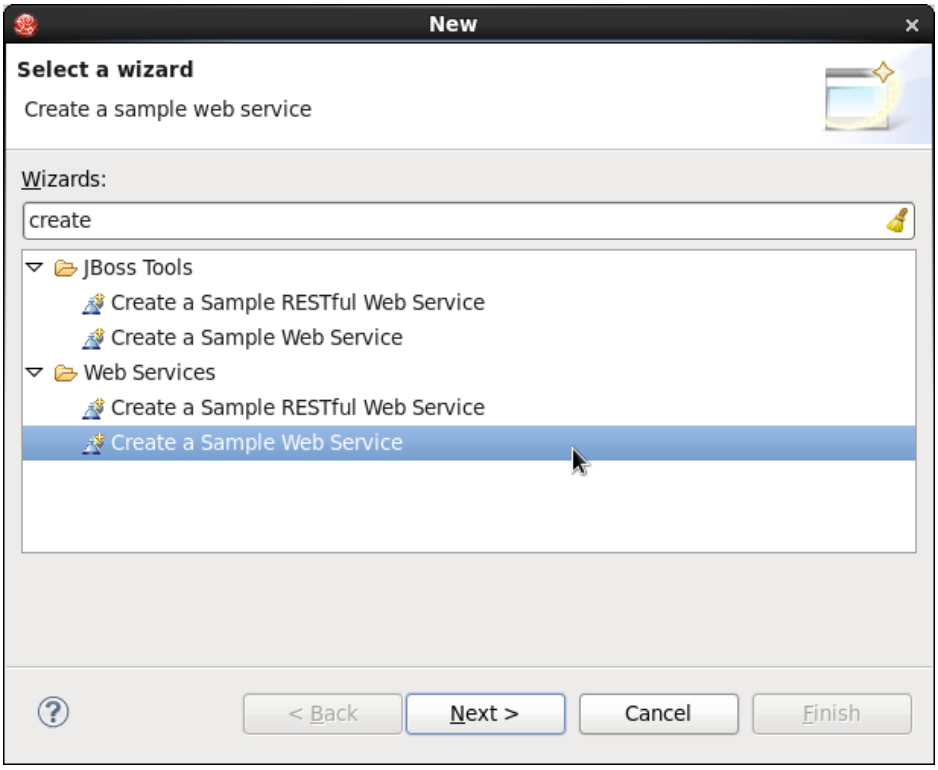


Illustration 91: Create a sample web service

Select Next. Name the web service and Class “MyFirstWebService” as shown. This essentially is the servlet or end-point name that we will use to communicate to the web service, as shown:

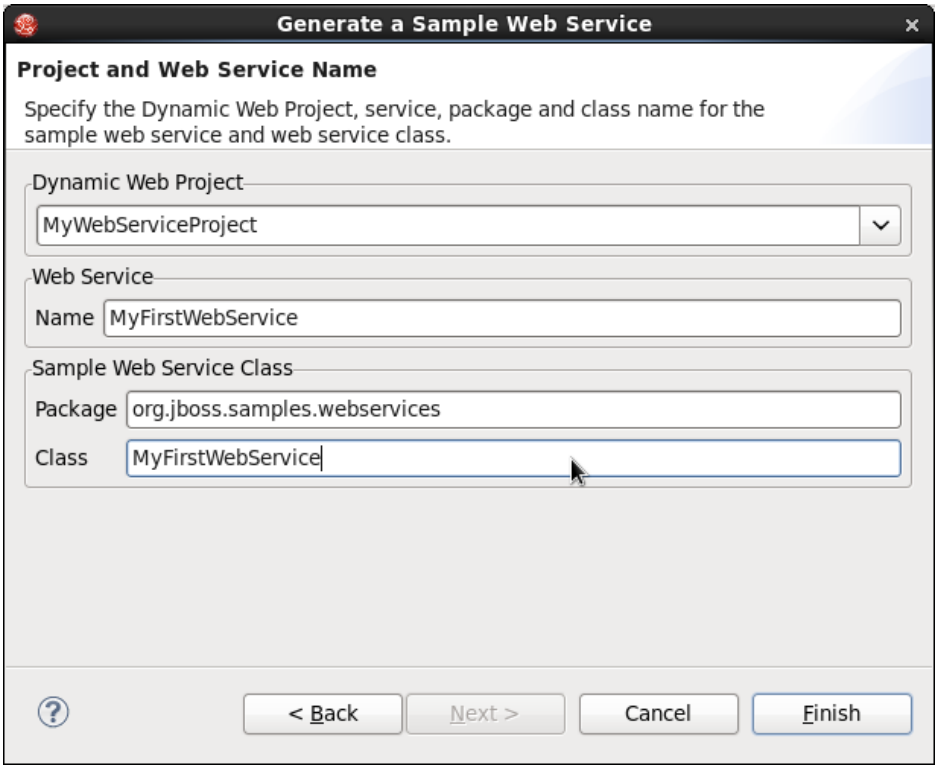


Illustration 92: Configure the web service

Click Finish. This will generate the JSR-181 annotated class as shown:

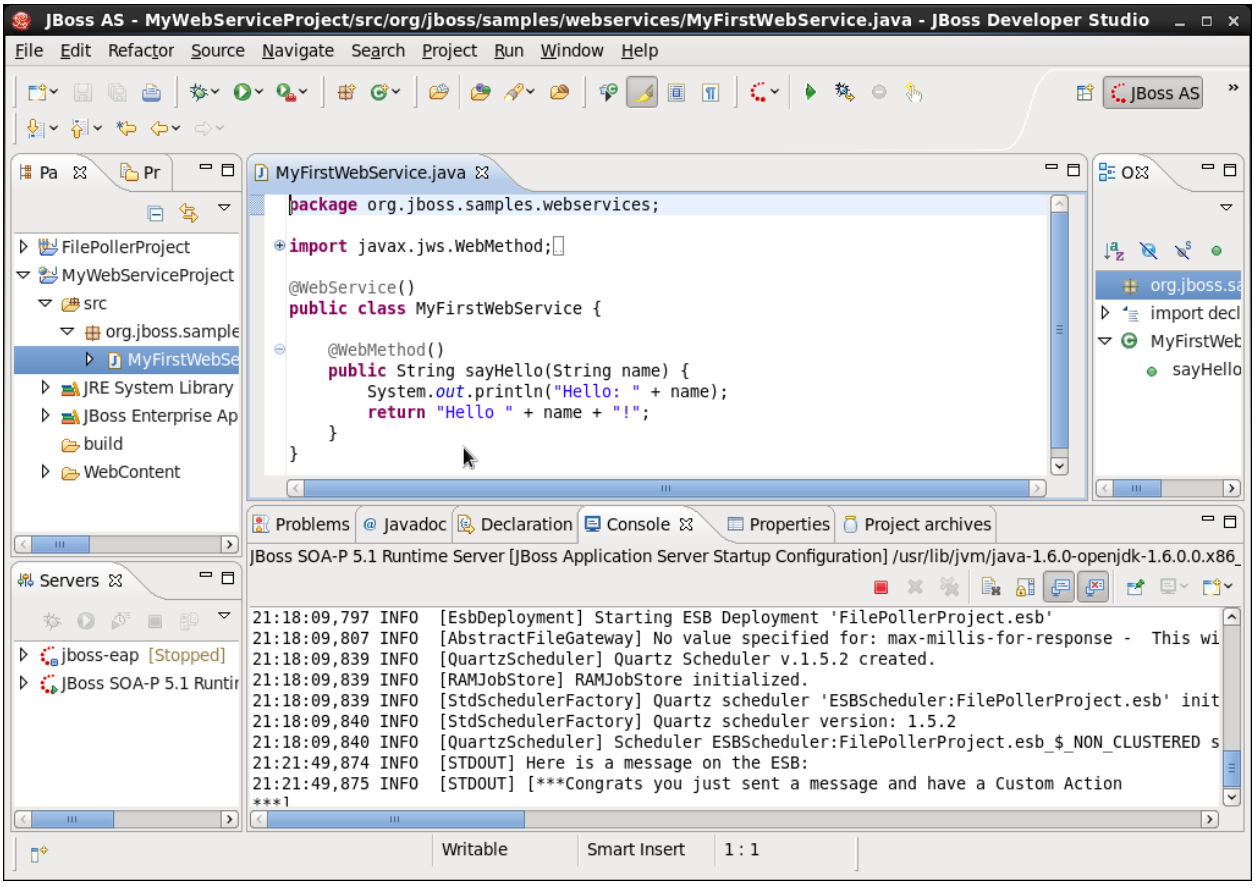


Illustration 93: Auto-generated web service

It will also have generated the mappings in the web.xml file for you automatically, as shown:

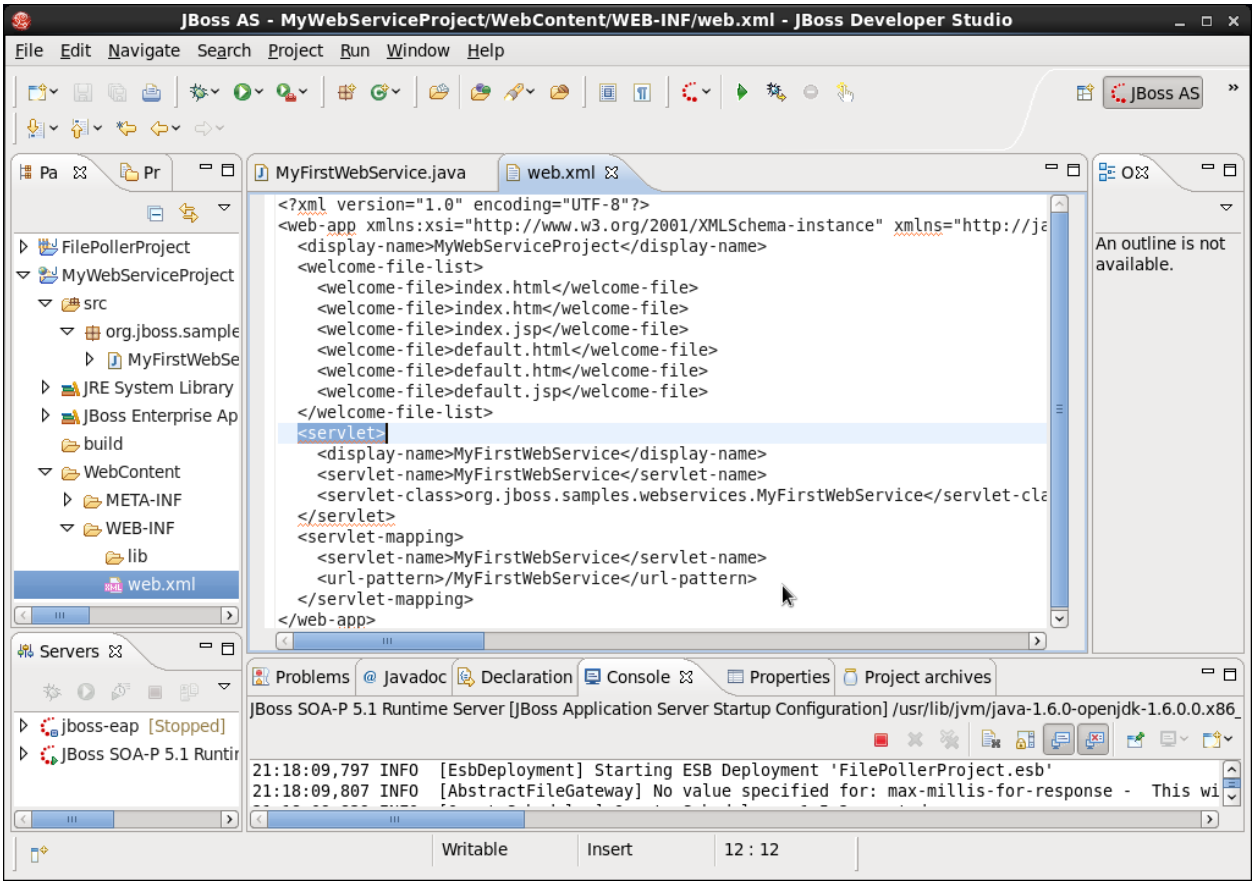


Illustration 94: Auto-generated mappings

Deploy

Now it is time to deploy this web service. So, save the project “File - > Save All”. Then, right-click on the “JBossSOA-P 5.1 Runtime Server” under the Servers tab (bottom of JBDS) and select “Add and Remove...”, highlight the project and click “Add” and Finish as shown:

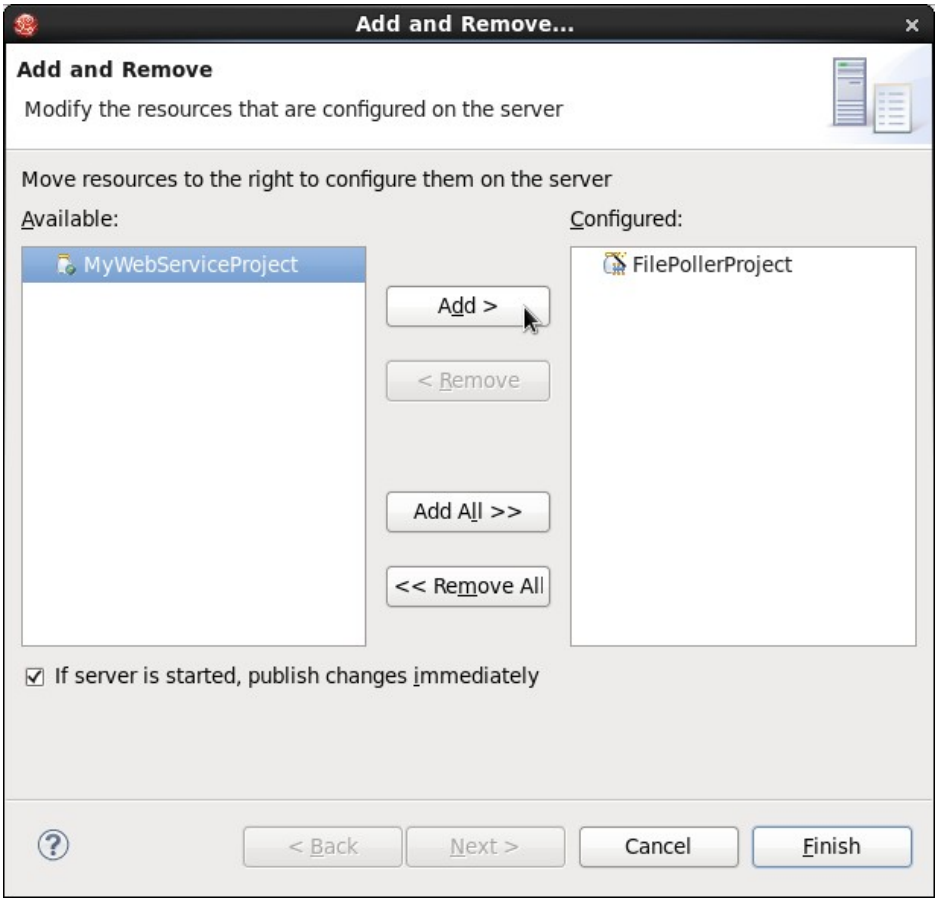


Illustration 95: Deploy the new web service

You should see that the web service has deployed in the console as shown:

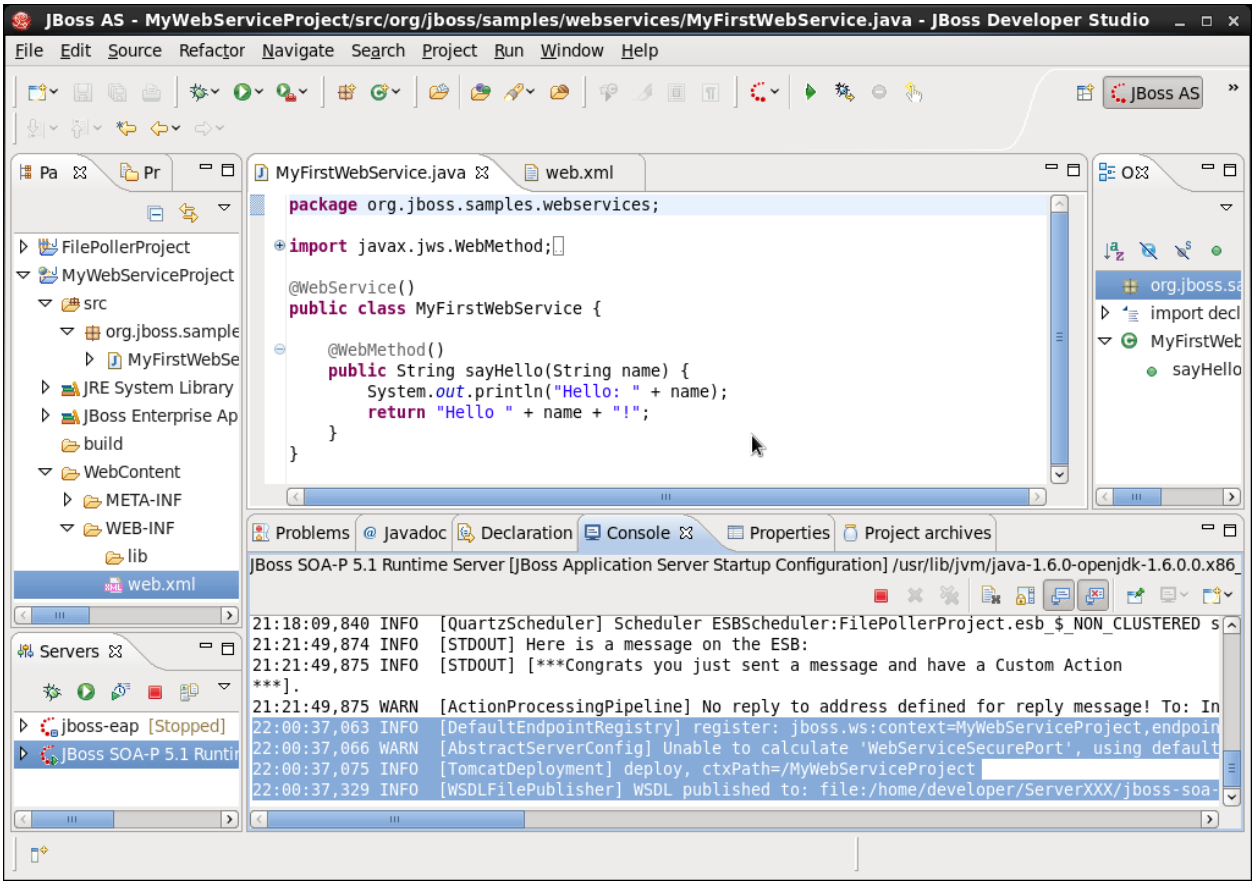


Illustration 96: Web service is deployed

To browse to your deployed webservice, you can go to http://localhost:8080/jbossws/services, you will need to type in admin admin for the username and password, and scroll down a bit to see your deployed MyFirstWebService, as shown:

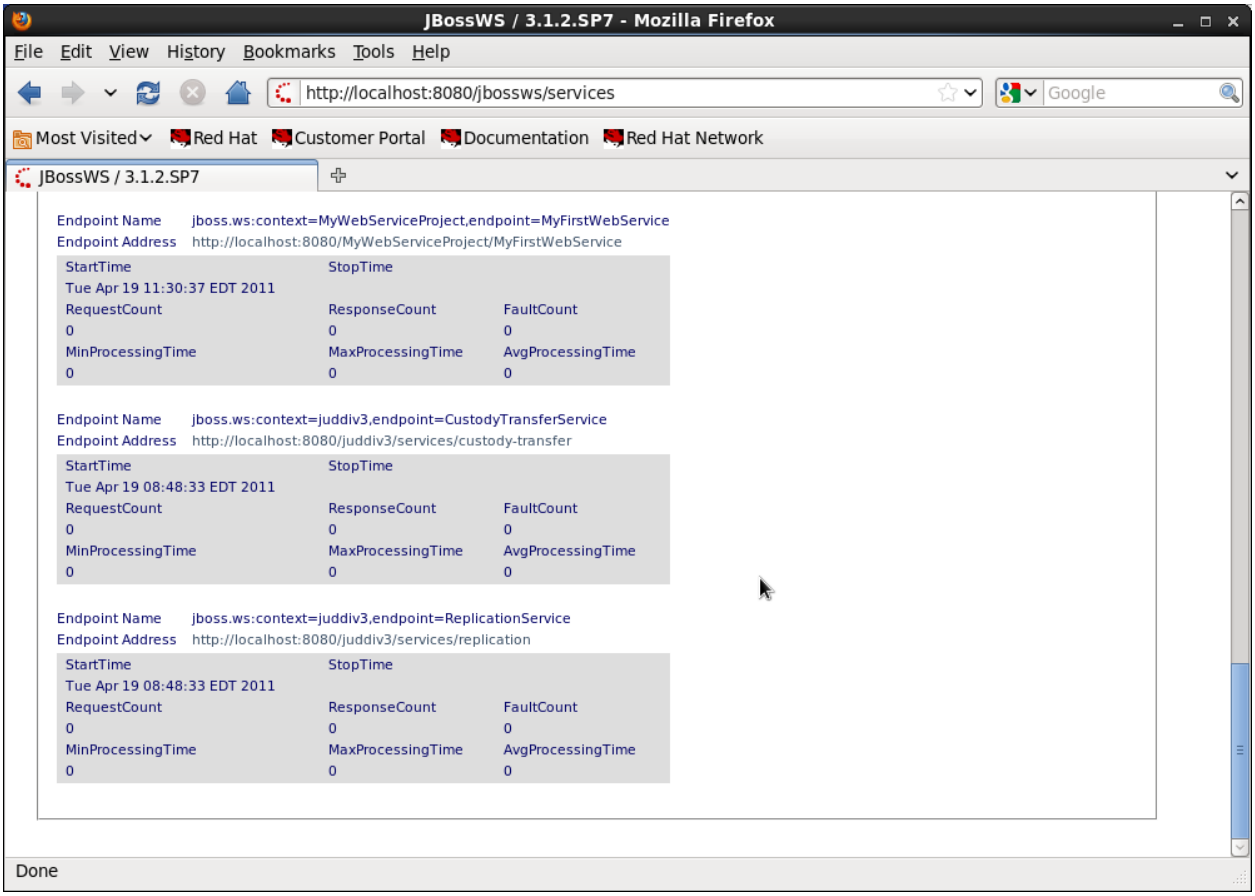


Illustration 97: Confirm web service deployment

Execute via soapUI

To test the web service we will need to use soapUI. Go to File -> New soapUI Project and the wsdl you will need to use is, http://localhost:8080/MyFirstWebServiceProject/MyFirstWebService?wsdl

You should fill in the dialog as shown:

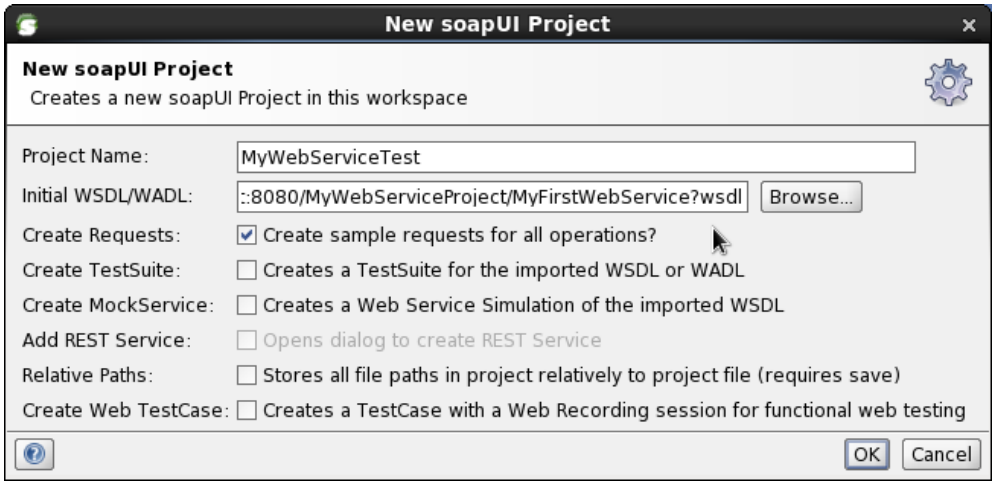


Illustration 98: Configure soapUI project

Click OK. When soapUI is done importing, it should look like this:

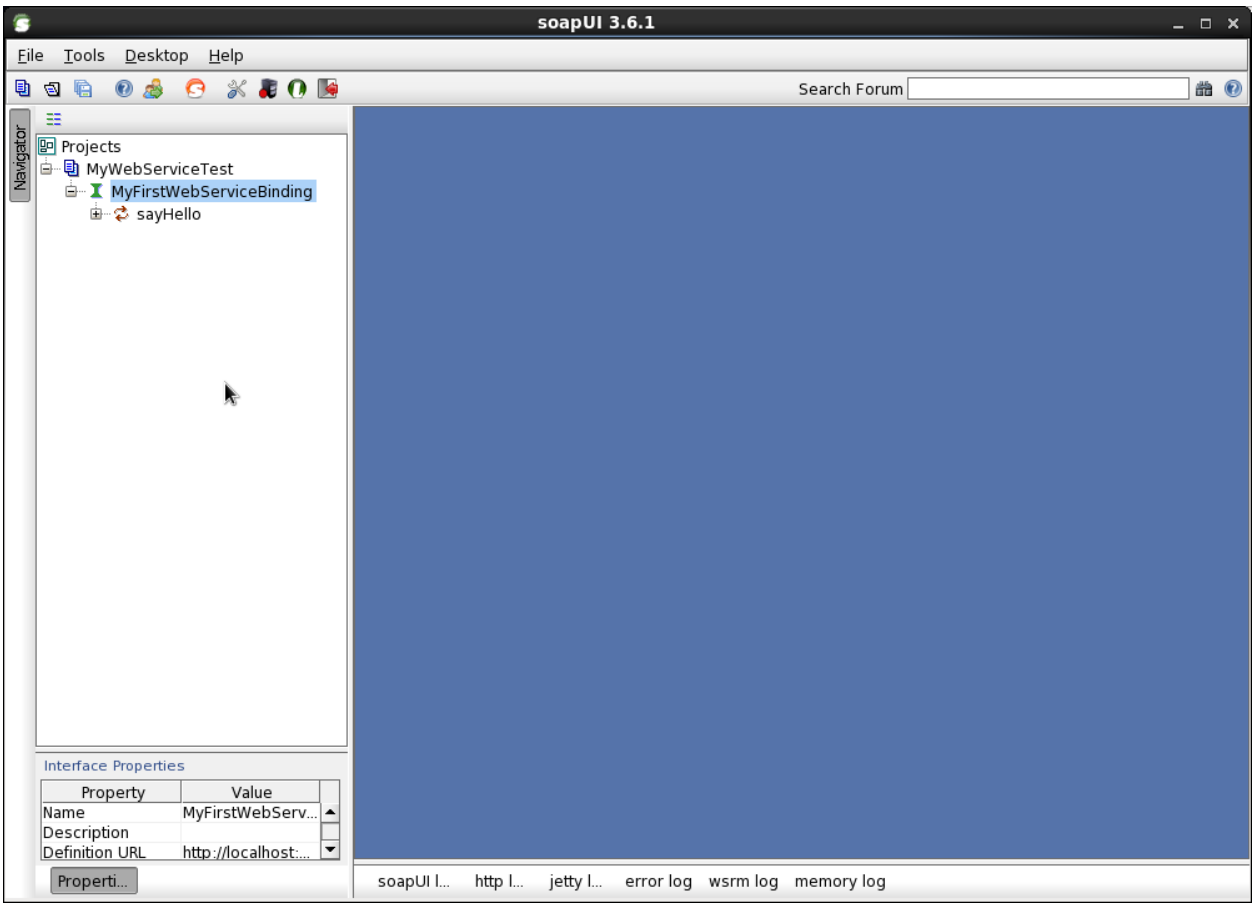


Illustration 99: Imported WSDL in soapUI

Now expand the sayHello by clicking on it so that Request 1 shows. Right-click on Request 1 and select "Show Request Editor" to get the window shown below. Replace the ? in the left window with your name, and press the green play button to invoke the web service. You should see output on the right hand side as shown:

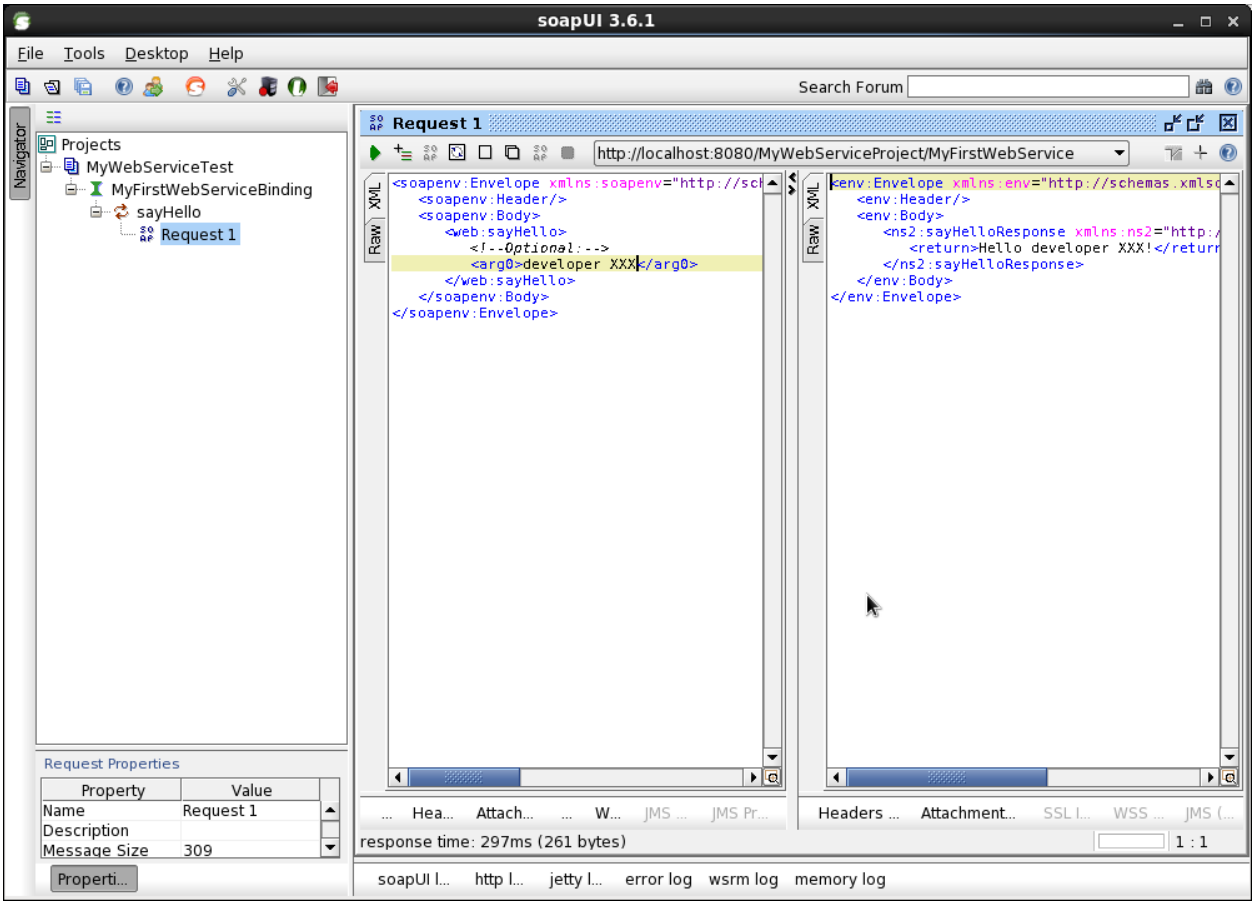
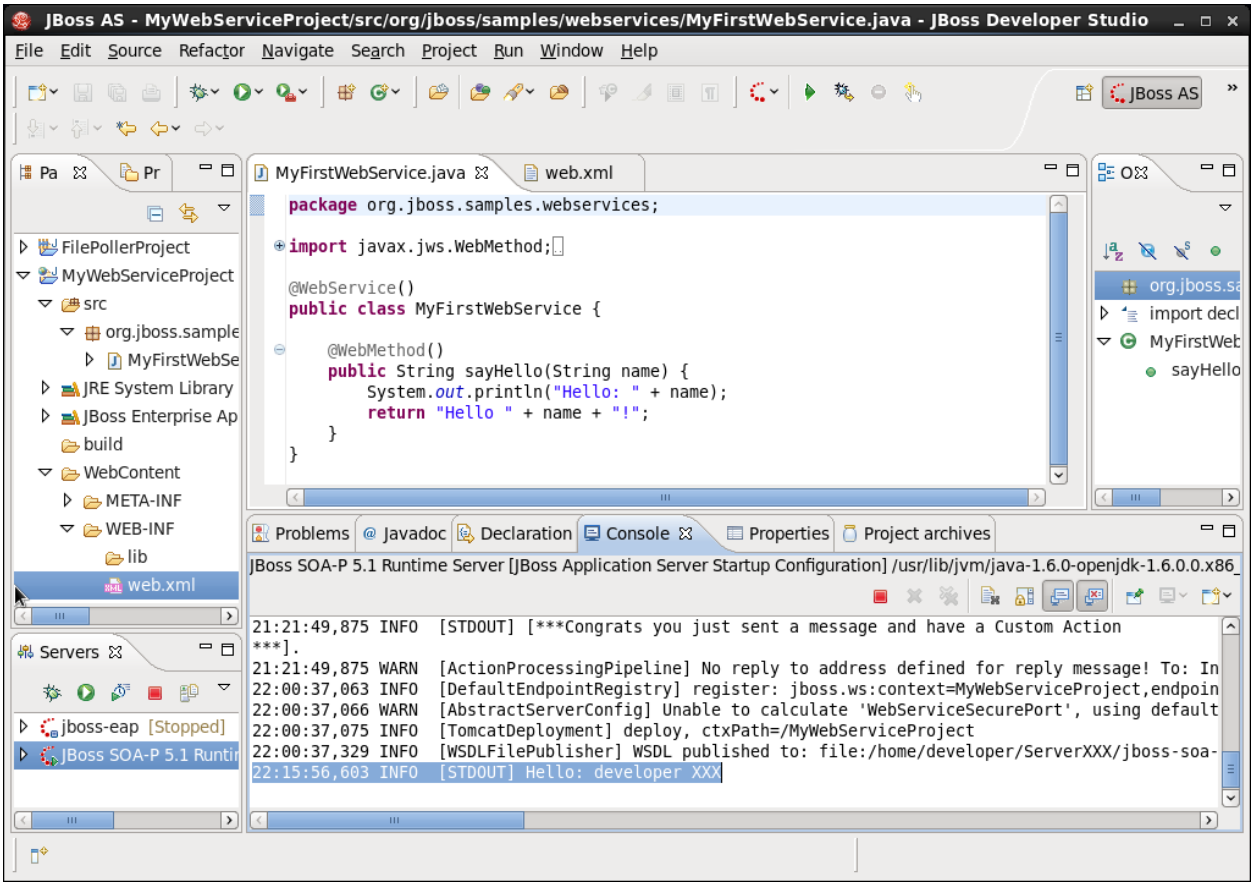


Illustration 100: Test the web service

Also notice the output in the console in your JBDS instance as shown:



This lab is now complete, if you are having any problems with this lab please raise your hand. Congratulations you have now deployed a JSR-181 web service, you have tested it, and you are now ready to have the SOA-P proxy the communication with the web service.

Lab Number 9: Proxy the Just Created Web Service

One of the most common use cases for an ESB is to mediate web services. The client sends a request to the ESB, the ESB may do specialized security, transformations, routing, or a host of other actions, and then the ESB routes to a back-end web service implementation. We will proxy the just created web service in this lab.

Start by creating a new ESB Project File -> New -> Other

From the pop up window search on esb and Select ESB Project as Shown, Click Next:

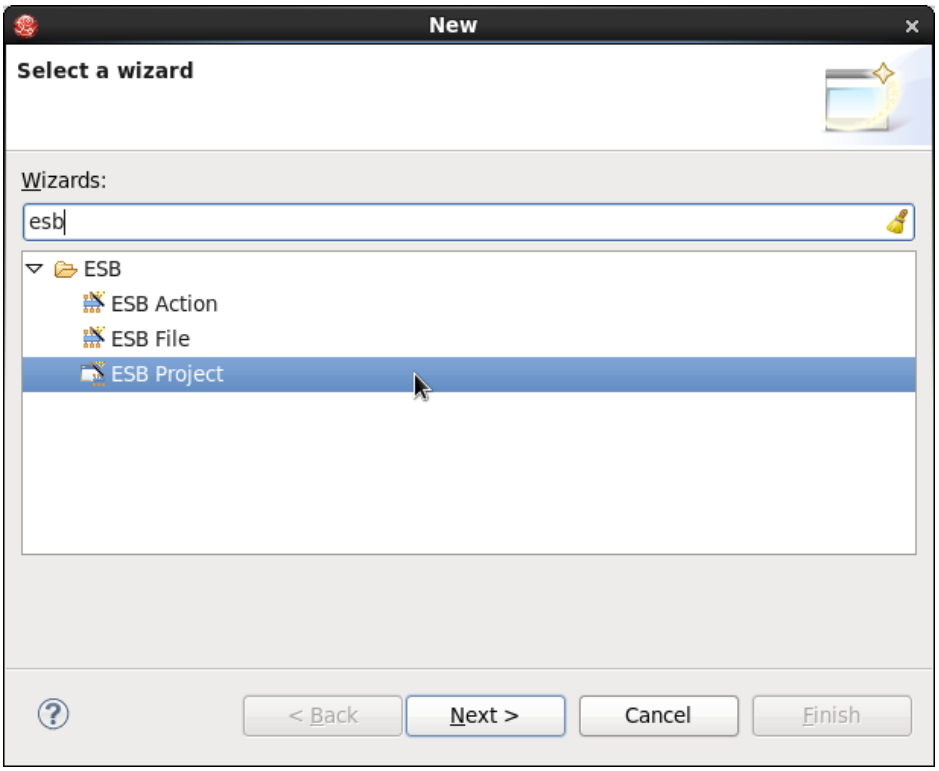


Illustration 101: Select ESB wizard

Make sure the Target runtime is the Jboss SOA-P 5.1 Runtime as shown, and type in the Project Name ESBWebServiceProxy as shown:

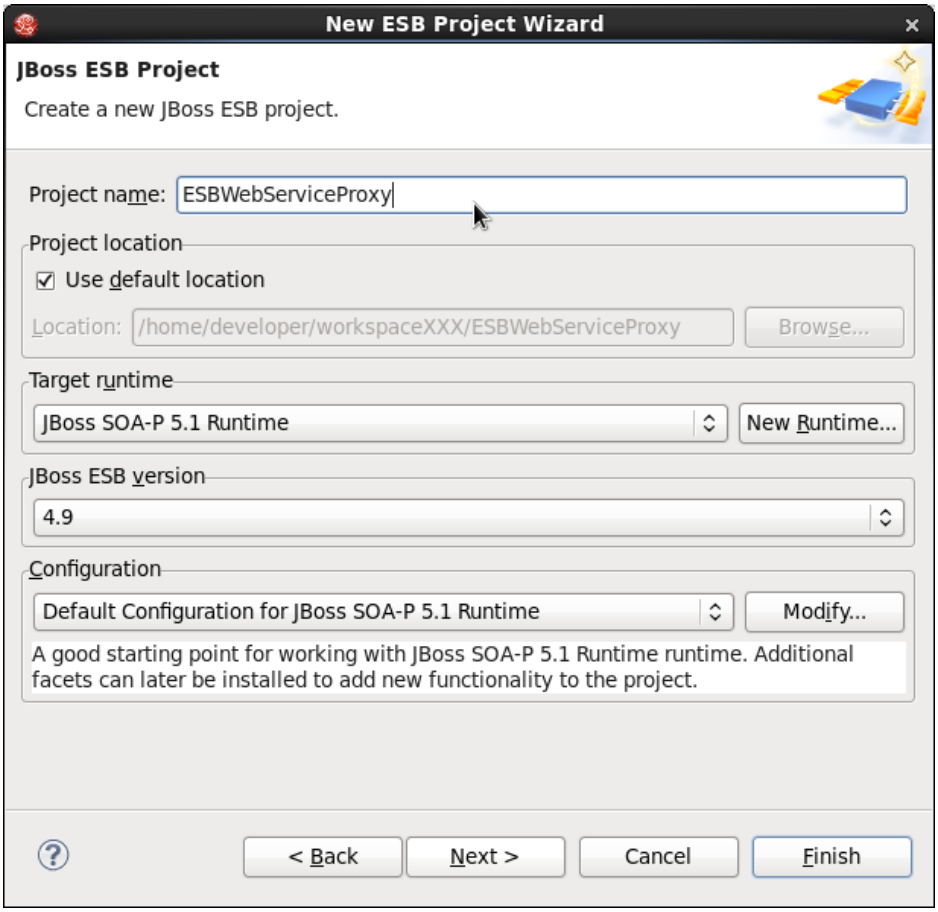


Illustration 102: Configure ESB project

Click Finish. As before we basically have the same steps to create a new ESB Projects, Providers and Services:

First right-click on Providers and select the HTTP Provider as shown:

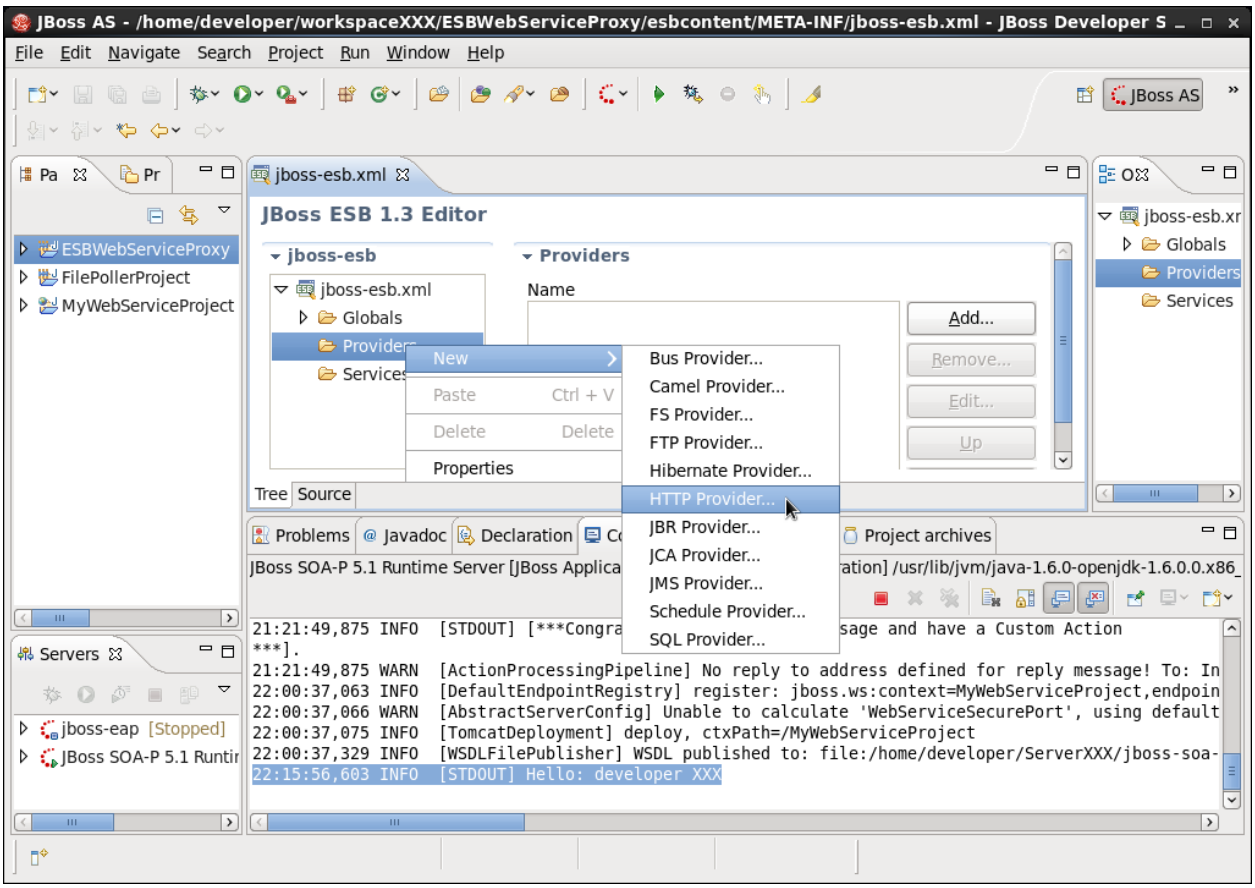


Illustration 103: Add new provider

Fill in the Name as “HTTP Proxy Provider”

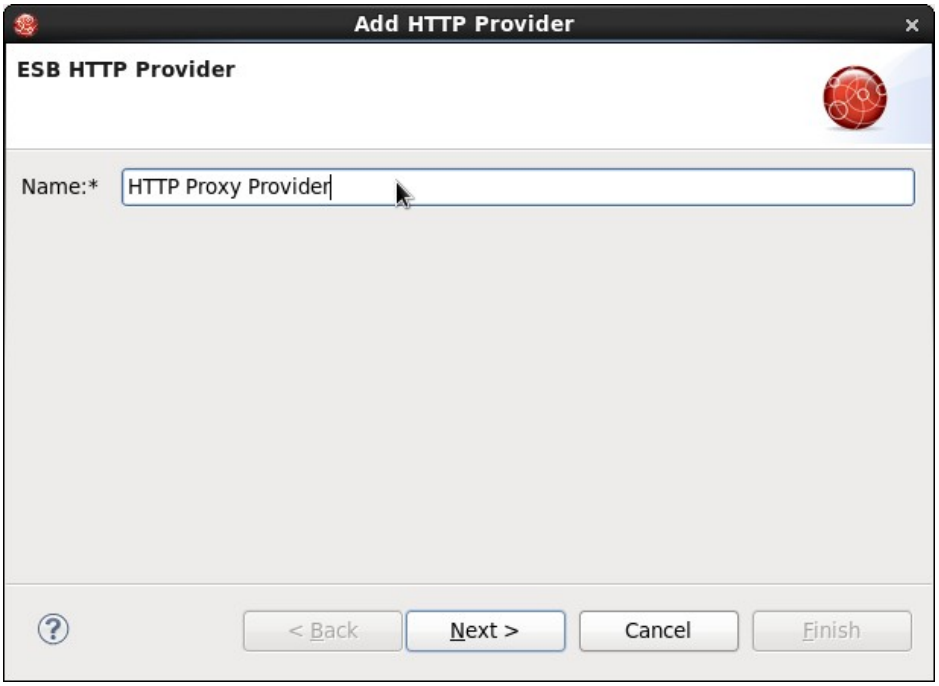


Illustration 104: Set provider name

Click Next. Fill in a Channel ID, “HTTP Proxy Channel ID”, as shown:

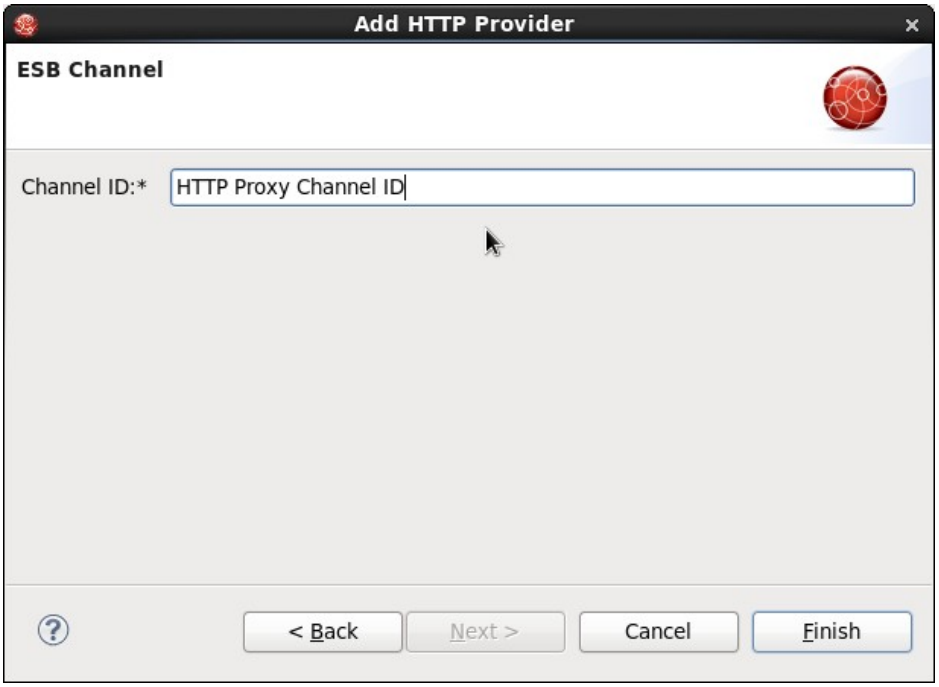


Illustration 105: Set channel ID

Click Finish. Now right click on Services as shown:

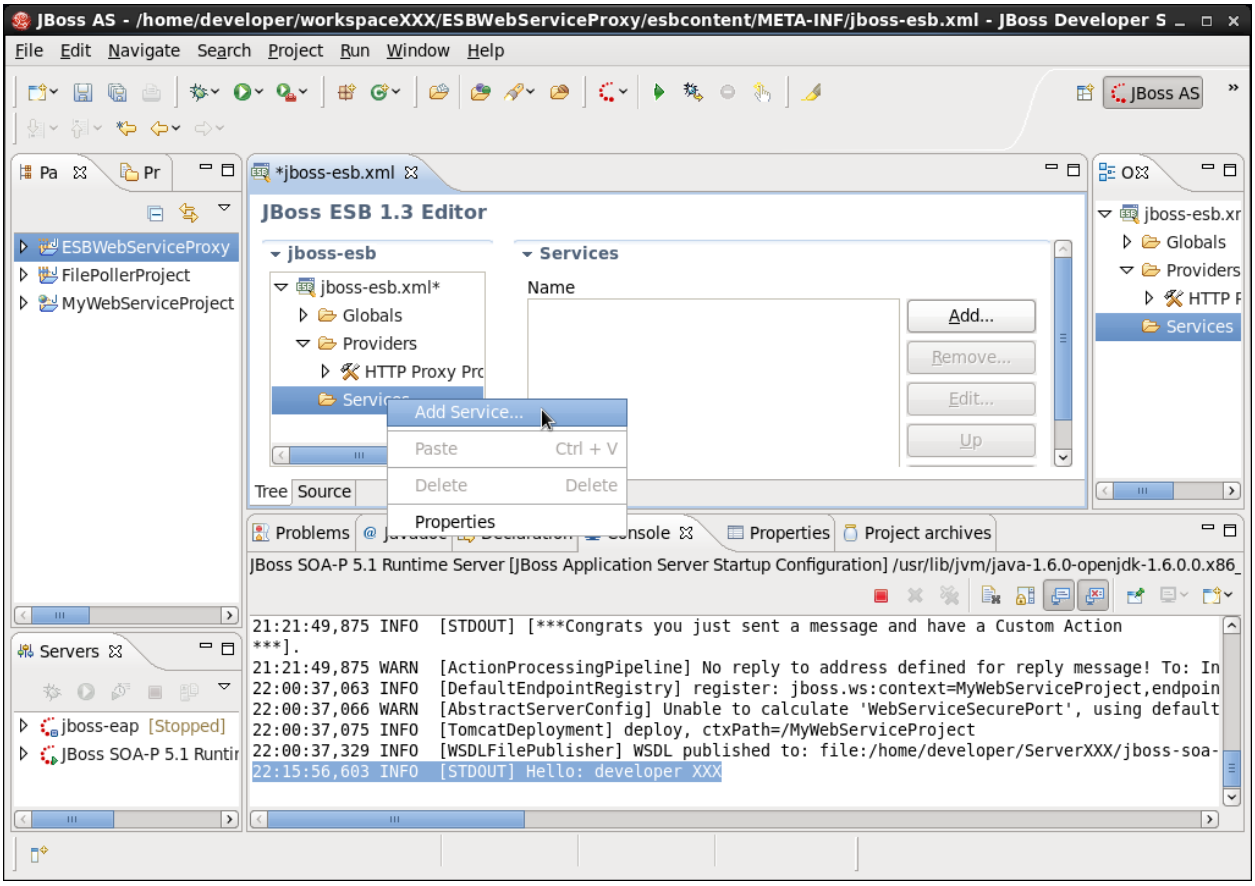


Illustration 106: Add a new service

Fill in the name “HTTP Proxy Service”, category as “Lab Services”, description as “Service to Proxy Web Services”, as shown:

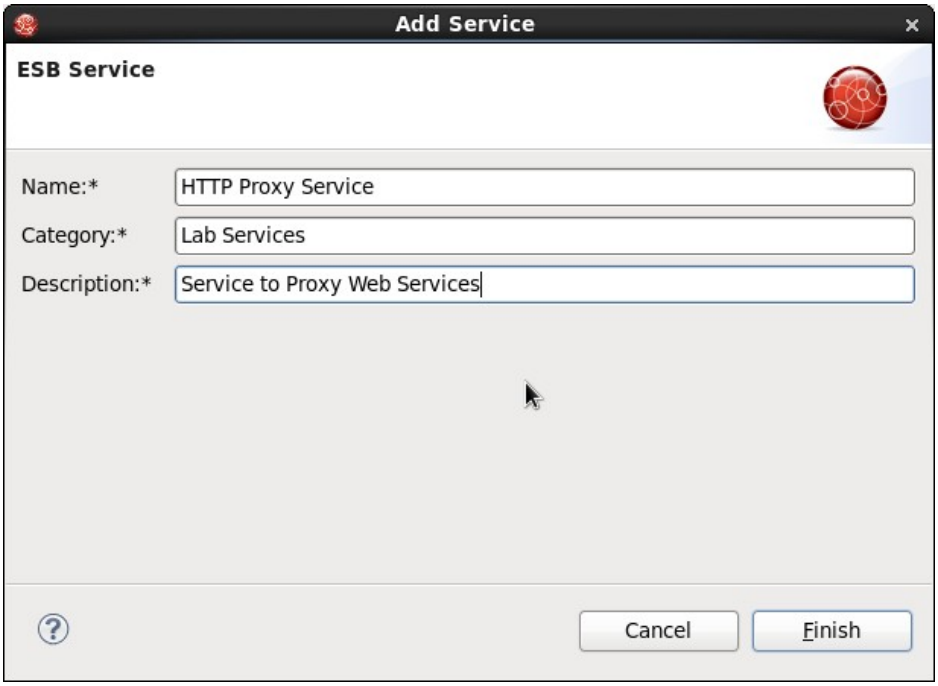


Illustration 107: Configure the service

Click Finish. Change the Invm Scope to Global as shown below:

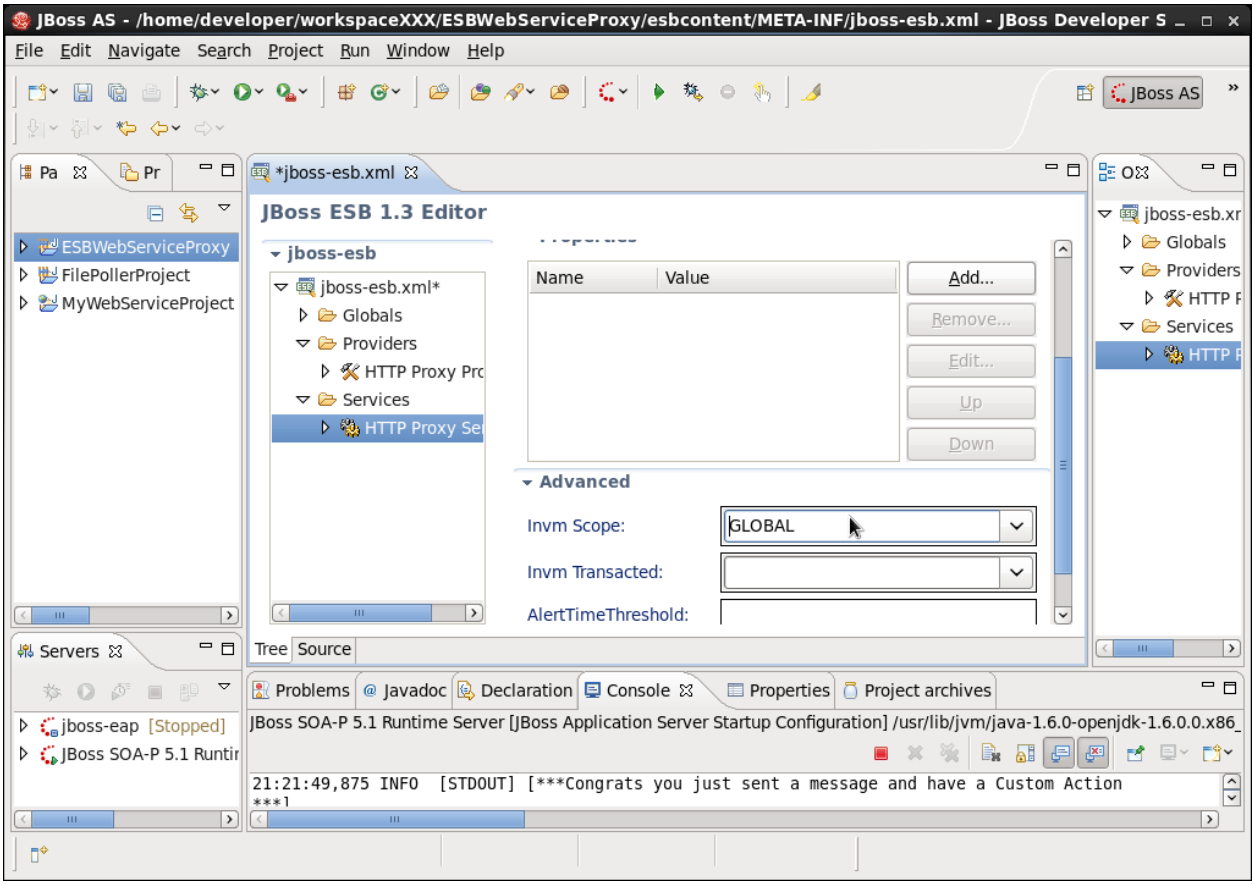


Illustration 108: Optimize the service

Now we need to add a listener/gateway and bind it to the provider we just created. Right-click on Listeners, selecting HTTP Gateway as shown:

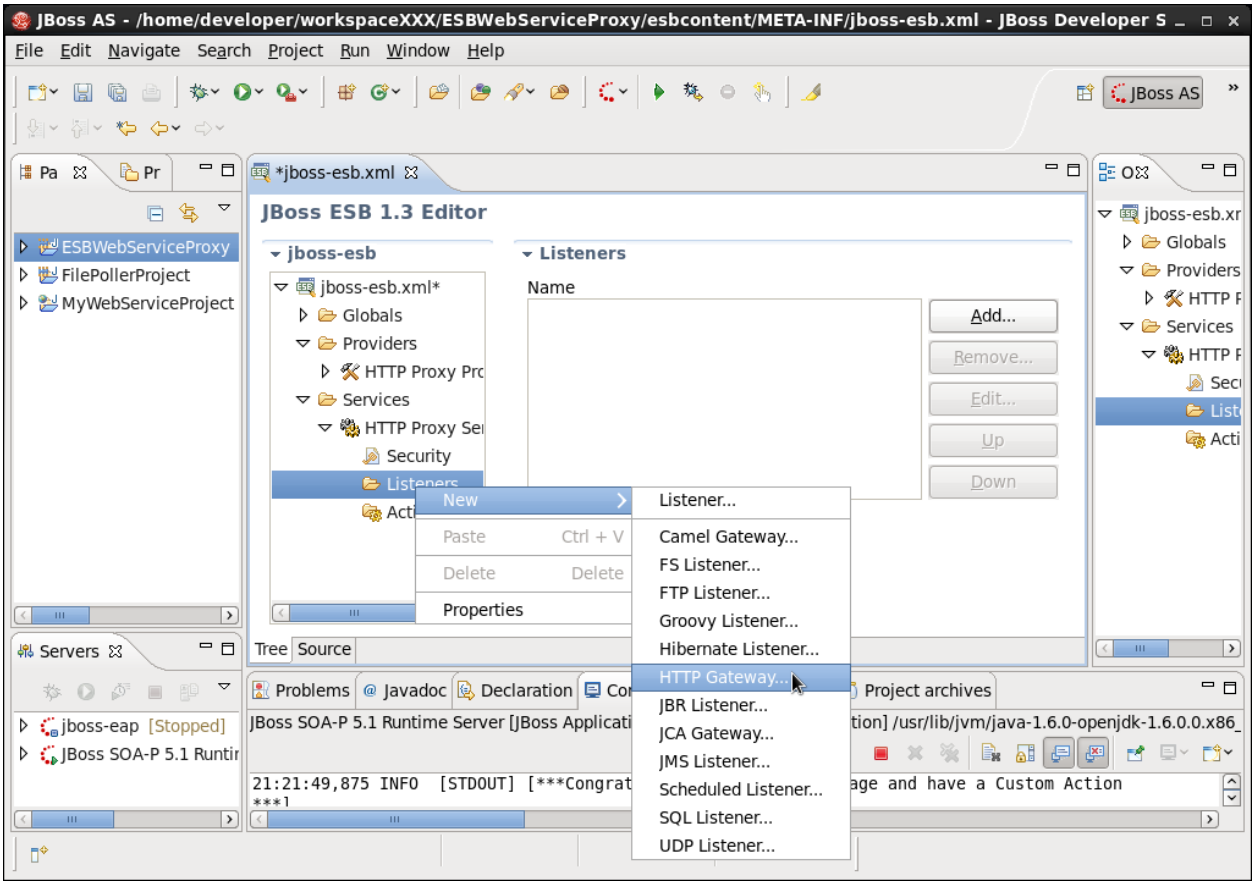


Illustration 109: Create gateway

Now fill in the name as “HTTP Proxy Listener” and channel ID ref as “HTTP Proxy Channel ID” as shown:

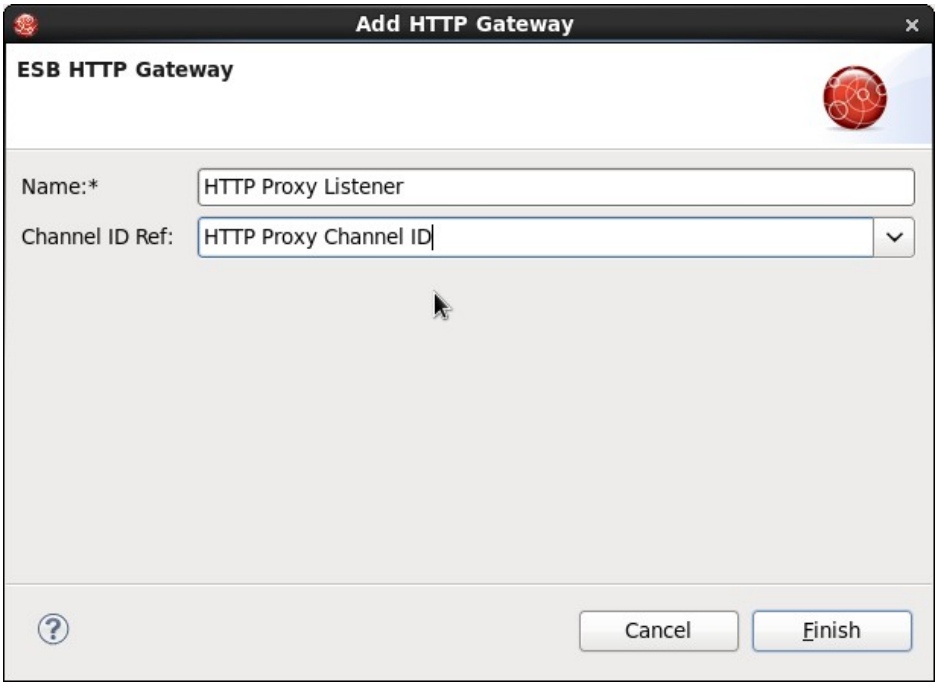


Illustration 110: Configure the gateway

Click Finish. Now update the URL Pattern to be “proxy/*” as shown:

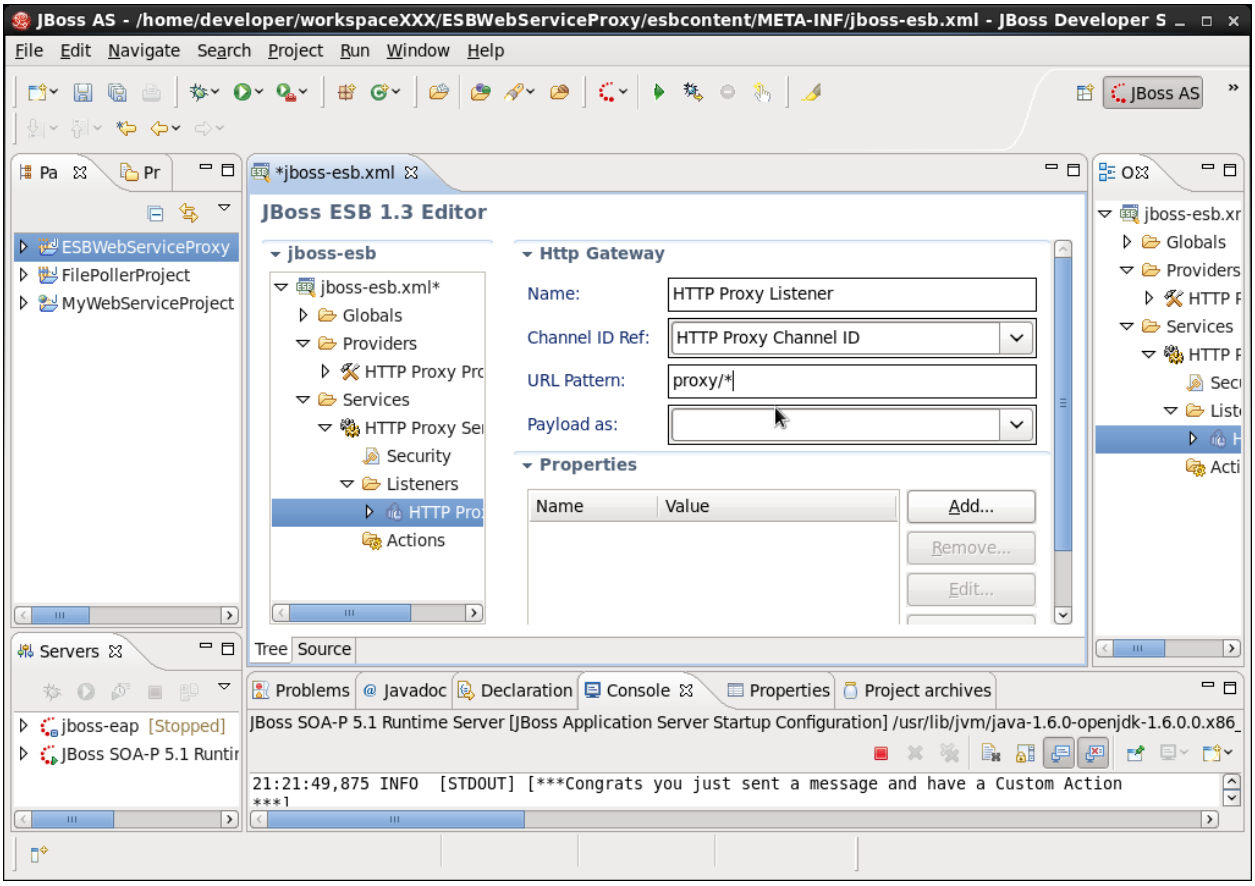


Illustration 111: Change URL pattern

Now lets add a Println action as before. Right click on Actions as shown below:

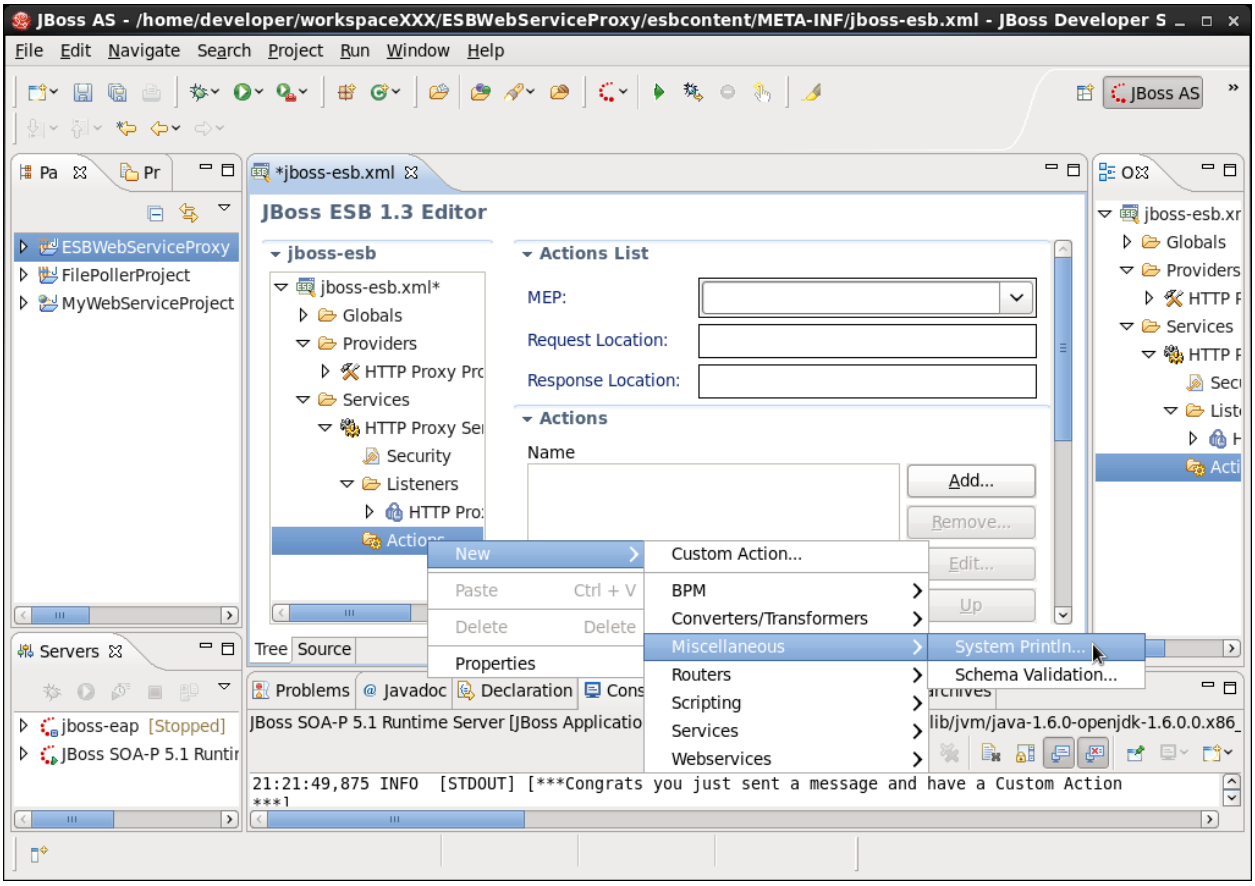


Illustration 112: Add an action

Add in a name for this action “Print Line Action” and a message “Here is our test message”

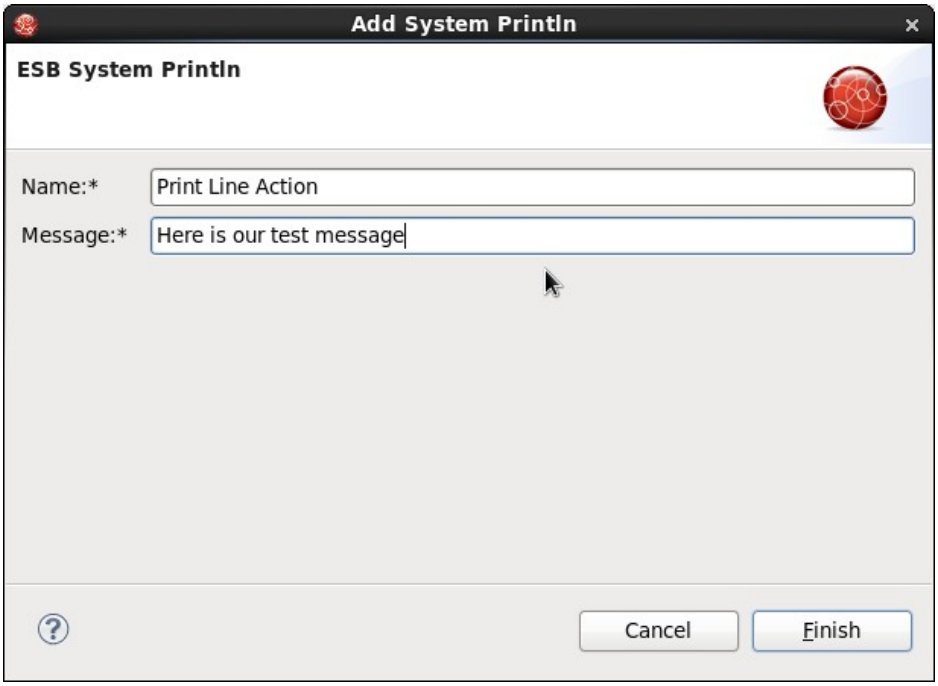


Illustration 113: Configure the action

Click Finish. Let's save this off and test and deploy it.

Select the “Servers” tab at the bottom-left and then right-click on the JBoss SOA-P 5.1 Runtime Server and select add/remove projects:

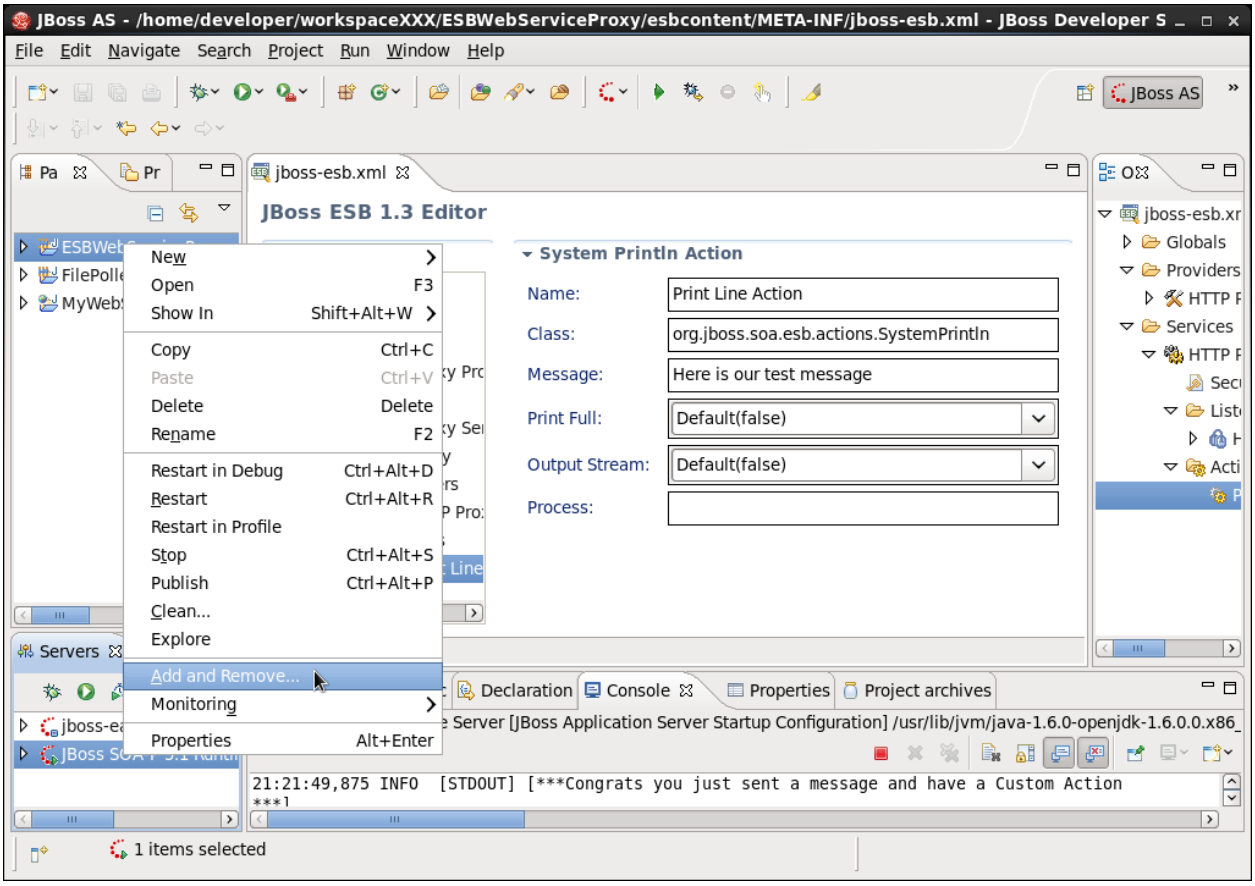


Illustration 114: Prepare to deploy the project

Then add the project and make sure it shows up on the right-hand side:

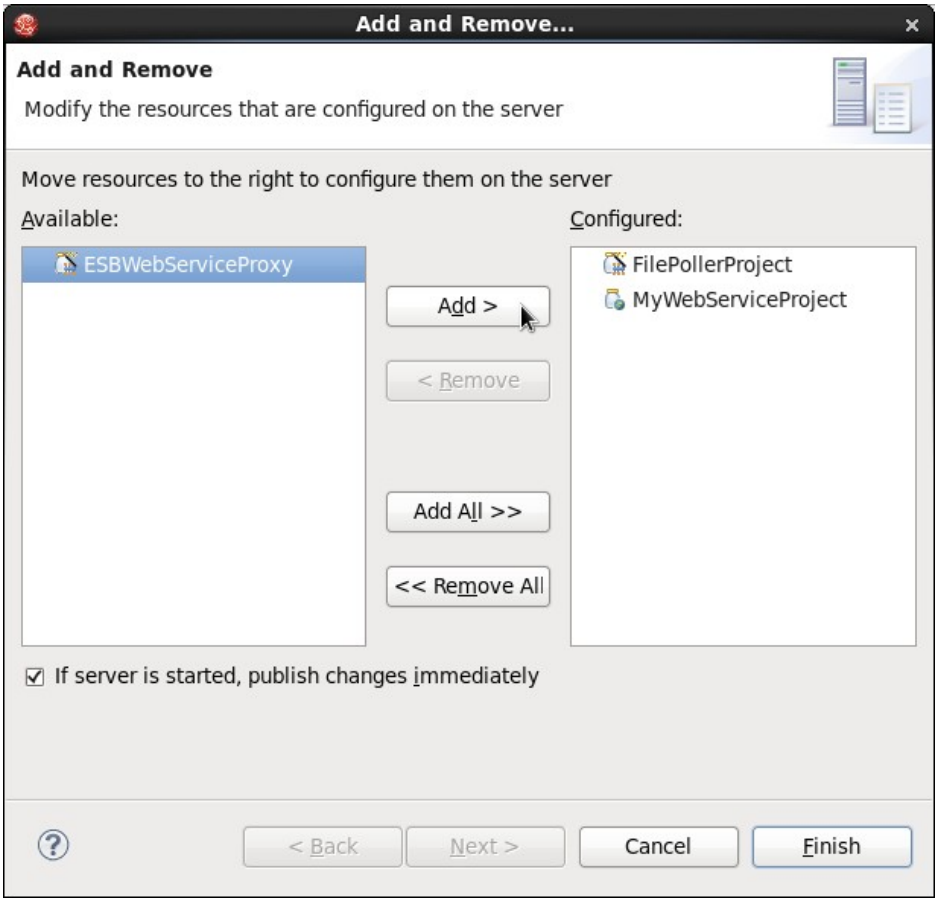


Illustration 115: Deploy the project

Click Finish once it is over there. Note in your console that the application has deployed.

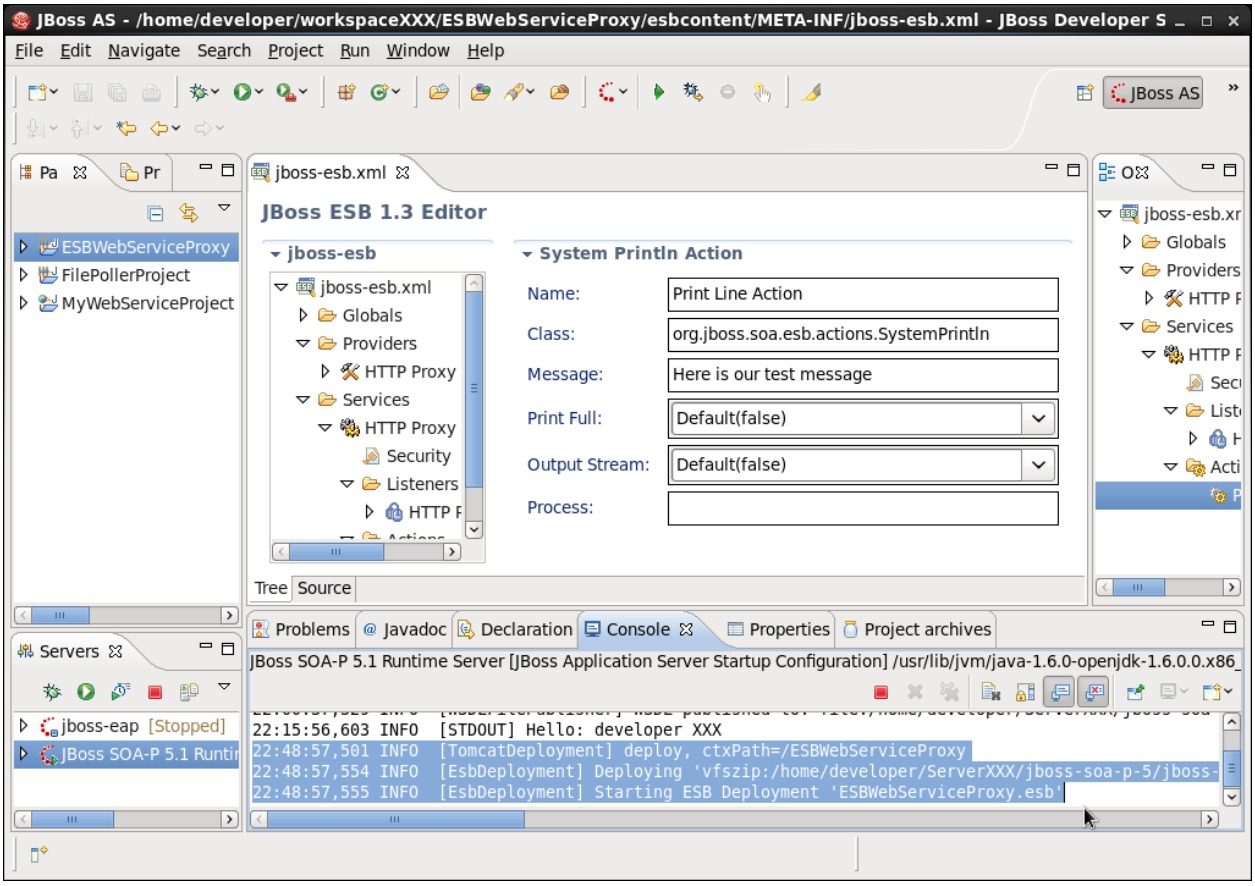


Illustration 116: Confirm that the project deployed

Test the Proxy

Now it is time to see if this works, go back to soapUI and update the URL you are testing to http://localhost:8080/ESBWebServiceProxy/http/proxy as shown:

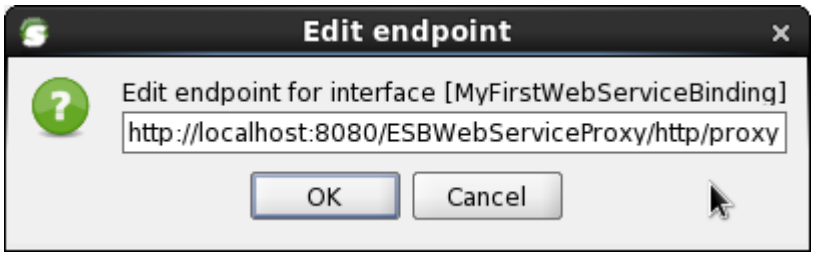


Illustration 117: Change soapUI endpoint

Click the green play button and see the result, which is basically just what you passed in, and also notice the output in the console. SoapUI output is unchanged:

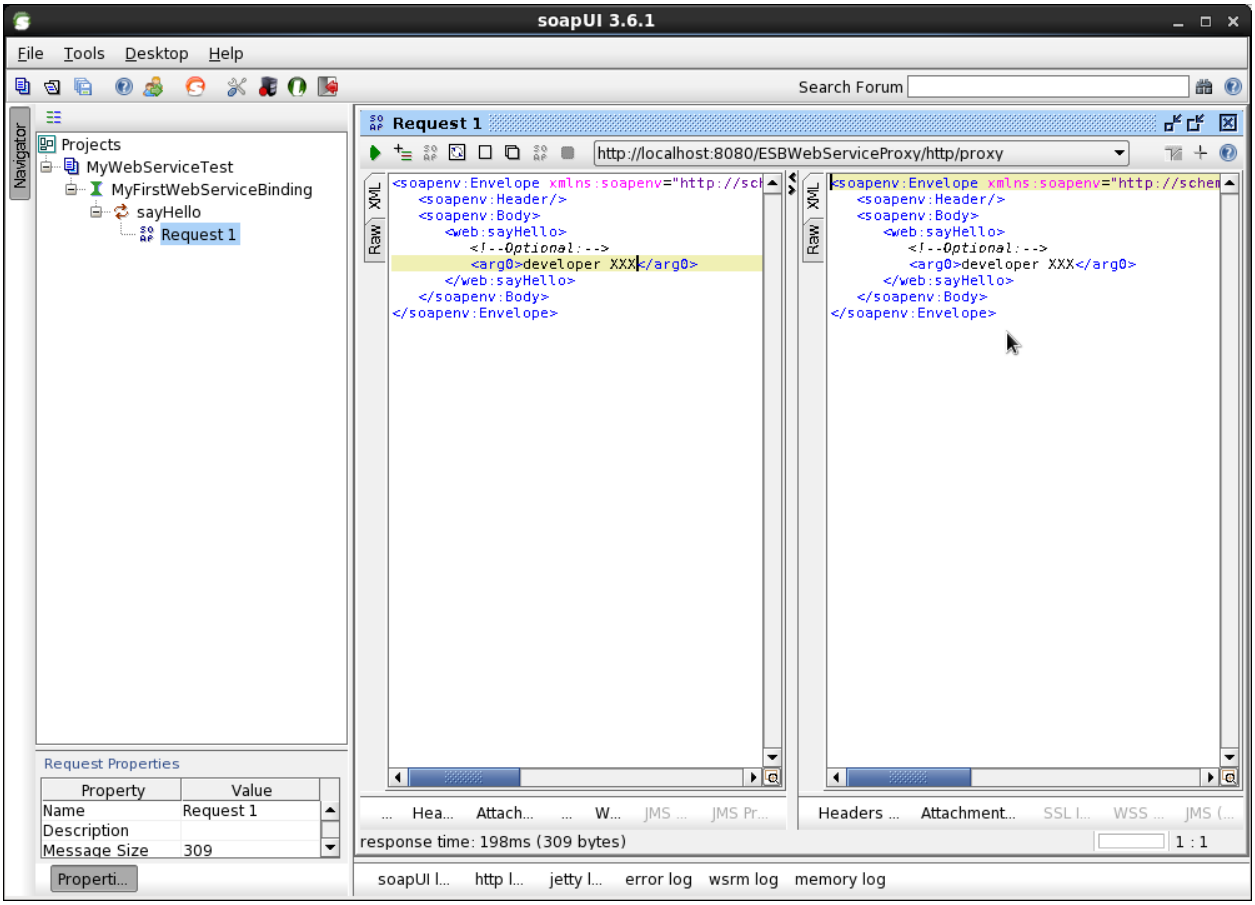


Illustration 118: soapUI output unchanged

Now see the console output in JBDS:

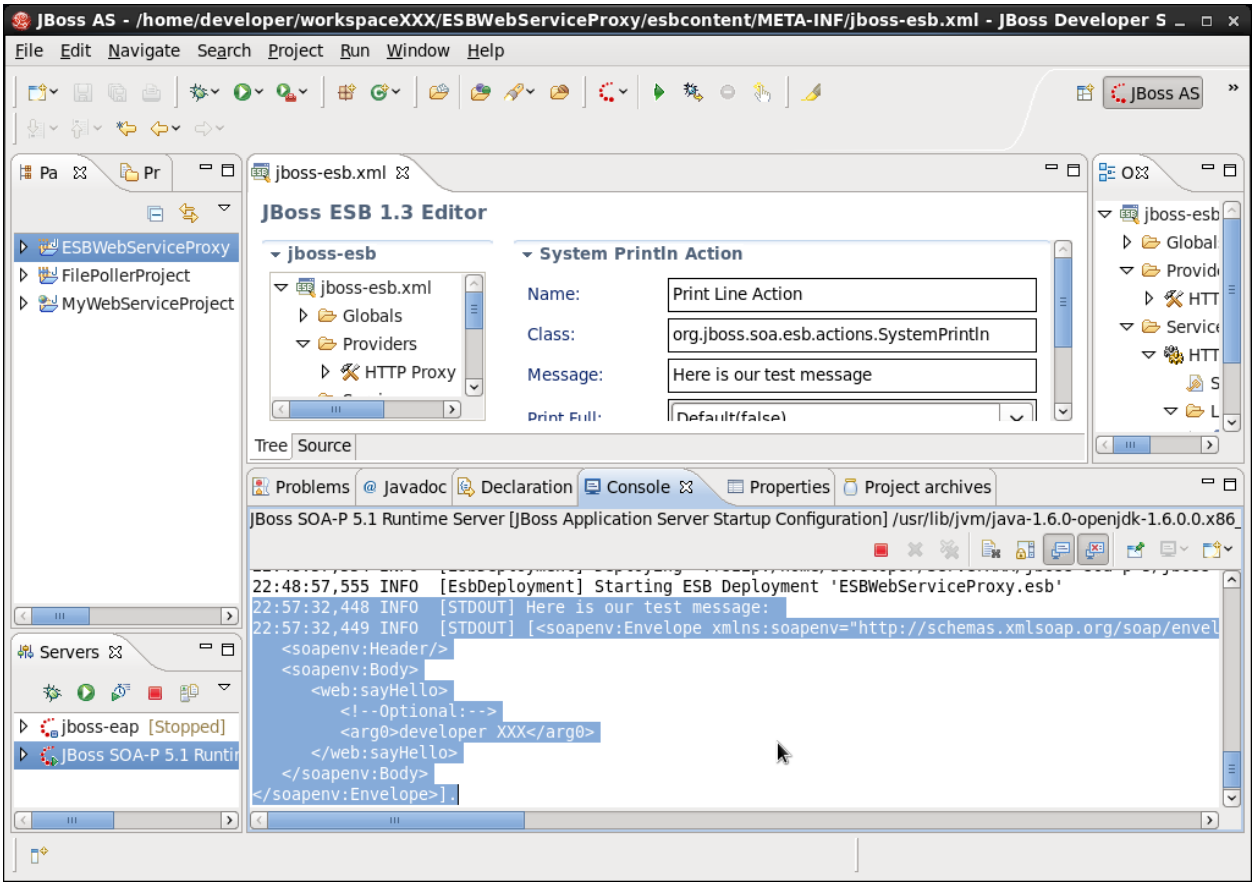


Illustration 119: JBDS console output

Okay that is great, but this lab was about proxying the call to the JSR-181 Web Service we created earlier. Well, that is as simple as creating a new action.

Right click on Actions, and do New -> Routers -> HTTP Router as shown:

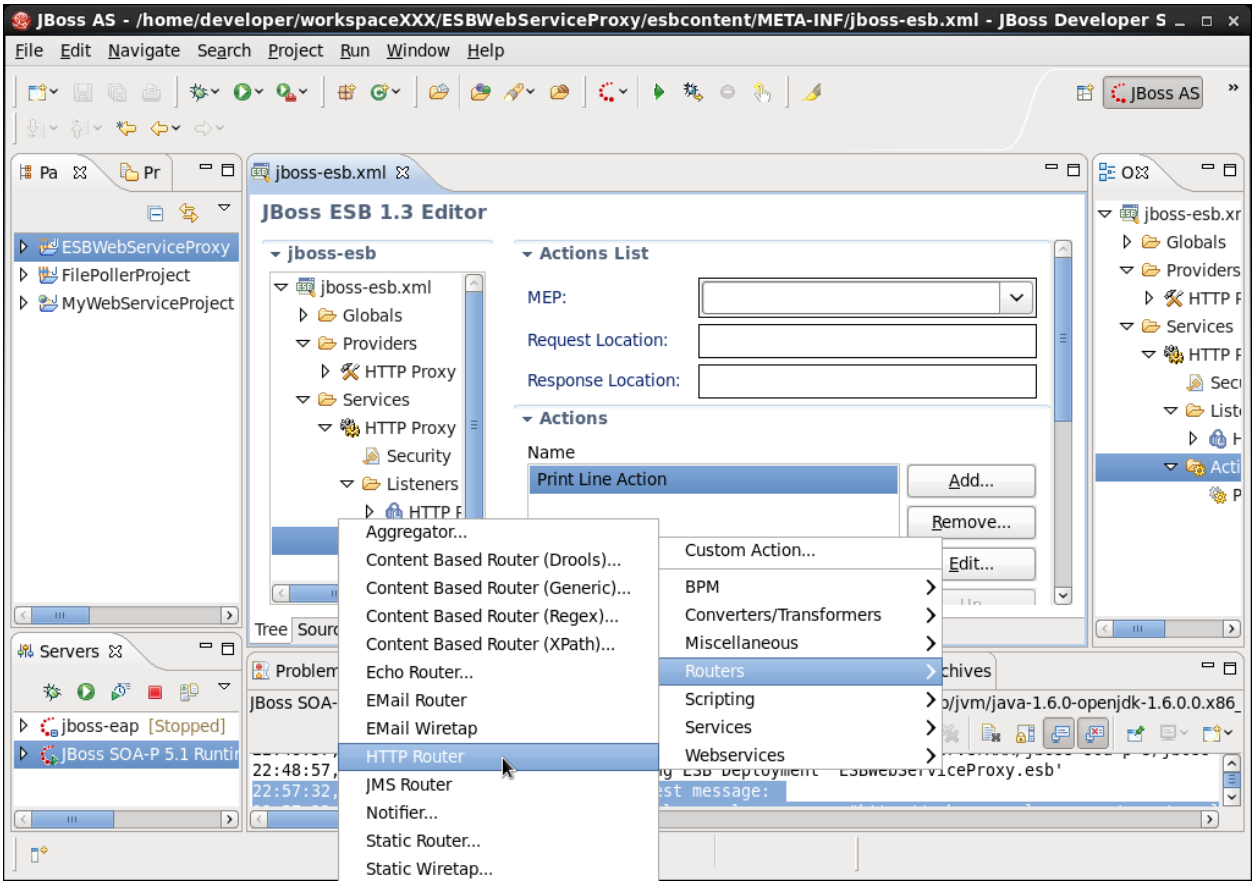


Illustration 120: Create new router

Now fill in the name as “JSR-181” the end point as “http://localhost:8080/MyWebServiceProject/MyFirstWebService” and change the method to Post as shown:

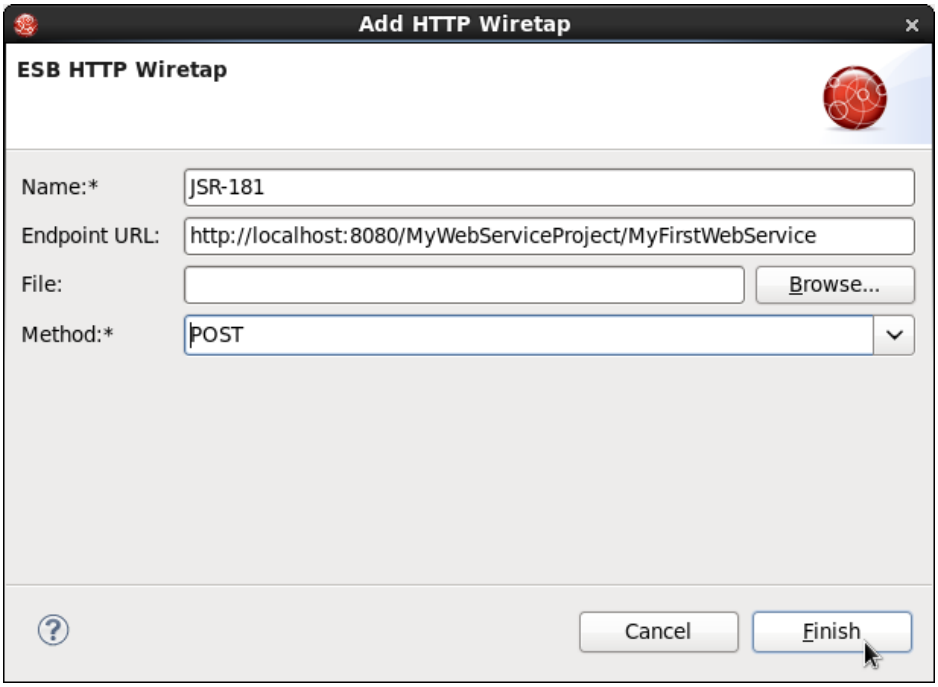


Illustration 121: Configure new action

Click Finish and save off the changes. Note it will save and redeploy automatically.

Go back to soapUI and press that green play button again and note we invoked our web service and return back a slightly better formatted message:

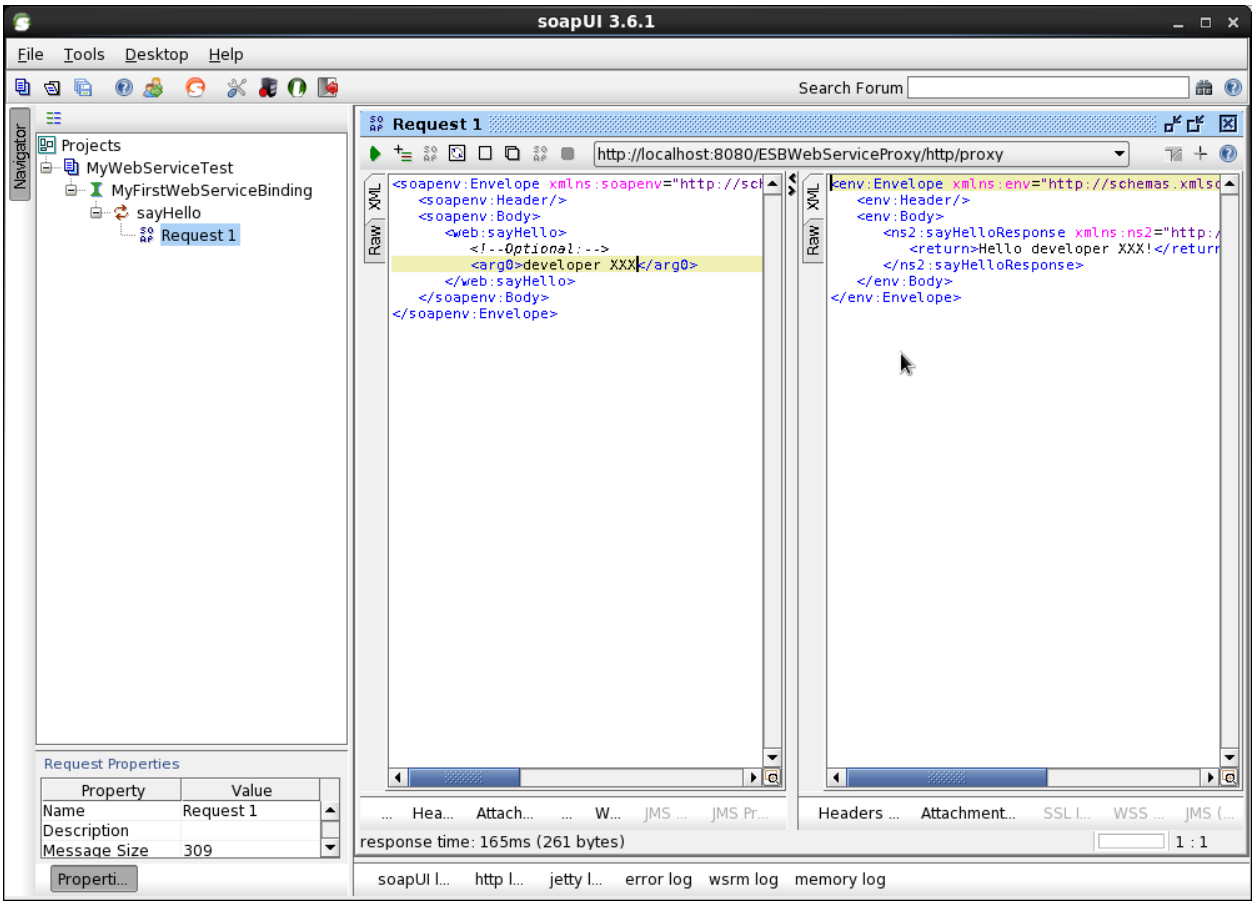
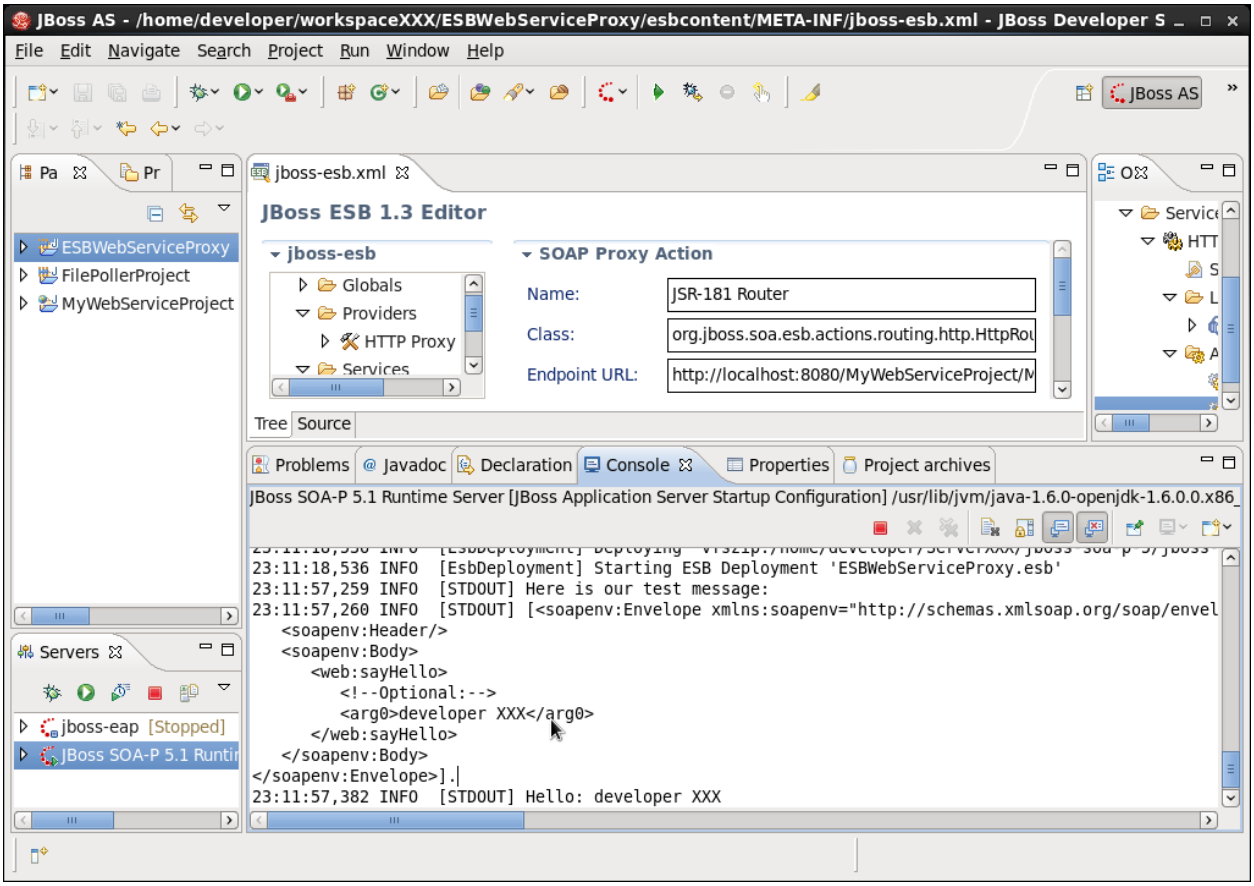


Illustration 122: Updated response message

Lets look at the console. Note our unchanged message from our print line action, and also our output from our custom code inside of our JSR-181 Web Service:



Congratulations you have now completed this lab.

Conclusion

What you learned

- How to install SOA-P
- Learned about the Quickstarts
- Learned about the hot deployment nature of the quickstarts
- Creating a Custom Action
- Learned how to create a JSR-181 Webservice
- How to test that Web Service via soapUI