Lab 1

hello_world

Lab Goals

- Introduction to hello_world in SOA 5
- Introduction to hello_world in SOA 6
- Step-by-step migration using Windup rules
- Deploy and test application in SOA 6

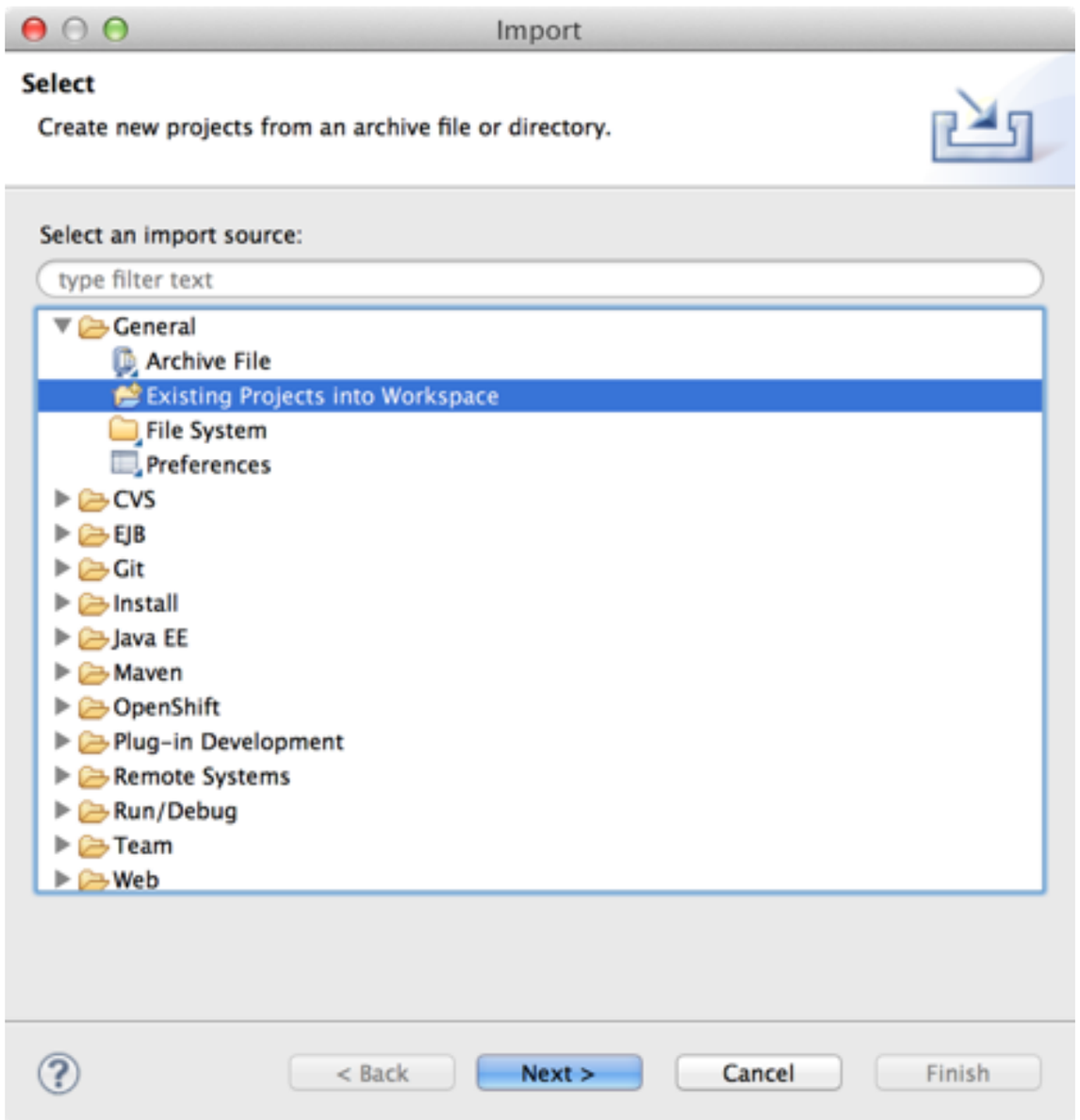
Importing SOA 5 Hello World

TODO

1. File -> Import ... from the JBDS menu.

2. Select General -> Existing Projects into Workspace

3. Click Next



Importing SOA 5 Hello World

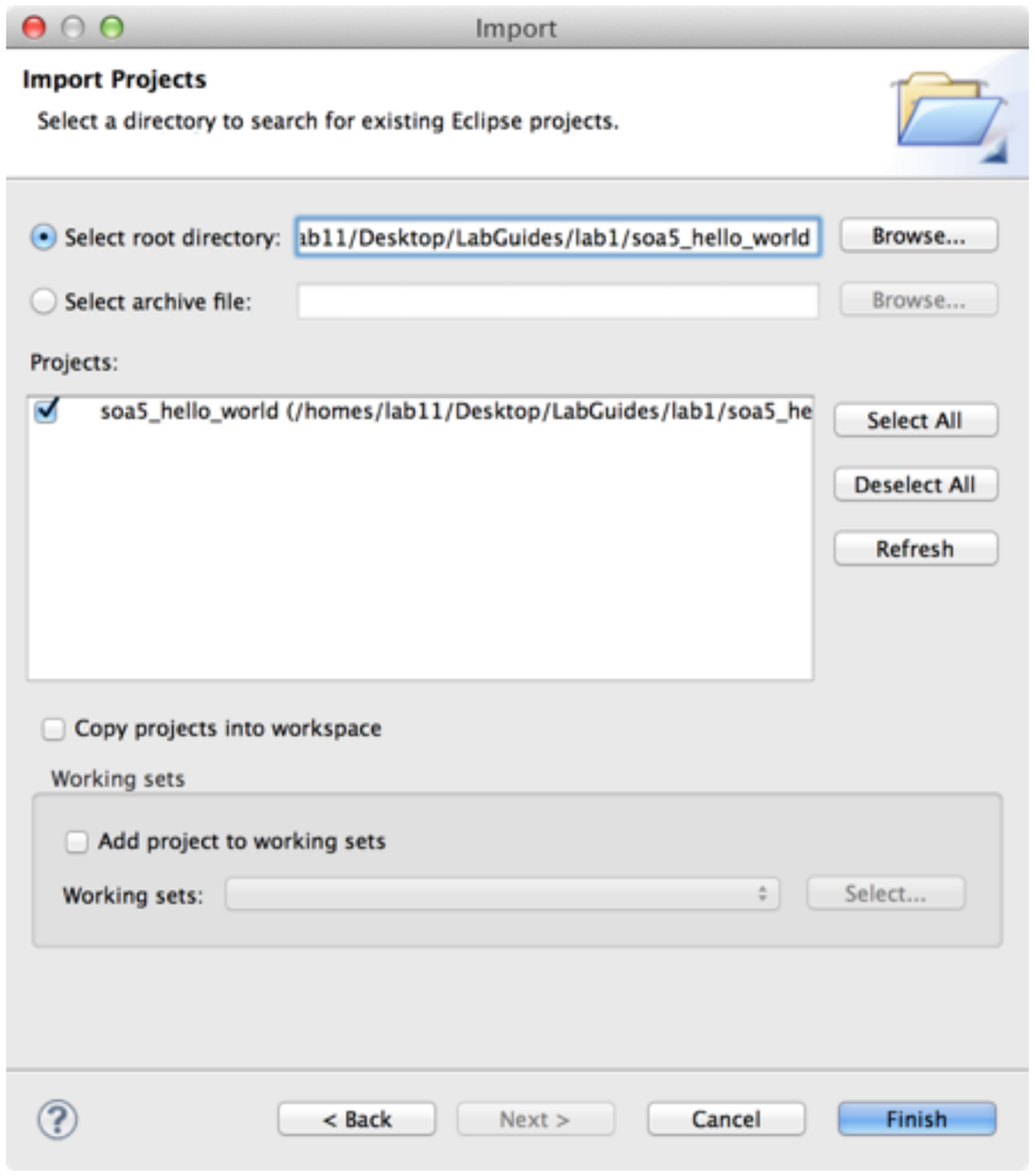
TODO

1. Click Browse ... and navigate to:

`/home/lab11/Desktop/LabGuides/lab1/soa5_hello_world`

2. Make sure the soa5_hello_world project is checked

3. Click Finish

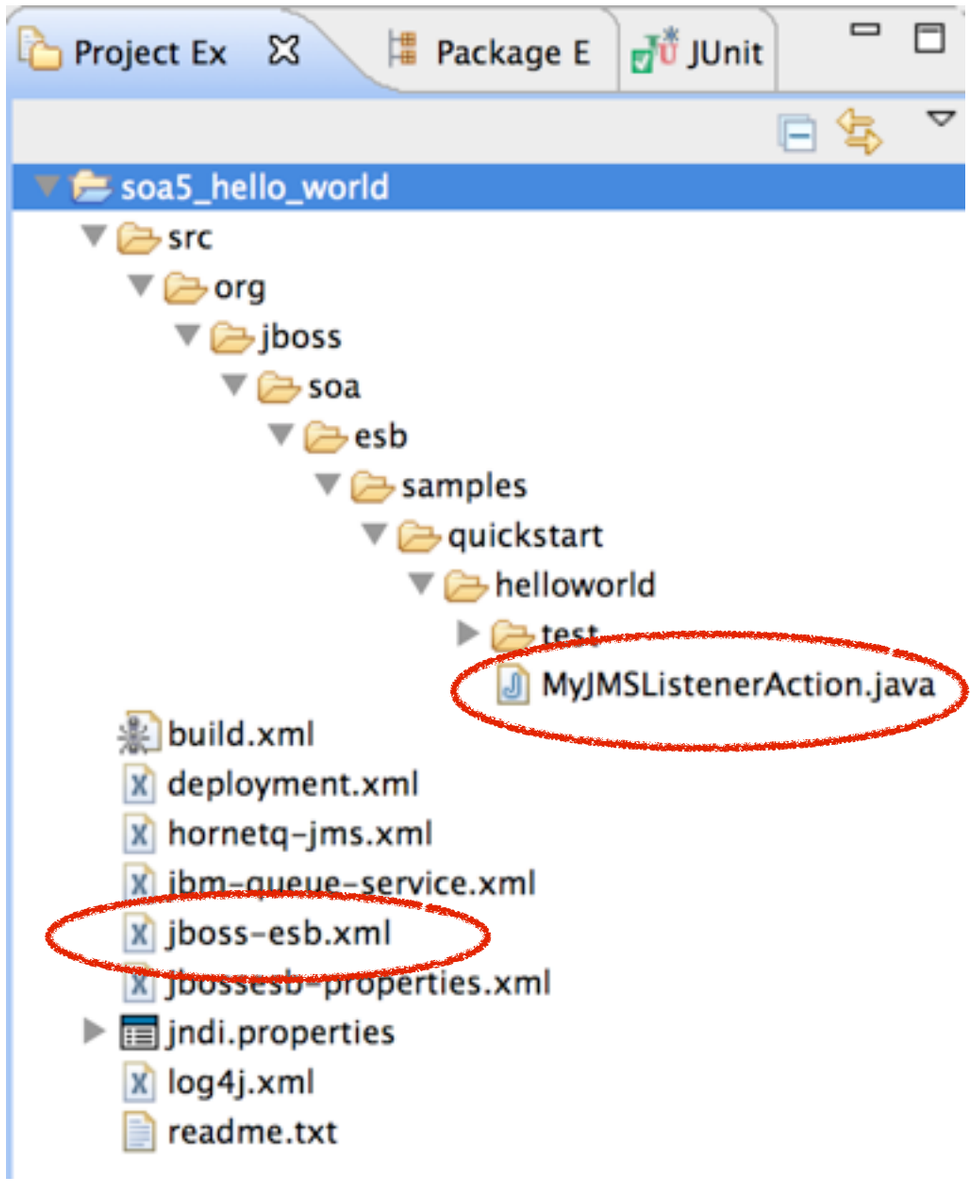


Files To Note

TODO

1. `jboss-esb.xml` contains the project's service definitions and configuration. Open the file by double-clicking on it in the Project Explorer.

2. `MyJMSListenerAction.java` contains the service logic for the application. Open the file by double-clicking on it in the Project Explorer.



jboss-esb.xml

jboss-esb.xml

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2  <jbossesb xmlns="http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/etc/schemas
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://anonsvn.labs.jboss.com/labs/jbossesb/trunk/product/etc/
5      parameterReloadSecs="5">
6
7      <providers>
8            <jms-provider name="JBossMQ" connection-factory="ConnectionFactory">
9                <jms-bus busid="quickstartGwChannel">
10                   <jms-message-filter
11                       dest-type="QUEUE"
12                       dest-name="queue/quickstart_helloworld_Request_gw"
13                    />
14               </jms-bus>
15               <jms-bus busid="quickstartEsbChannel">
16                   <jms-message-filter
17                       dest-type="QUEUE"
18                       dest-name="queue/quickstart_helloworld_Request_esb"
19                   />
20               </jms-bus>
21
22           </jms-provider>
23     </providers>
24
25     <services>
26       <service
27           category="FirstServiceESB"
28           name="SimpleListener"
```

Design Source

MyJMSListenerAction.java

```
package org.jboss.soa.esb.samples.quickstart.helloworld;

import org.jboss.soa.esb.actions.AbstractActionLifecycle;

public class MyJMSListenerAction extends AbstractActionLifecycle
{

    protected ConfigTree  _config;

    public MyJMSListenerAction(ConfigTree config) { _config = config; }


    public Message displayMessage(Message message) throws Exception{

            System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&");
            System.out.println("Body: " + message.getBody().get()) ;
            System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&");
            return message;

      }


}
```

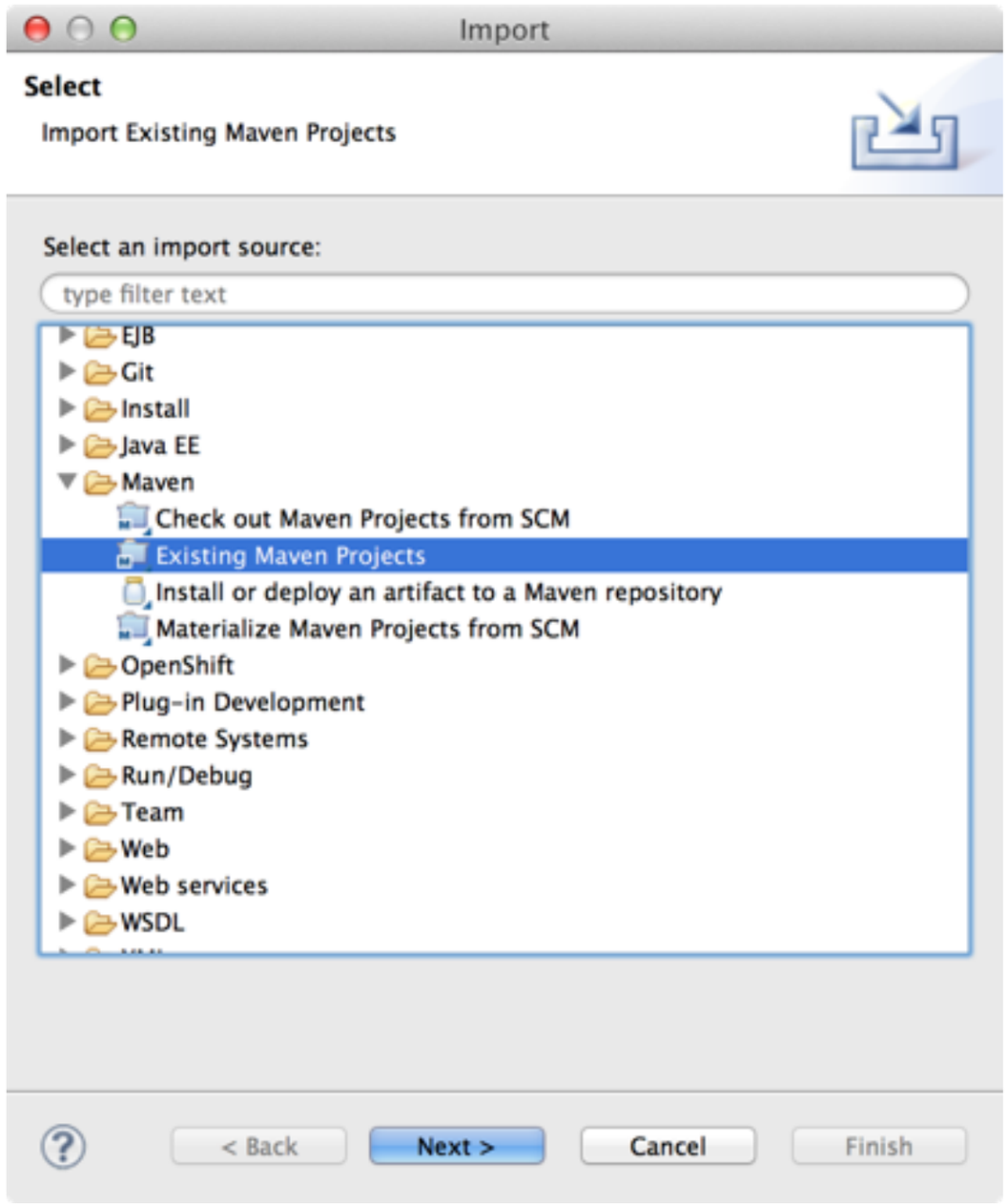
Importing SOA 6 Hello World

TODO

1. File -> Import ... from the JBDS menu.

2. Select Maven -> Existing Maven Projects

3. Click Next



Importing SOA 6 Hello World

TODO

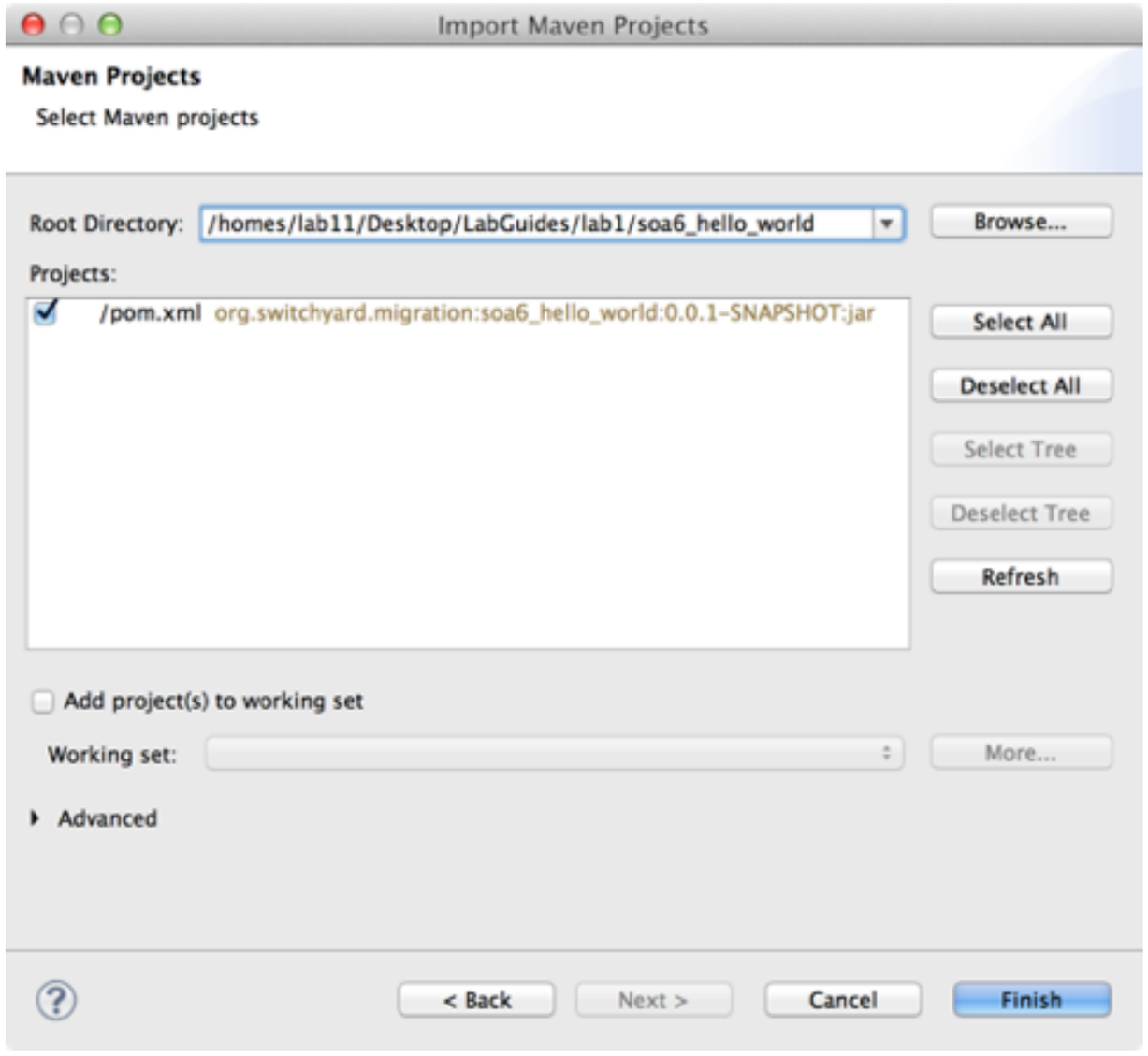
1. Click Browse ... and navigate to:

`/home/lab11/Desktop/LabGuides/lab1/soa6_hello_world`

2. Make sure the pom.xml is checked for:

`org.switchyard.migration:soa6_hello_world`

3. Click Finish



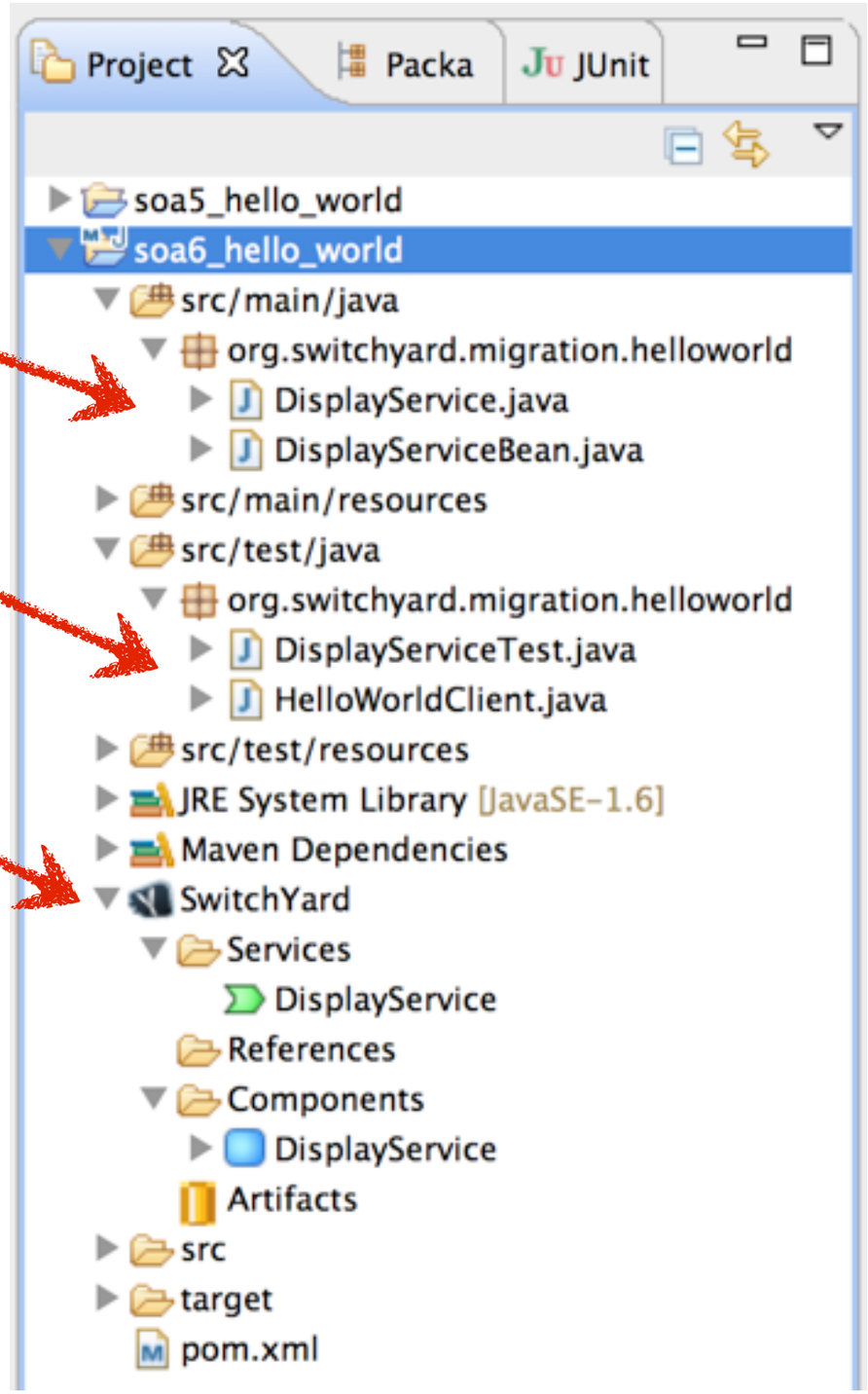
Files To Note

1. Java implementation classes for your application are stored in src/main/java

2. Test classes are stored in src/test/java

3. The SwitchYard node in the Project Explorer can be used to open the visual editor for the SwitchYard Project. Try double-clicking the SwitchYard node to open the editor.

Open these files and familiarize yourself with the content



Migrating `hello_world`

- A step-by-step migration walkthrough follows
- Each step is driven by a notification in the windup report
- Read the notification and microsite details for each step
- Study the before and after for each step using the applications you've imported into JBDS

Step 1

Converting Action Class to Bean Service

Notification

- Action : service binding configuration in jms-bus: busid="quickstartGwChannel"
- Action : service binding configuration in jms-bus: busid="quickstartEsbChannel"
- Action : composite service required for service: name="SimpleListener"
- Action : composite service binding required for listener: name="JMS-Gateway"
- Action : create component service for action processing pipeline
- Action : convert action class: class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"

Notifcation

`jboss-esb.xml`

FYI

This is the notification from jboss-esb.xml, noting that a custom action class is referenced in the action processing pipeline that needs to be migrated.

```
40.          <action name="action1"
41.             class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"
42.             process="displayMessage"
43.          />
```

Action : convert action class: class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"

Custom action classes should be migrated to CDI Beans in SOA 6. These beans can be defined as services or called directly from a Camel route.

For additional information and tips, see the action class microsite.

Notifcation

`MyJMSListenerAction.java`

FYI

This is the notification from MyJMSListenerAction.java providing API level details on what needs to change in the action class implementation.

27. **public class** MyJMSListenerAction **extends** AbstractActionLifecycle

Extending class 'org.jboss.soa.esb.actions.AbstractActionLifecycle' at line 27

Action classes in SOA 5 are simply CDI Beans in SOA 6. The extension of AbstractionActionLifecycle is no longer necessary. An action class can become a standalone service in SOA 6 or it can be invoked as a bean from a Camel route.

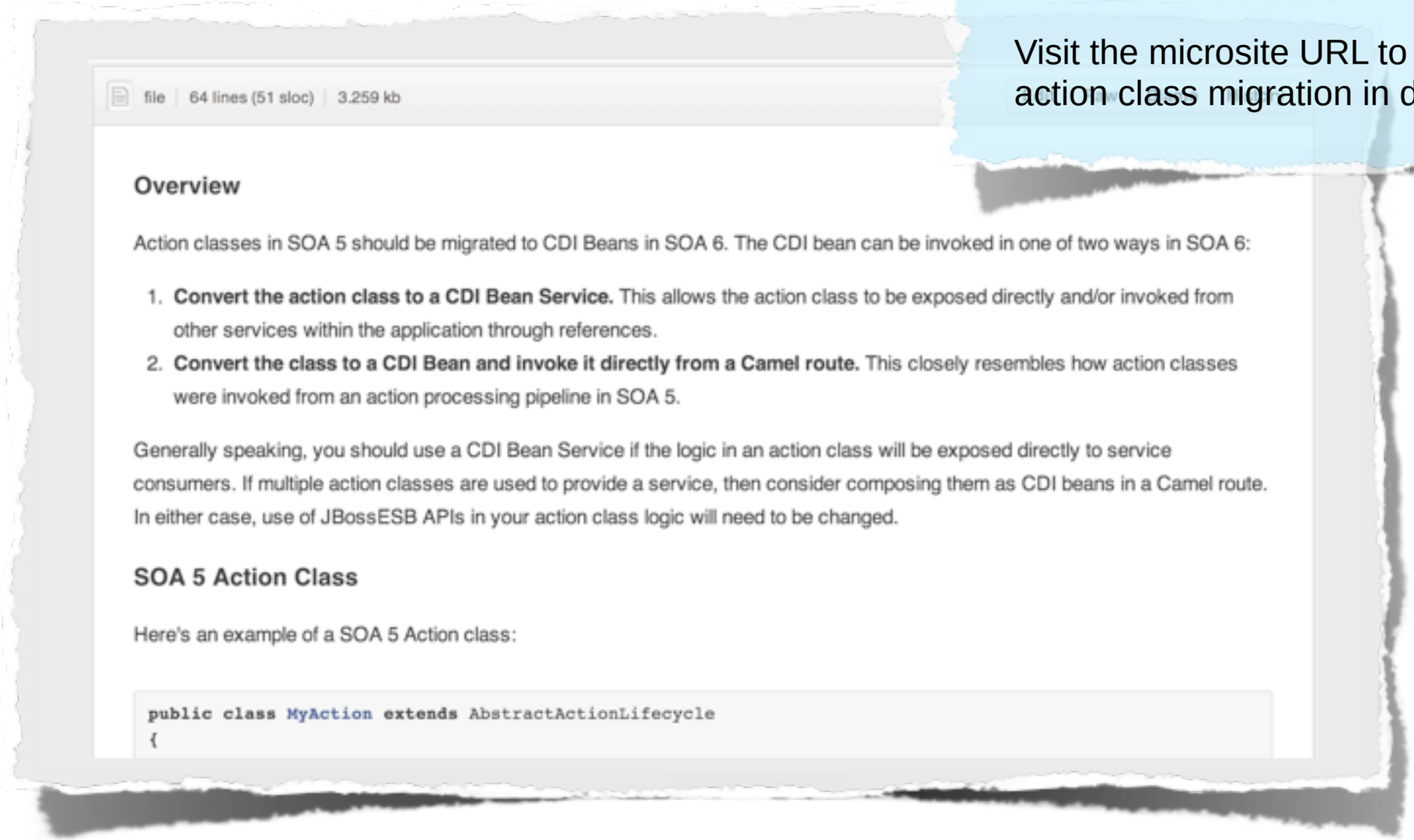
For additional information and tips, see the action class microsite.

Microsite

action class

TODO

Visit the microsite URL to learn about action class migration in detail.



file | 64 lines (51 sloc) | 3.259 kb

Overview

Action classes in SOA 5 should be migrated to CDI Beans in SOA 6. The CDI bean can be invoked in one of two ways in SOA 6:

1. **Convert the action class to a CDI Bean Service.** This allows the action class to be exposed directly and/or invoked from other services within the application through references.
2. **Convert the class to a CDI Bean and invoke it directly from a Camel route.** This closely resembles how action classes were invoked from an action processing pipeline in SOA 5.

Generally speaking, you should use a CDI Bean Service if the logic in an action class will be exposed directly to service consumers. If multiple action classes are used to provide a service, then consider composing them as CDI beans in a Camel route. In either case, use of JBossESB APIs in your action class logic will need to be changed.

SOA 5 Action Class

Here's an example of a SOA 5 Action class:

```
public class MyAction extends AbstractActionLifecycle
{
```

https://github.com/windup/soa-migration/blob/master/advice/action-class-migration.md

Service Contract

FYI

This is the service contract for DisplayService in SOA 6. All services in SOA 6 have a contract.

```
package org.switchyard.migration.helloworld;

public interface DisplayService {

    void display(String message);

}
```

Service Implementation

FYI

This is the implementation of DisplayService in SOA 6 using a CDI Bean Service.

```
package org.switchyard.migration.helloworld;

import org.switchyard.component.bean.Service;

@Service(DisplayService.class)
public class DisplayServiceBean implements DisplayService {

    @Override
    public void display(String message) {
        System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&");
        System.out.println("Body: " + message) ;
        System.out.println("&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&");
    }

}
```

Step 2

Converting the Action Processing Pipeline

Notification

- Action : service binding configuration in jms-bus: busid="quickstartGwChannel"
- Action : service binding configuration in jms-bus: busid="quickstartEsbChannel"
- Action : composite service required for service: name="SimpleListener"
- Action : composite service binding required for listener: name="JMS-Gateway"
- Action : create component service for action processing pipeline
- Action : convert action class: class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"

Action Processing Pipeline

```
<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>

   <!-- The next action is for Continuous Integration testing -->
  <action name="testStore" class="org.jboss.soa.esb.actions.TestMessageStore"/>

</actions>
```

Notifcation

`jboss-esb.xml`

39. `<actions mep="OneWay">`

Action : create component service for action processing pipeline

The logic and execution flow of a service in SOA 5 is defined in an action processing pipeline. In SOA 6, this logic is contained within a service component definition and expressed using any of the available implementation types in SwitchYard.

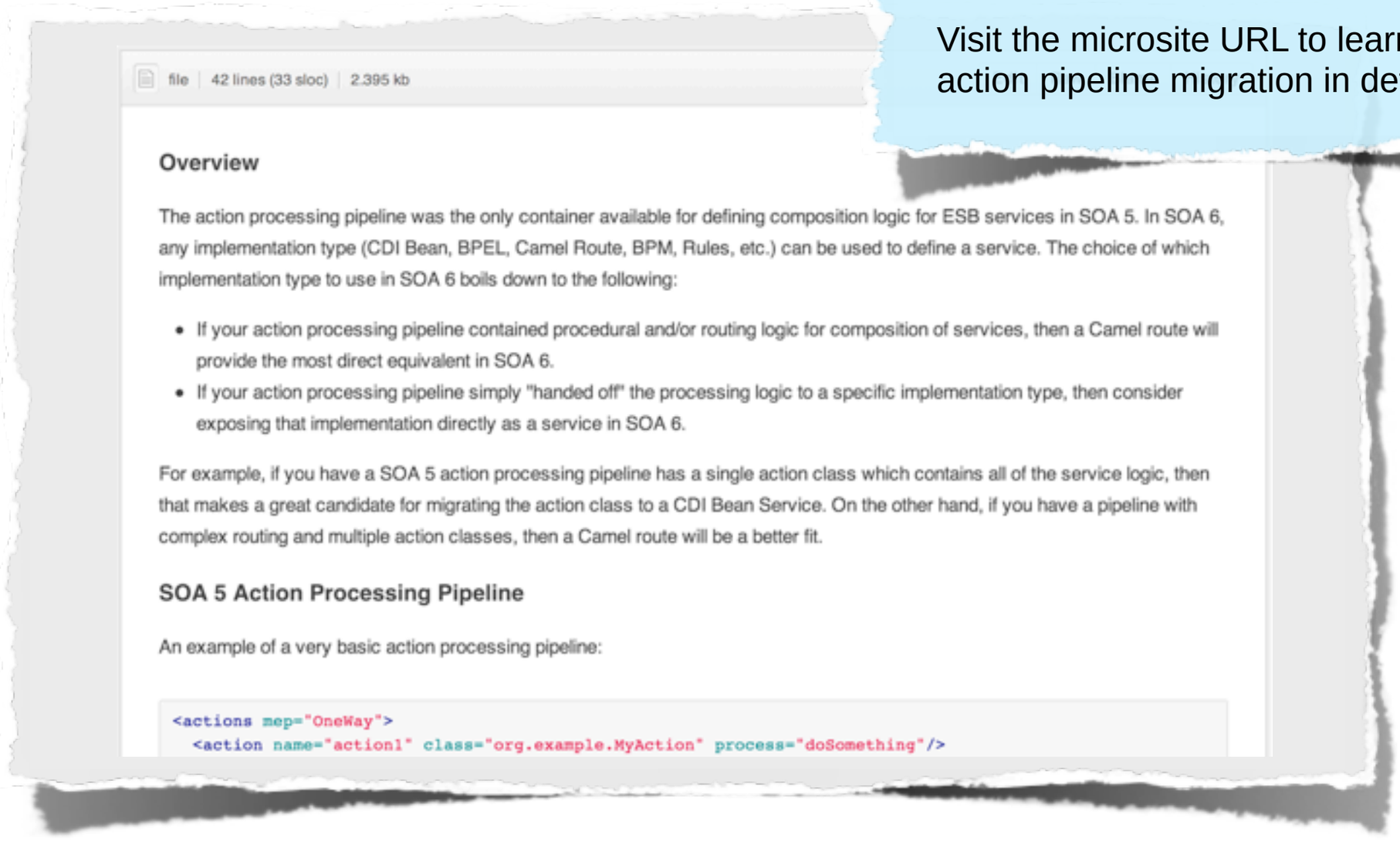
For additional information and tips, see the action pipline microsite.

Microsite

action pipeline

TODO

Visit the microsite URL to learn about action pipeline migration in detail.


file | 42 lines (33 sloc) | 2.395 kb

Overview

The action processing pipeline was the only container available for defining composition logic for ESB services in SOA 5. In SOA 6, any implementation type (CDI Bean, BPEL, Camel Route, BPM, Rules, etc.) can be used to define a service. The choice of which implementation type to use in SOA 6 boils down to the following:

- If your action processing pipeline contained procedural and/or routing logic for composition of services, then a Camel route will provide the most direct equivalent in SOA 6.
- If your action processing pipeline simply "handed off" the processing logic to a specific implementation type, then consider exposing that implementation directly as a service in SOA 6.

For example, if you have a SOA 5 action processing pipeline has a single action class which contains all of the service logic, then that makes a great candidate for migrating the action class to a CDI Bean Service. On the other hand, if you have a pipeline with complex routing and multiple action classes, then a Camel route will be a better fit.

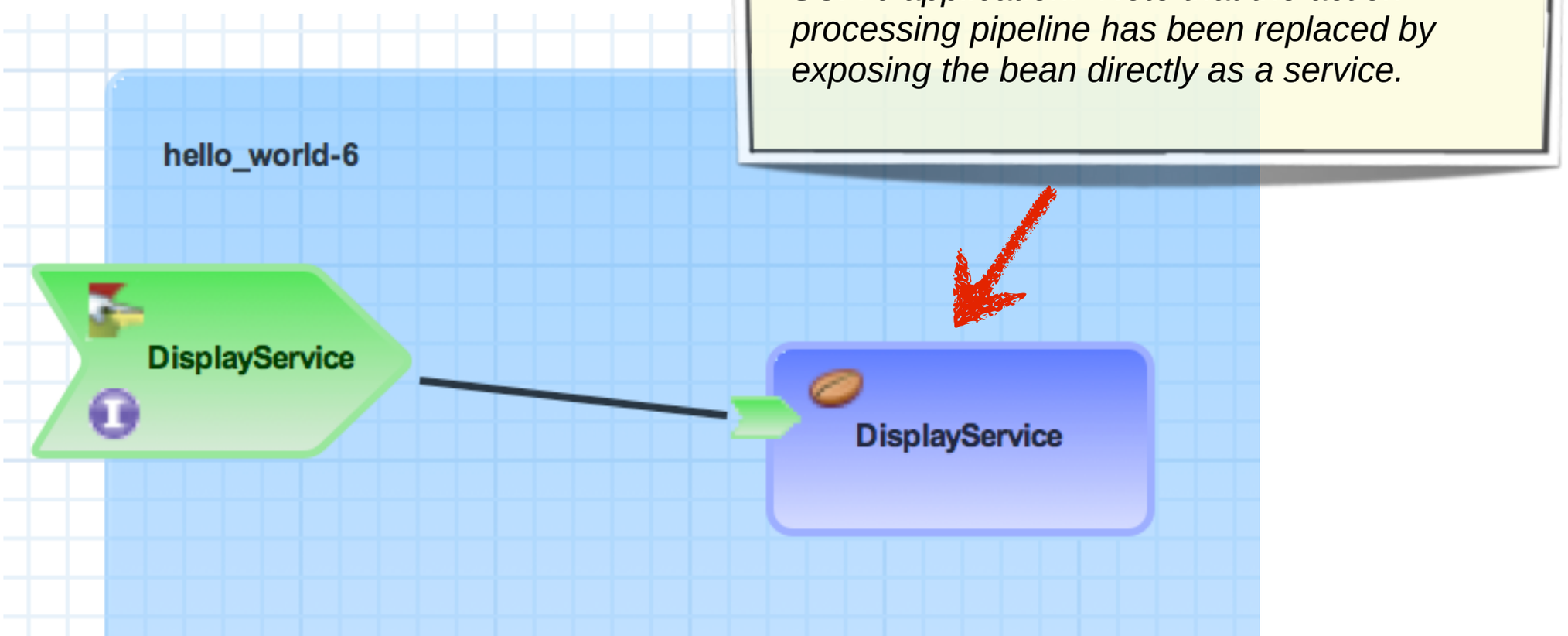
SOA 5 Action Processing Pipeline

An example of a very basic action processing pipeline:

```
<actions mep="OneWay">
  <action name="action1" class="org.example.MyAction" process="doSomething"/>
```

https://github.com/windup/soa-migration/blob/master/advice/action-pipeline-migration.md

Component Service



Step 3

Creating Composite Service For ESB Services

Notification

- Action : service binding configuration in jms-bus: busid="quickstartGwChannel"
- Action : service binding configuration in jms-bus: busid="quickstartEsbChannel"
- Action : composite service required for service: name="SimpleListener"
- Action : composite service binding required for listener: name="JMS-Gateway"
- Action : create component service for action processing pipeline
- Action : convert action class: class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"

Service Definition

```
25.         <services>
26.          <service
27.              category="FirstServiceESB"
28.              name="SimpleListener"
29.              description="Hello World">
```

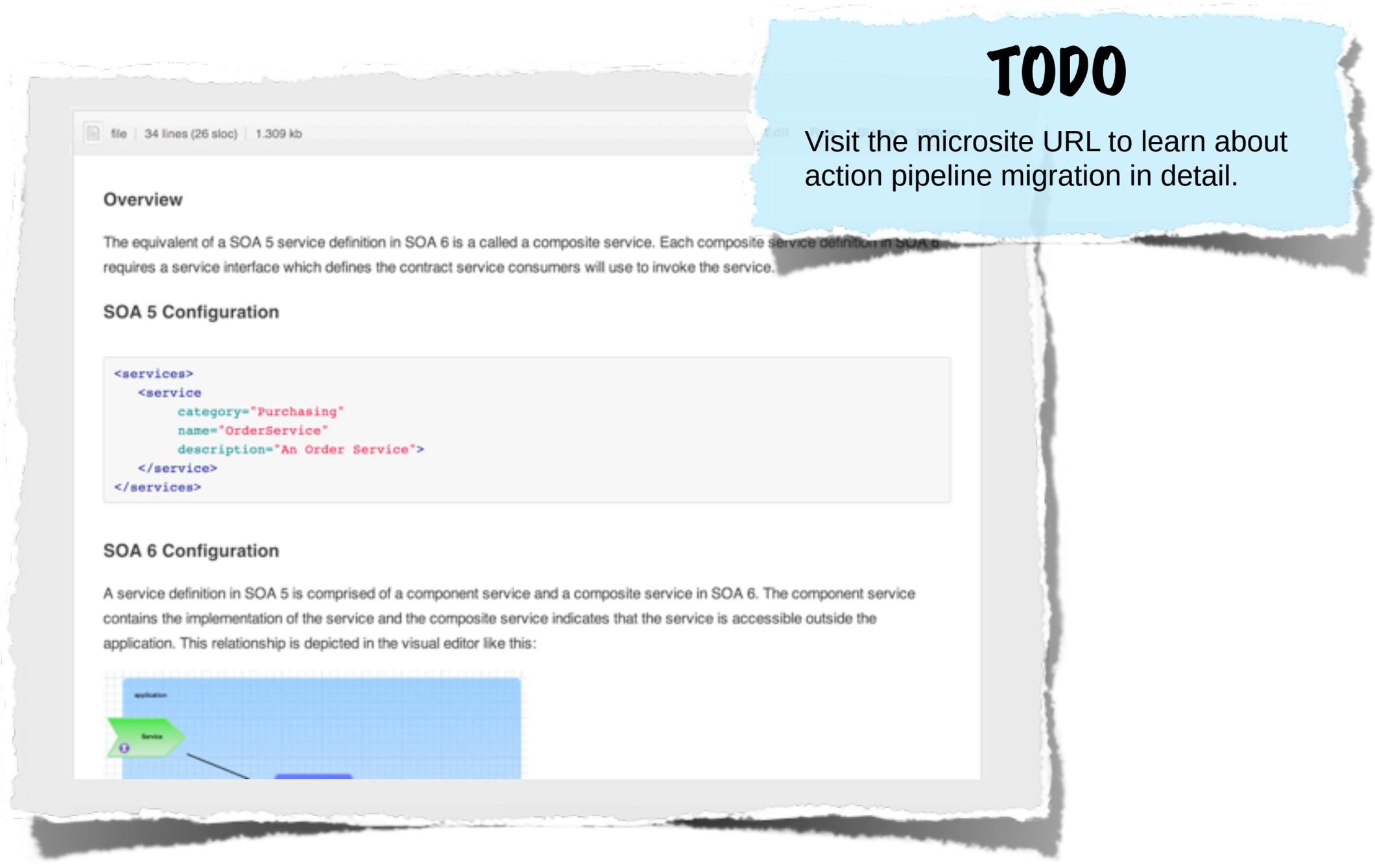
Action : composite service required for service: name="SimpleListener"

Each definition in SOA 5 represents a service which can be called from outside the application through an ESB listner. The equivalent definition in SOA 6 is a composite service.

For additional information and tips, see the service migration microsite.

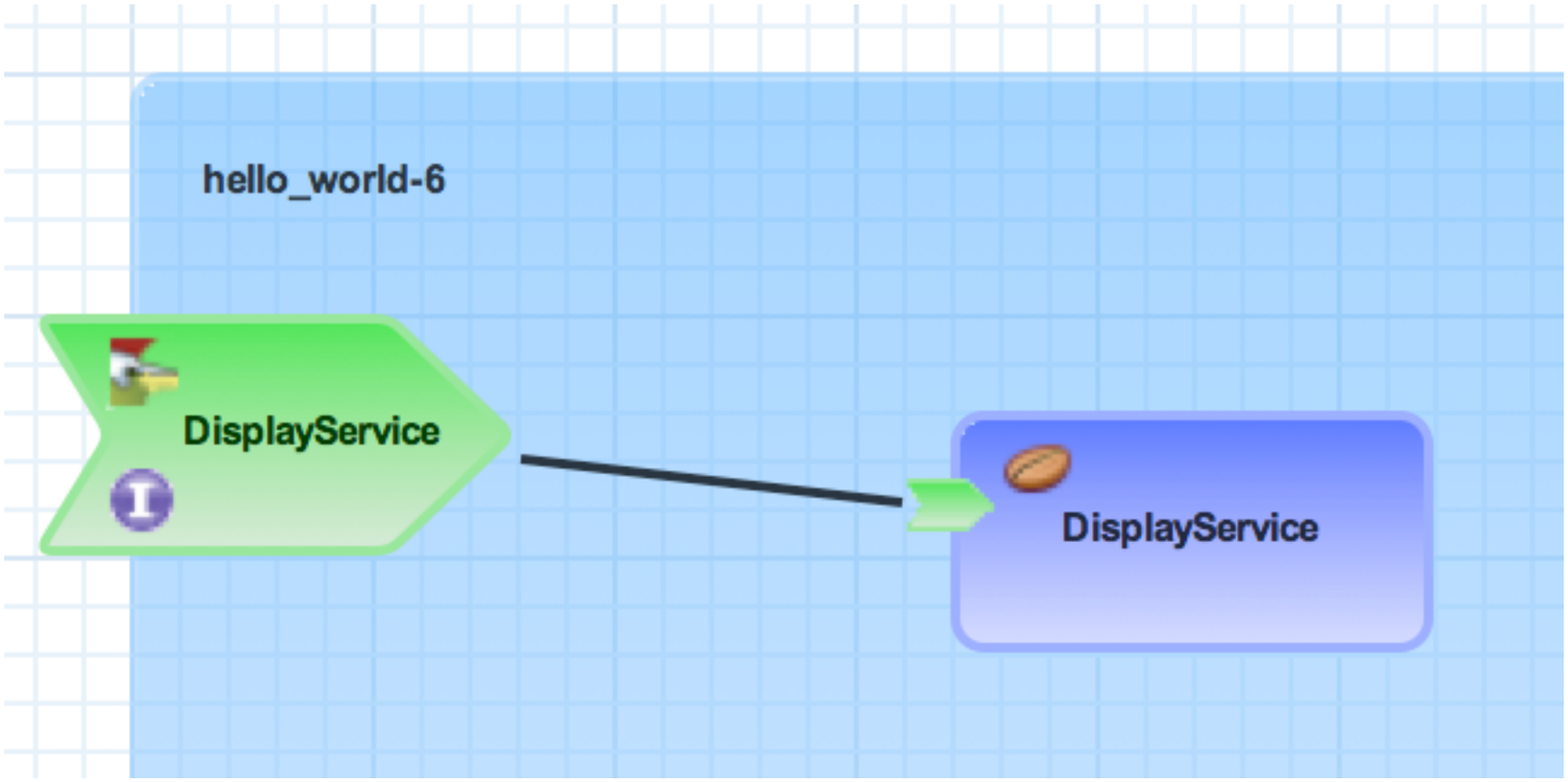
Microsite

service migration



https://github.com/windup/soa-migration/blob/master/advice/service-migration.md

Composite Service



Step 4

Migrating JMS Gateway Listener

Notification

- Action : service binding configuration in jms-bus: busid="quickstartGwChannel"
- Action : service binding configuration in jms-bus: busid="quickstartEsbChannel"
- Action : composite service required for service: name="SimpleListener"
- Action : composite service binding required for listener: name="JMS-Gateway"
- Action : create component service for action processing pipeline
- Action : convert action class: class="org.jboss.soa.esb.samples.quickstart.helloworld.MyJMSListenerAction"

Gateway Listener

```
30.          <listeners>
31.             <jms-listener name="JMS-Gateway"
32.                busidref="quickstartGwChannel"
33.                is-gateway="true"
34.          />
```

Action : composite service binding required for listener: name="JMS-Gateway"

This listener requires a composite service binding in SwitchYard. The configuration for a JCA or JMS binding can be found in the jms-bus definition associated with this listener.

For additional information and tips, see the gateway listener microsite.

Microsite

listener migration

TODO

Visit the microsite URL to learn about listener migration in detail.

file | 25 lines (22 sloc) | 1.122 kb

Overview

There are two types of listeners in SOA 5:

- ESB-aware Listeners are used for internal message dispatch and drive an action processing pipeline.
- Gateway Listeners are used to expose services to external consumers over a protocol binding.

When migrating to SOA 5, ESB-aware listeners are no longer required. Gateway listeners become Service Bindings in SOA 6. A gateway listeners have a value of true for the 'is-gateway' attribute on a listener definition in jboss-esb.xml.

Listener to Service Binding Conversion

The following table provides a mapping of SOA 5 gateway listeners to SOA 6 service bindings:

SOA 5 Listener	SOA 6 Service Binding
fs-listener	binding.file
ftp-listener	binding.ftp
sql-listener	binding.sql
jbr-listener	binding.tcp
jms-listener	binding.jca or binding.jms

https://github.com/windup/soa-migration/blob/master/advice/gateway-listener-migration.md

Gateway Listener Config

09. `<jms-bus busid="quickstartGwChannel">`

Action : service binding configuration in jms-bus: busid="quickstartGwChannel"

A jms-bus definition can be converted to a JMS or JCA gateway binding on a composite service in SwitchYard.

If the jms-bus configuration is used for a non-gateway listener, it does not need to be migrated to SOA 6.

For additional information and tips, see the jms-bus migration microsite.

```
10.         <jms-message-filter
11.            dest-type="QUEUE"
12.            dest-name="queue/quickstart_helloworld_Request_gw"
13.          />
14.       </jms-bus>
```

Microsite

jms-bus migration

TODO

Visit the microsite URL to learn about jms-bus migration in detail.

https://github.com/windup/soa-migration/blob/master/advice/jms-bus-migration.md

Service Bindings

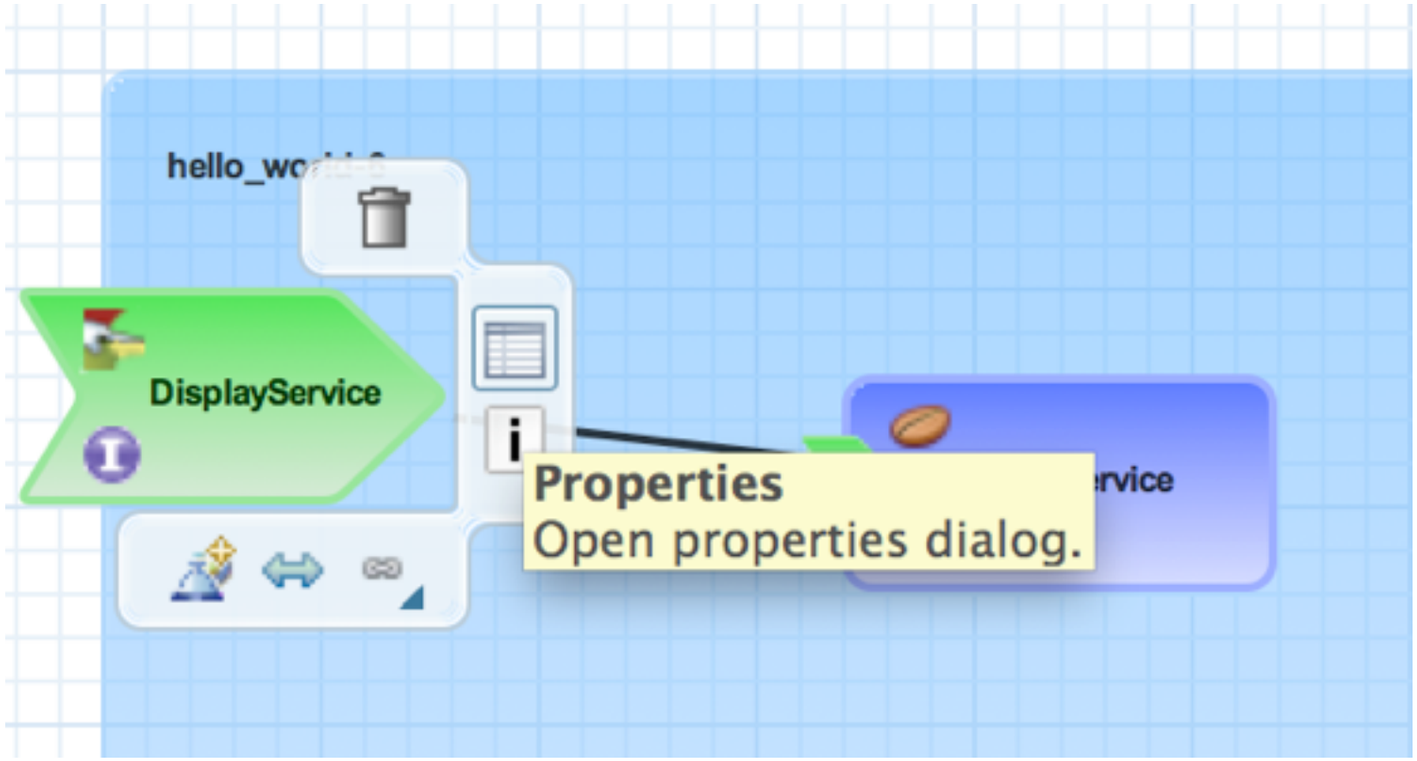
TODO

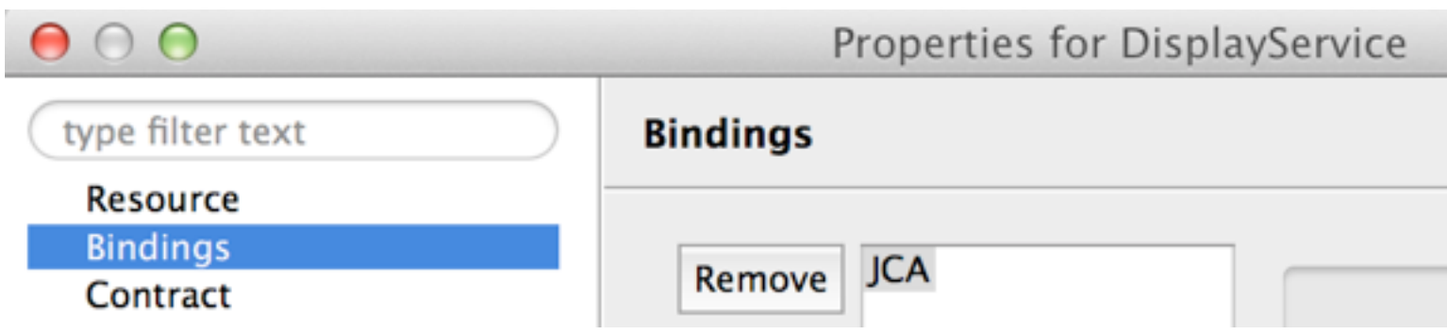
1. Bring up the button bar for the composite service ‘DisplayService’ by hovering over the large green service icon on the left of the composite.

2. When the button bar appears, click on the table icon which brings up the service properties.

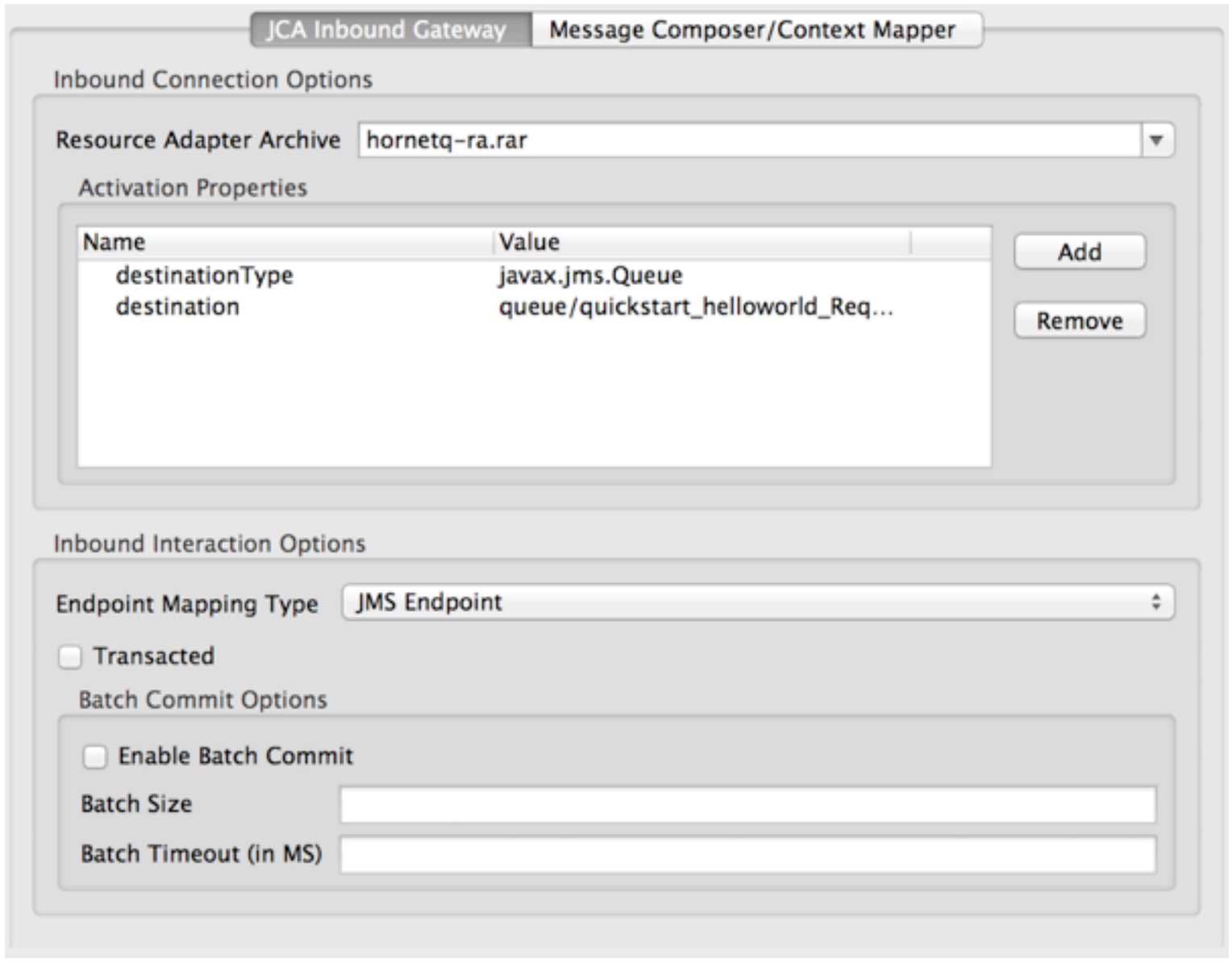
3. On the resulting property page, select the Bindings page in the left frame.

4. View the configuration details of the JCA binding definition in SOA 6.





Binding Configuration

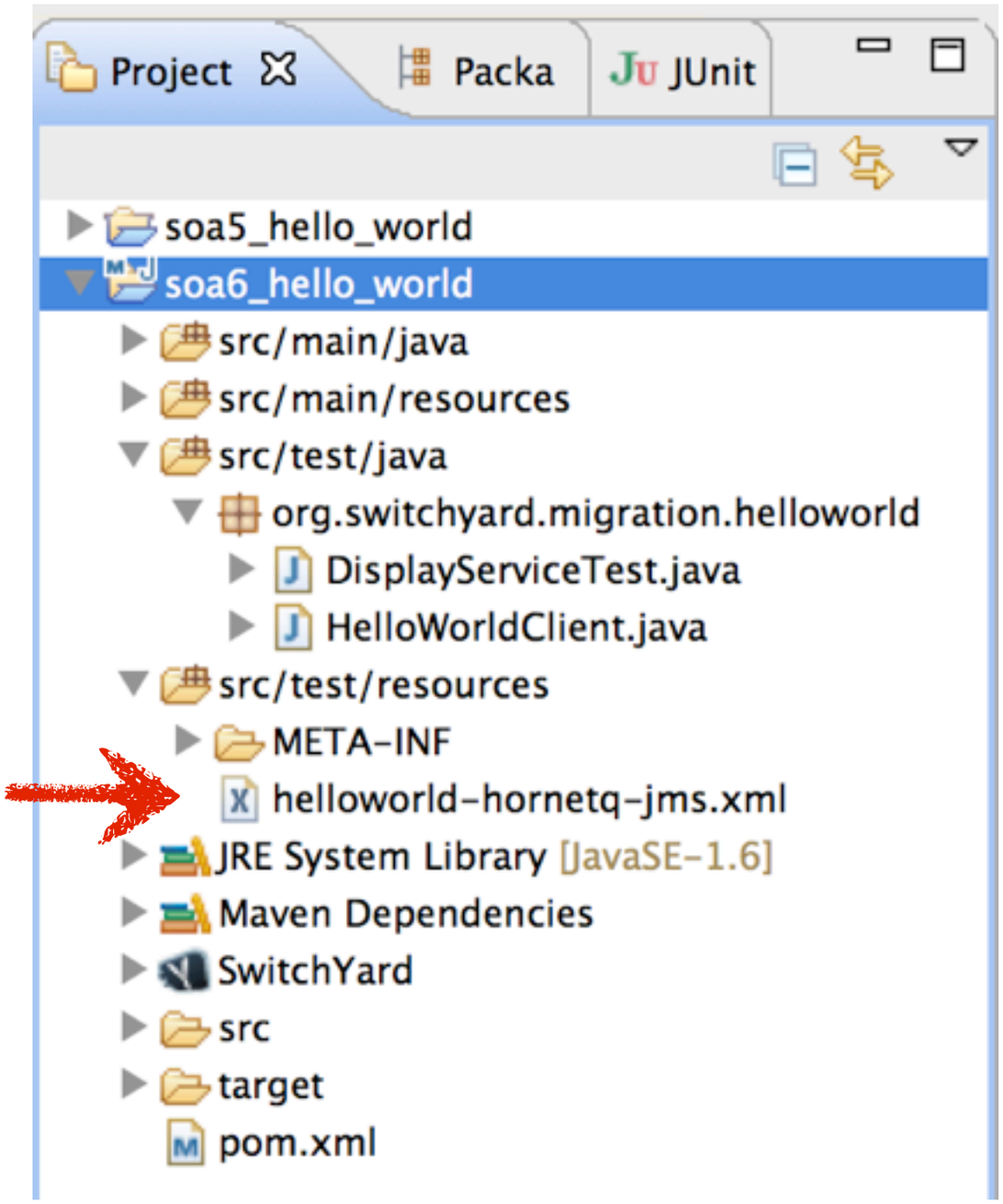


Deploy Queue

TODO

1. Right-click on helloworld-hornetq-jms.xml and select ‘Mark as Deployable’.

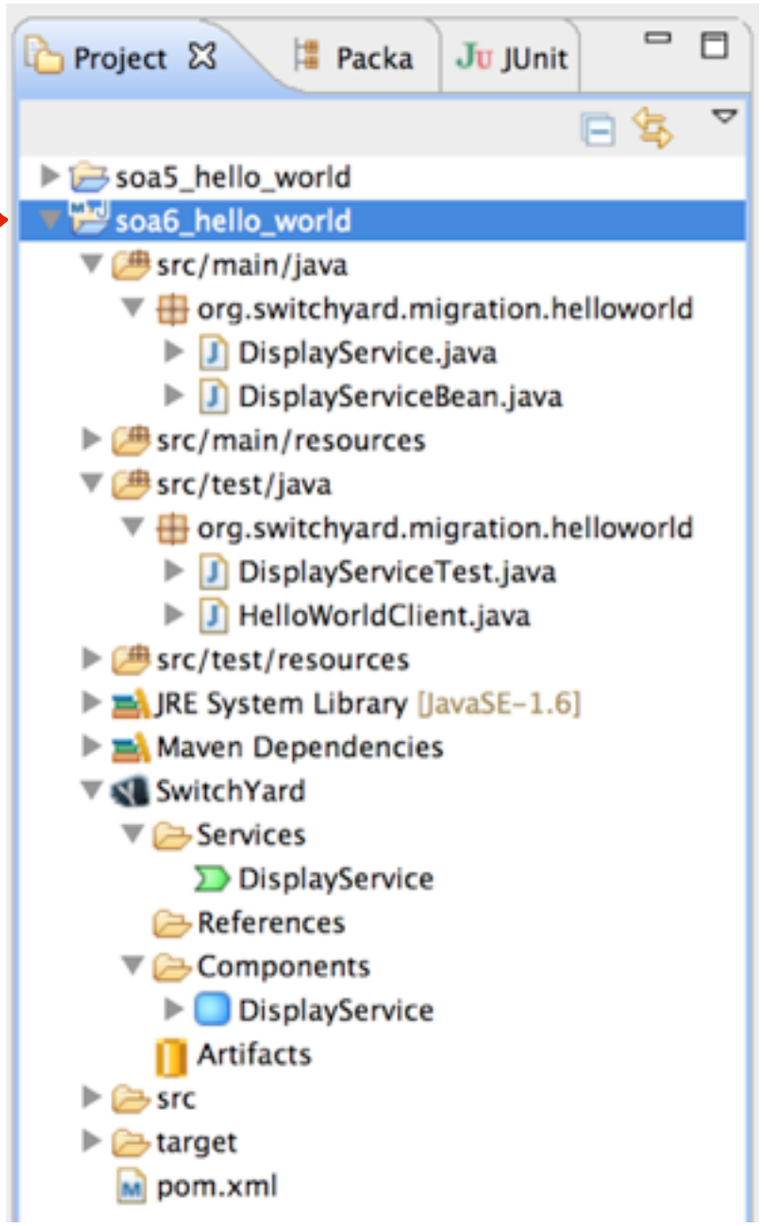
2. When prompted with “Really mark these as deployable?”, click OK.



Deploy Application

TODO

1. Right-click on the soa6_hello_world project and select Run As ... Run on Server

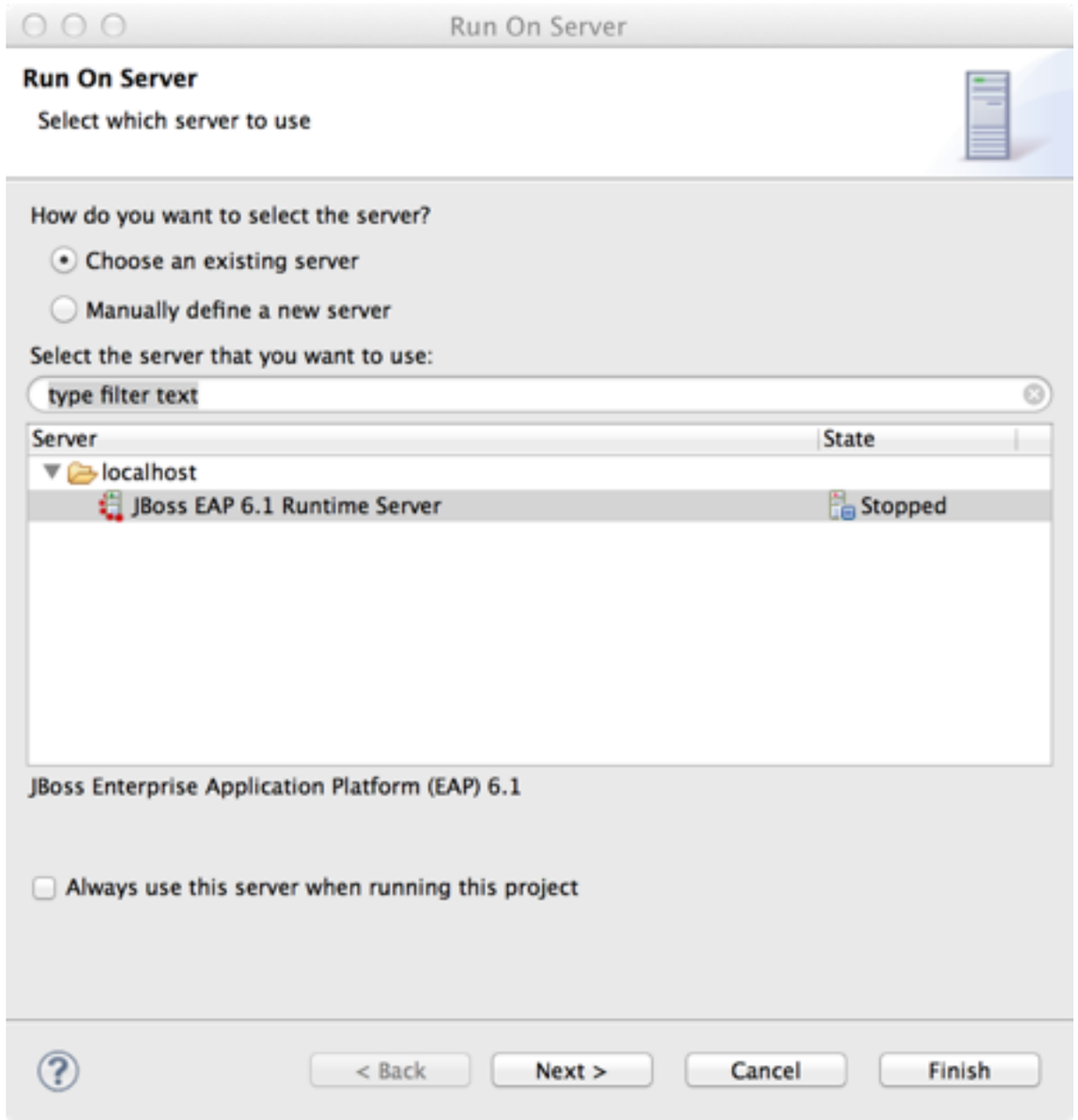


Select Server

TODO

1. Select the JBoss EAP 6.1 Runtime Server

2. Click Finish

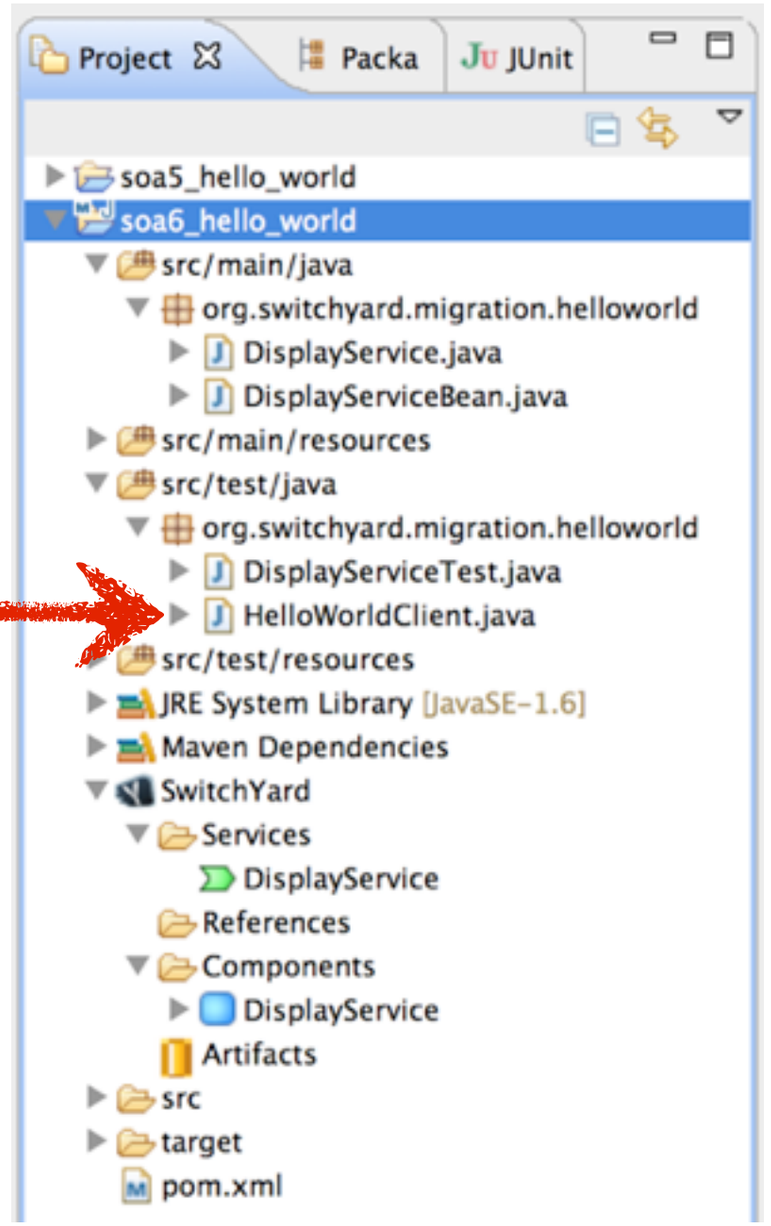


Run Test Client

TODO

1. Open HelloWorldClient.java from the Project Explorer view.

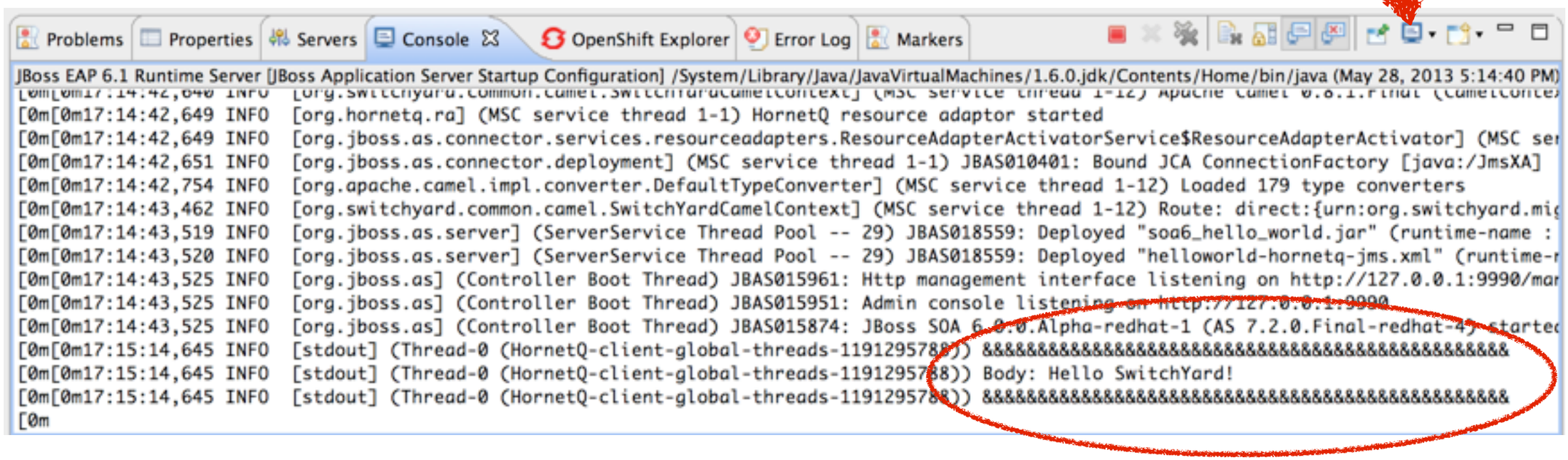
2. Go to the Run menu in the main menu bar and select ‘Run As -> Java Application’



Verify Output

TODO

1. Click here to swap between application output and server output.



Lab 1 Complete!