

Hello World SCA tutorial

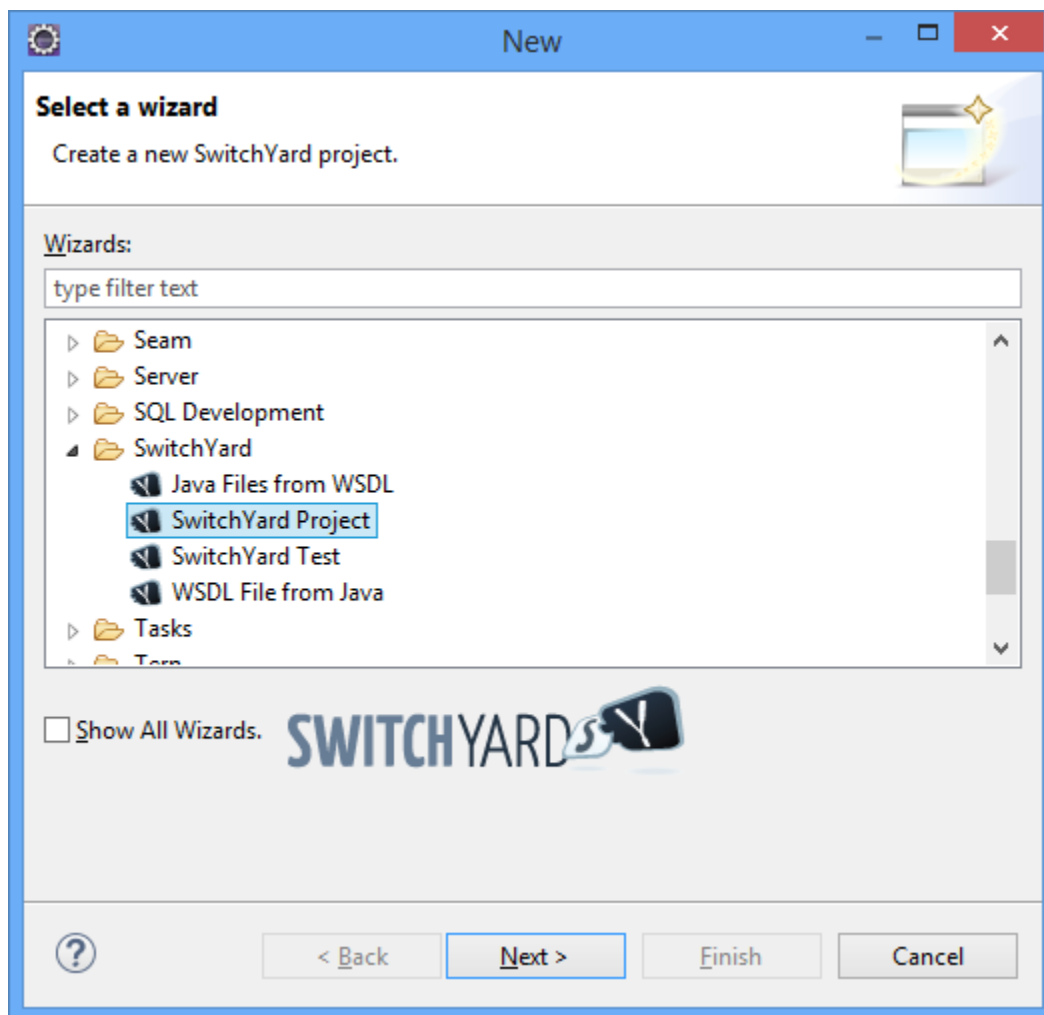
Balázs Simon (sbalazs@iit.bme.hu), BME IIT, 2015

1 Introduction

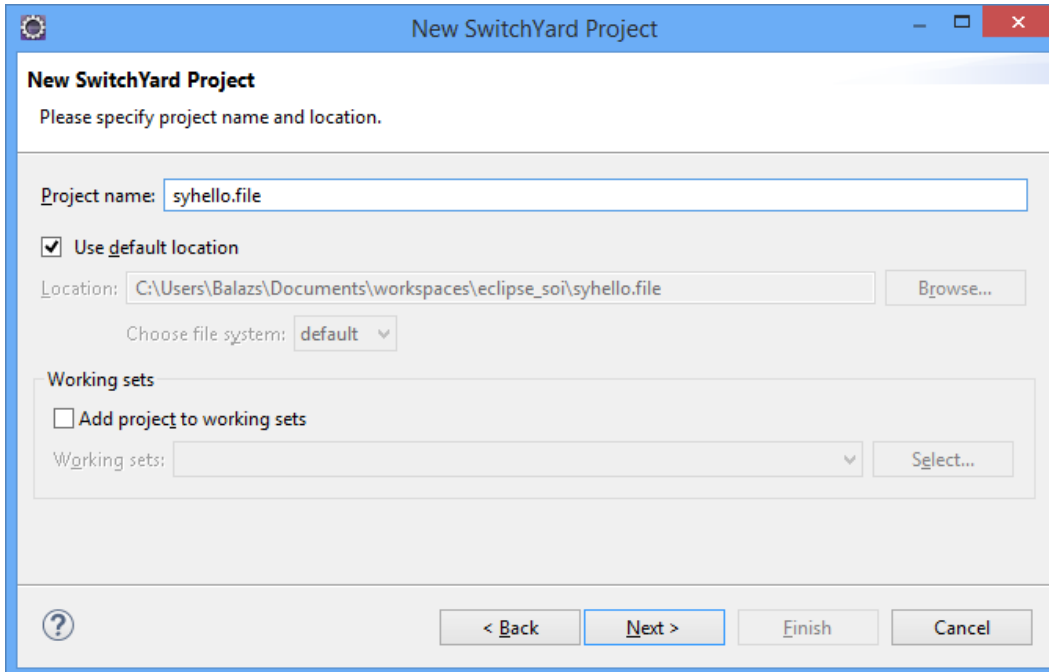
This document describes how to create Hello World SCA services in Eclipse using SwitchYard and WildFly.

2 Hello World file service

Create a new **SwitchYard Project** in Eclipse:

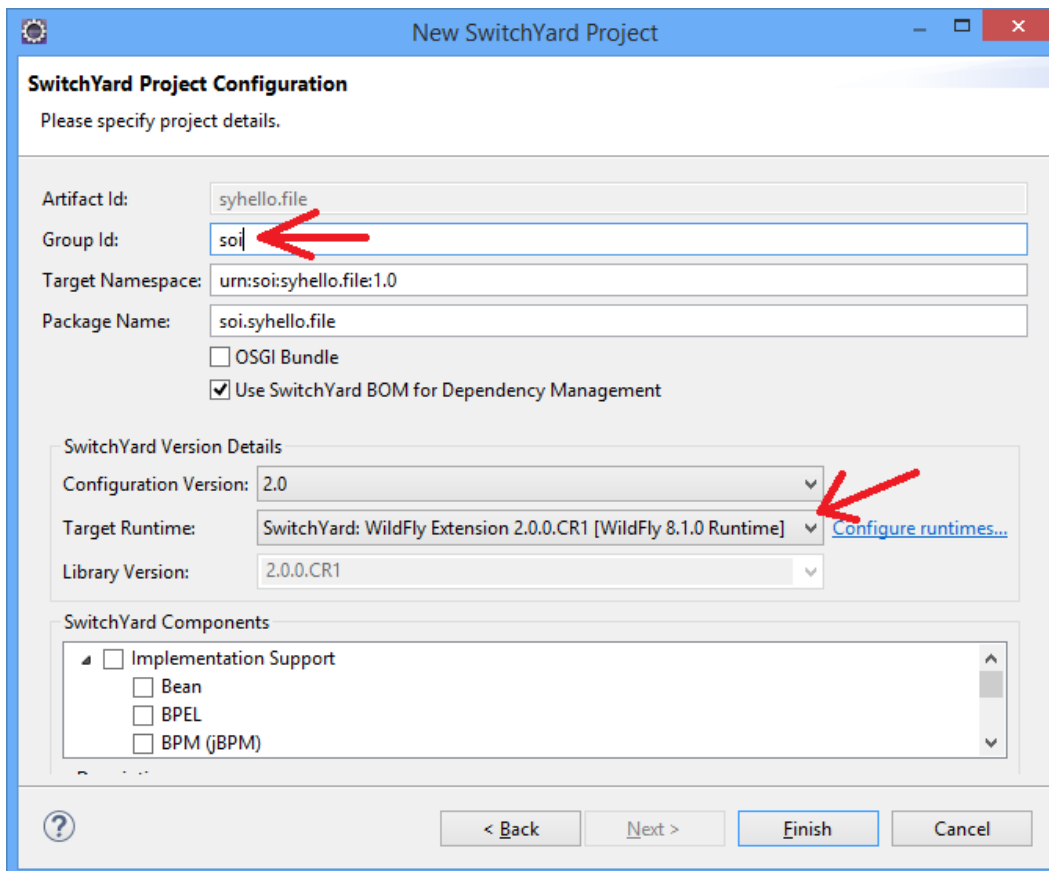


The project name should be **syhello.file**:



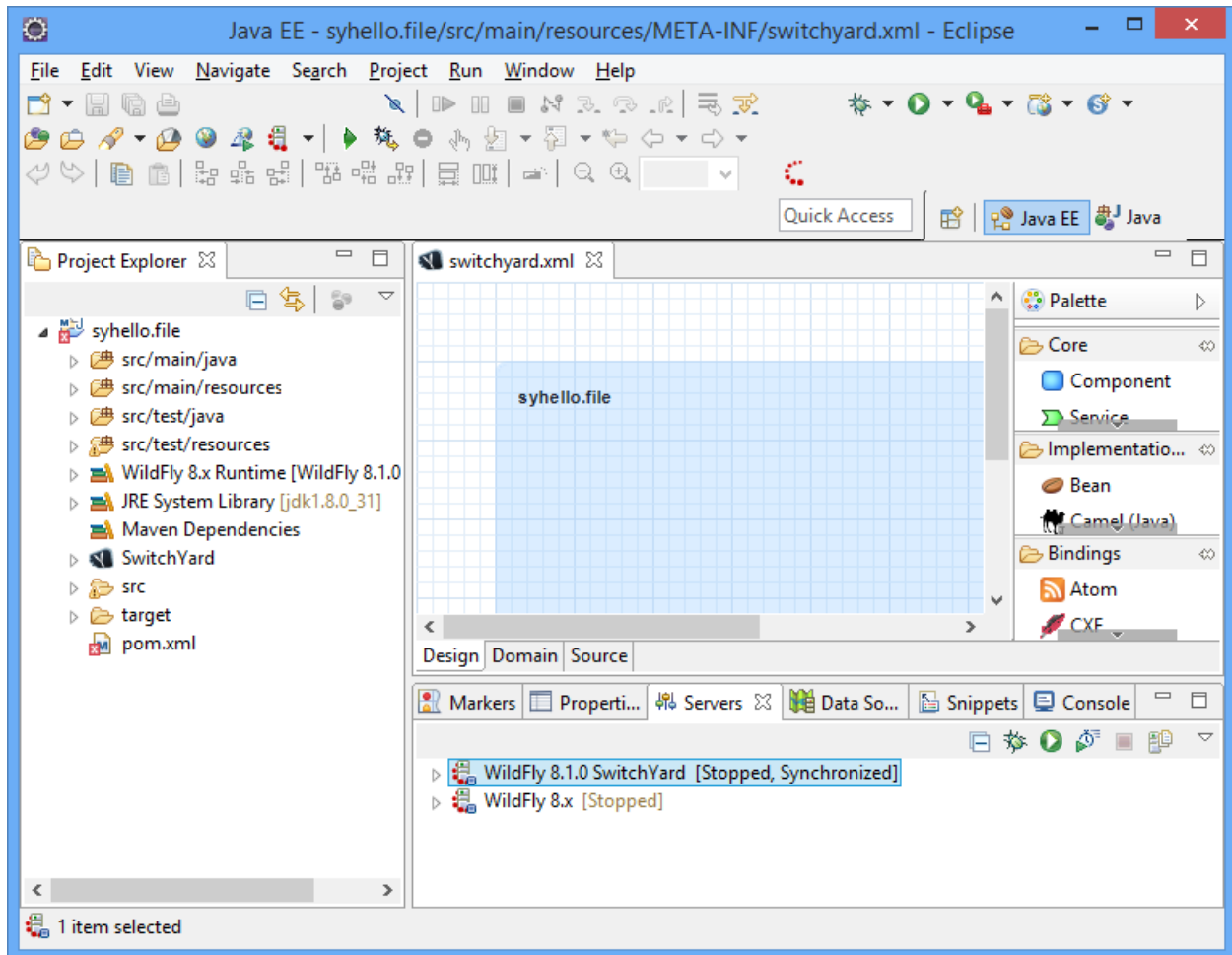
The 'New SwitchYard Project' dialog box is shown. It has a title bar with a gear icon and standard window controls. The main title is 'New SwitchYard Project' and the subtitle is 'Please specify project name and location.' The 'Project name' field contains 'syhello.file'. The 'Use default location' checkbox is checked. The 'Location' field shows 'C:\Users\Balazs\Documents\workspaces\eclipse_soi\syhello.file' with a 'Browse...' button. The 'Choose file system' dropdown is set to 'default'. The 'Working sets' section has an unchecked 'Add project to working sets' checkbox and a 'Working sets' dropdown with a 'Select...' button. At the bottom are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

Click **Next**. Change the **group id** to **soi** and make sure that the correct runtime is selected:



The 'SwitchYard Project Configuration' dialog box is shown. It has a title bar with a gear icon and standard window controls. The main title is 'SwitchYard Project Configuration' and the subtitle is 'Please specify project details.' The 'Artifact Id' field contains 'syhello.file'. The 'Group Id' field contains 'soi', with a red arrow pointing to it. The 'Target Namespace' field contains 'urn:soi:syhello.file:1.0'. The 'Package Name' field contains 'soi.syhello.file'. There are checkboxes for 'OSGI Bundle' (unchecked) and 'Use SwitchYard BOM for Dependency Management' (checked). The 'SwitchYard Version Details' section has a 'Configuration Version' dropdown set to '2.0', a 'Target Runtime' dropdown set to 'SwitchYard: WildFly Extension 2.0.0.CR1 [WildFly 8.1.0 Runtime]' with a red arrow pointing to it and a 'Configure runtimes...' link, and a 'Library Version' dropdown set to '2.0.0.CR1'. The 'SwitchYard Components' section has a tree view with 'Implementation Support' expanded, showing 'Bean', 'BPEL', and 'BPM (jBPM)' as sub-items. At the bottom are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

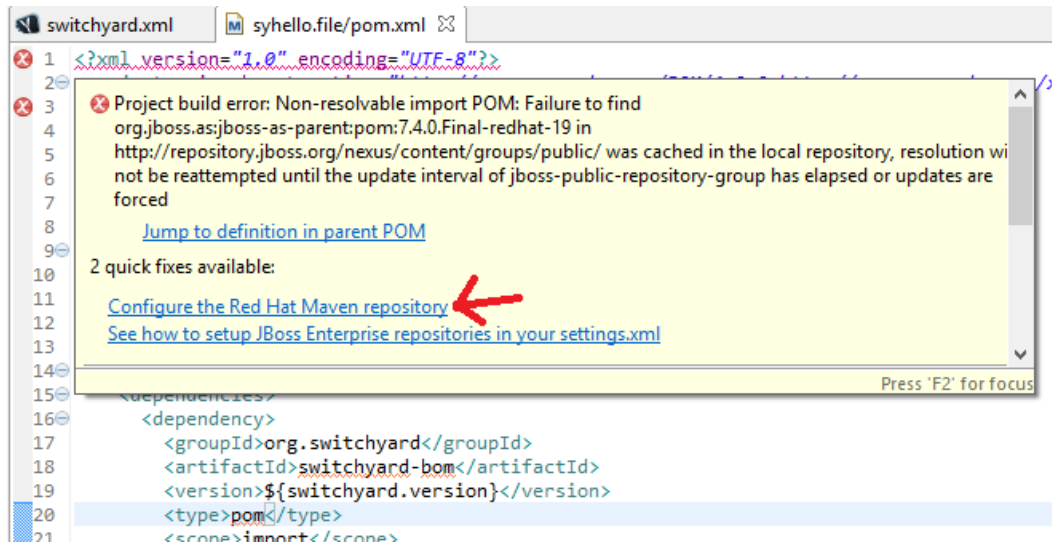
Click **Finish** and the project should look like this:



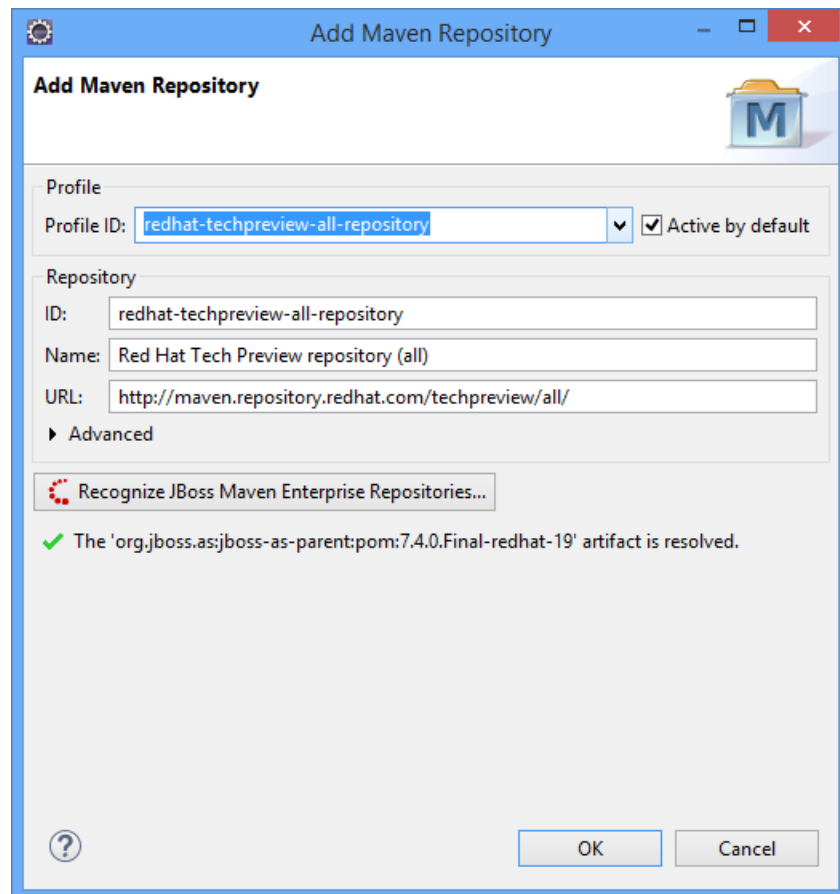
The **switchyard.xml** in center is the SCA editor. On the right is the Palette from which the SCA elements can be placed to the SCA editor.

The **pom.xml** may have a red cross indicating that Maven could not compile the project. The reason for this is that some dependencies are not in the central Maven repository. They, however, can be found in the Red Hat Maven repository.

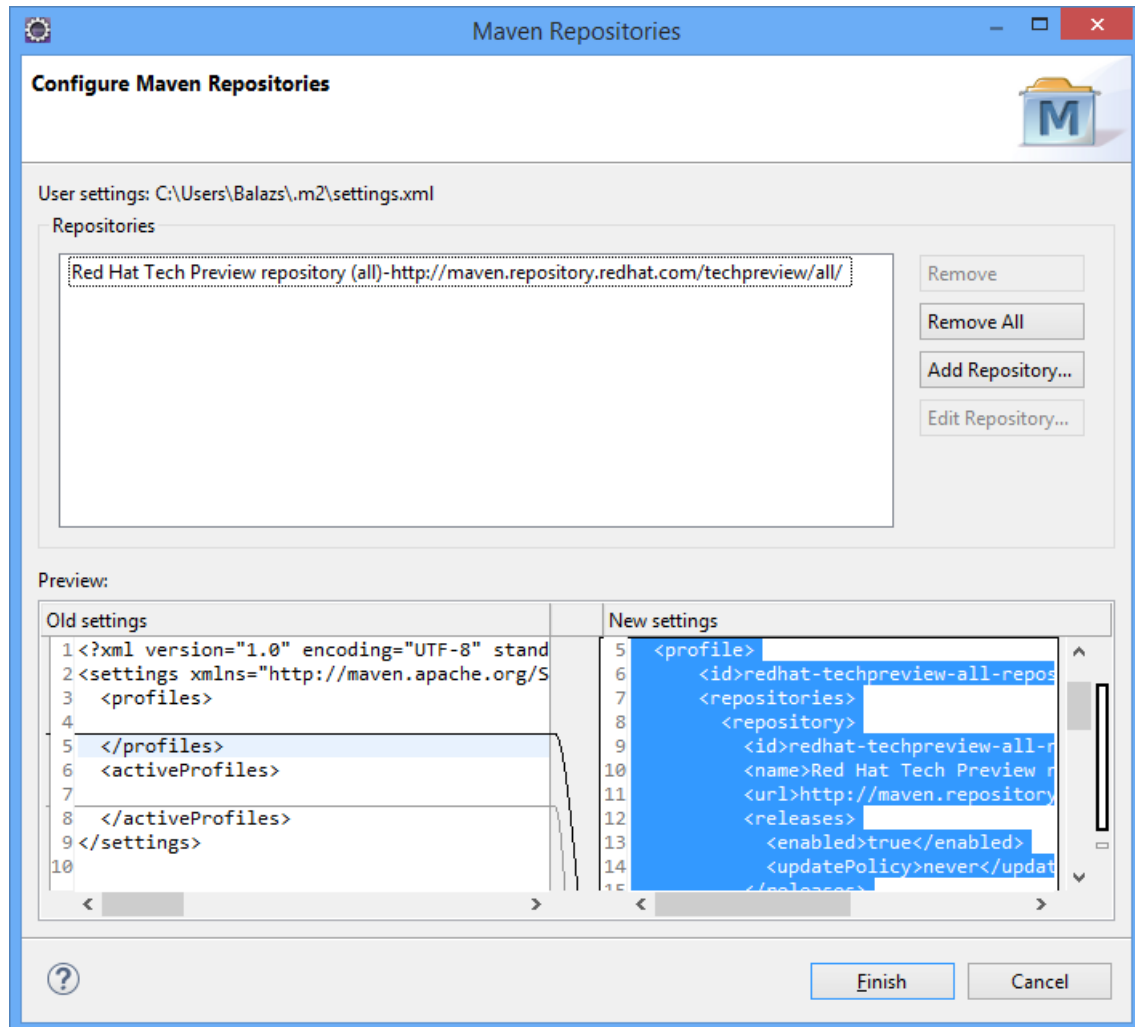
To fix the project, double click the **pom.xml** to open it and hover over the first error line with the mouse:



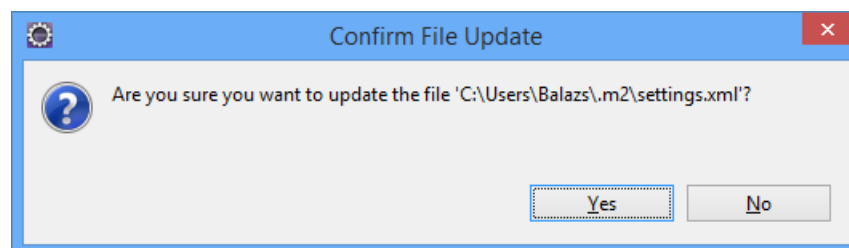
Click on **Configure the Red Hat Maven repository** and select the **redhat-techpreview-all-repository**:



Click **OK**, and in the next window click **Finish**:



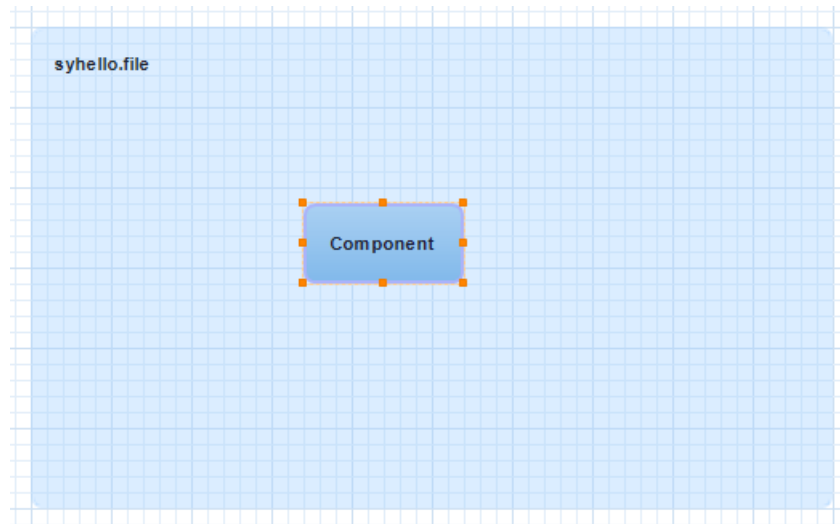
Click **Yes** in the confirmation dialog:



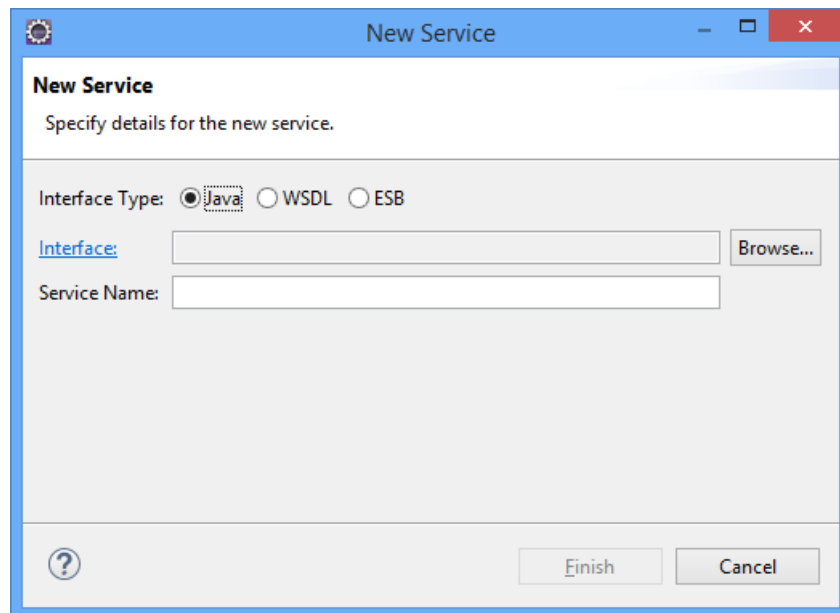
Maven will update and download the dependencies. It might take a while, so be patient.

When it is finished the compile problem should go away and this step will not have to be repeated for other SwitchYard projects anymore.

Create a new **Component** from the palette:

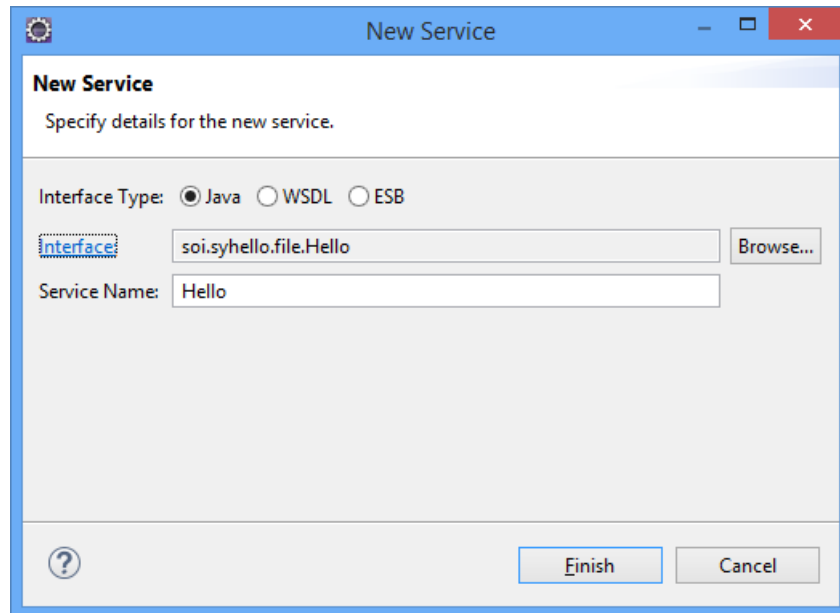


Create a new **Service** element for the component. When the service is added, the following dialog is shown:

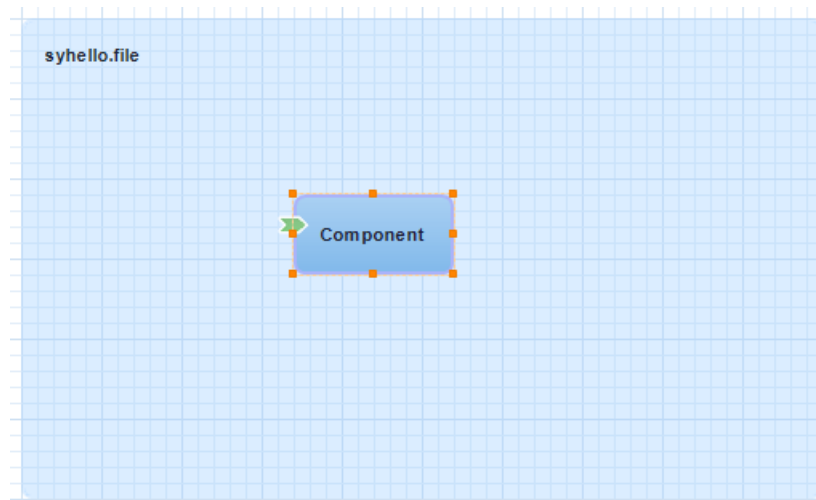


Select **Java** as the **interface type**. Click on the **Interface** link to create a new Java interface. The name of the interface should be **Hello**. Click **Finish** to close the Interface dialog.

In the **New Service** dialog the **Service Name** will also be **Hello**:

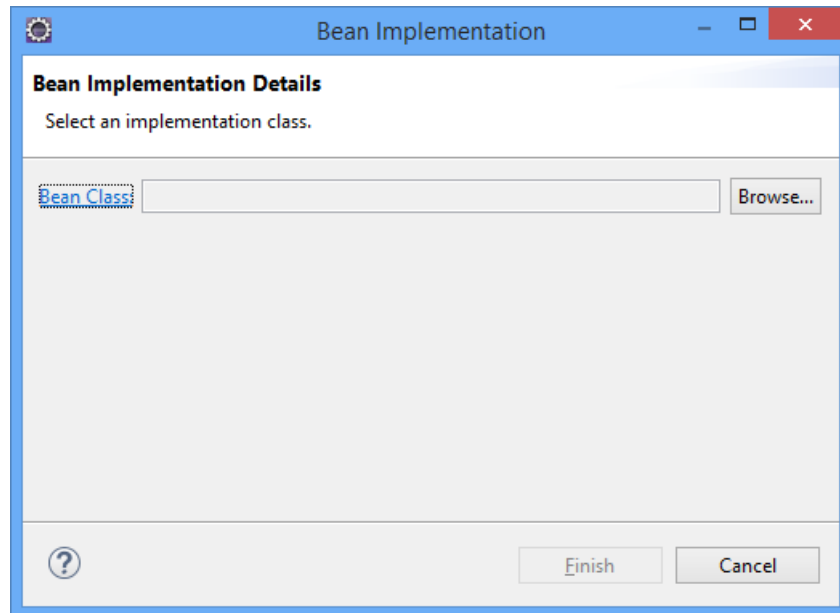


Click **Finish** and the diagram looks like this:



Select the **Bean** implementation from the palette, and drop it onto the component.

The following dialog will appear:



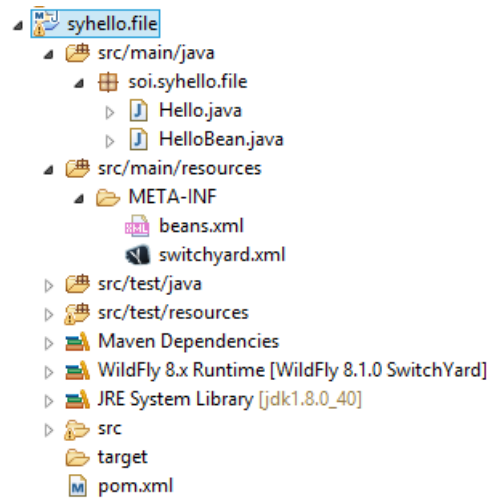
Click on the **Bean Class** link to create a new Java class.

In the **New Bean Service** dialog accept the default **HelloBean** class name, and click **Finish**. Close the **Bean Implementation** dialog also with **Finish**.

Rename the **Component** to **HelloBean** and the diagram should look like as:



The project structure looks like as:



Change the contents of **Hello.java** to:

```
package soi.syhello.file;

public interface Hello {
    public void sayHello(String name);
}
```

Change the contents of **HelloBean.java** to:

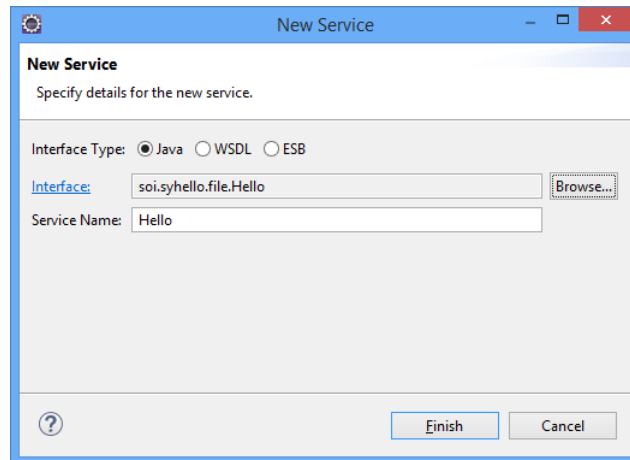
```
package soi.syhello.file;

import org.switchyard.component.bean.Service;

@Service(Hello.class)
public class HelloBean implements Hello {

    @Override
    public void sayHello(String name) {
        System.out.println("Hello: "+name);
    }
}
```

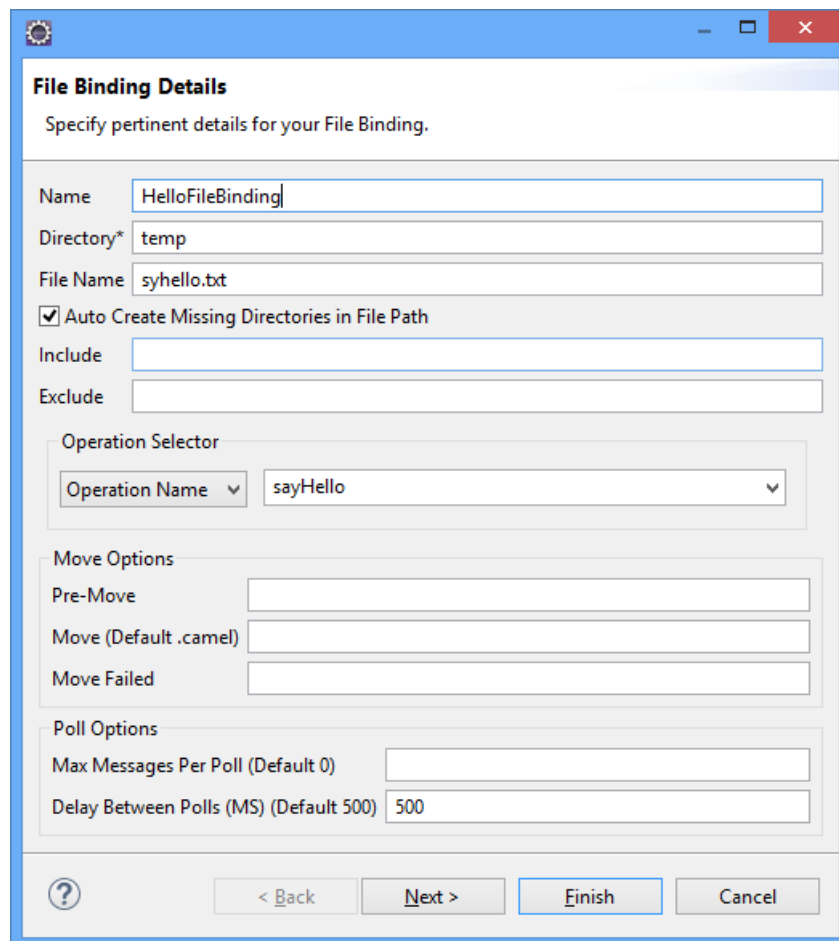
Add a new service to the **sayhello.file** composite component and browse for the **Hello** interface:



The 'New Service' dialog box is shown. It has a title bar with a gear icon and standard window controls. The main area is titled 'New Service' with the instruction 'Specify details for the new service.' Below this, there are three radio buttons for 'Interface Type': 'Java' (selected), 'WSDL', and 'ESB'. There are two text input fields: 'Interface:' containing 'soi.sayhello.file.Hello' and 'Service Name:' containing 'Hello'. A 'Browse...' button is next to the 'Interface:' field. At the bottom, there are three buttons: a help icon (?), 'Finish', and 'Cancel'.

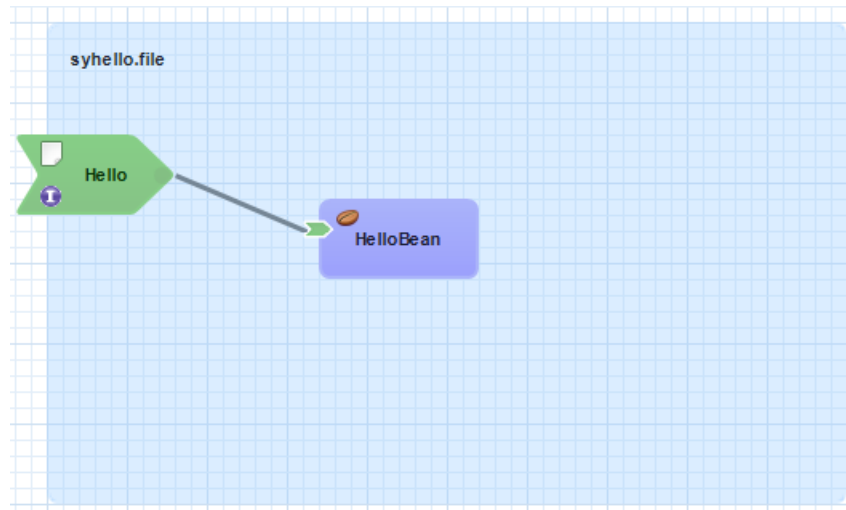
Click **Finish**. Connect the **Hello** service of the **sayhello.file** composite with the **Hello** service of the **HelloBean** component using the **Promote** element from the palette.

Select the **File** binding from the palette and drop it onto the **Hello** service of the **sayhello.file** composite. Set the following in the dialog:



The 'File Binding Details' dialog box is shown. It has a title bar with a gear icon and standard window controls. The main area is titled 'File Binding Details' with the instruction 'Specify pertinent details for your File Binding.' Below this, there are several sections: 'Name' with text 'HelloFileBinding', 'Directory*' with text 'temp', 'File Name' with text 'sayhello.txt', a checked checkbox 'Auto Create Missing Directories in File Path', 'Include' and 'Exclude' text boxes, an 'Operation Selector' section with a dropdown 'Operation Name' set to 'sayHello', a 'Move Options' section with 'Pre-Move', 'Move (Default .camel)', and 'Move Failed' text boxes, and a 'Poll Options' section with 'Max Messages Per Poll (Default 0)' and 'Delay Between Polls (MS) (Default 500)' text boxes. At the bottom, there are four buttons: a help icon (?), '< Back', 'Next >', and 'Finish'. The 'Finish' button is highlighted.

Click **Finish**. The diagram should look like as:



The corresponding SCA XML should be:

```
<?xml version="1.0" encoding="UTF-8"?>
<sy:switchyard xmlns:bean="urn:switchyard-component-bean:config:2.0"
xmlns:file="urn:switchyard-component-camel-file:config:2.0"
xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
xmlns:sy="urn:switchyard-config:switchyard:2.0" name="syhello.file"
targetNamespace="urn:soi:syhello.file:1.0">
  <sca:composite name="syhello.file" targetNamespace="urn:soi:syhello.file:1.0">
    <sca:component name="HelloBean">
      <bean:implementation.bean class="soi.syhello.file.HelloBean"/>
      <sca:service name="Hello">
        <sca:interface.java interface="soi.syhello.file.Hello"/>
      </sca:service>
    </sca:component>
    <sca:service name="Hello" promote="HelloBean/Hello">
      <sca:interface.java interface="soi.syhello.file.Hello"/>
      <file:binding.file name="HelloFileBinding">
        <sy:operationSelector operationName="sayHello"/>
        <file:directory>temp</file:directory>
        <file:fileName>syhello.txt</file:fileName>
        <file:consume/>
      </file:binding.file>
    </sca:service>
  </sca:composite>
</sy:switchyard>
```

Start the WildFly server and add the **syhello.file** project to the server. Although there may be some warnings written to the standard error, the project deployment should be successful:

```
Markers Properties Servers Data Source Explorer Snippets Console
WildFly 8.1.0 SwitchYard [JBoss Application Server Startup Configuration] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (Apr 2, 2015, 5:43:34 PM)
17:44:01,693 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) StreamCaching is not in use. If using streams t
17:44:01,693 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Total 0 routes, of which 0 is started.
17:44:01,694 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Apache Camel 2.0.0.CR1 (CamelContext: camel-1)
17:44:01,880 INFO [org.apache.camel.impl.converter.DefaultTypeConverter] (MSC service thread 1-1) Loaded 214 type converters
17:44:02,033 ERROR [stderr] (MSC service thread 1-1) Warning: org.apache.xerces.jaxp.SAXParserImpl$JAXPSAXParser: Property 'http://www.oracle.com/x
17:44:02,140 ERROR [stderr] (MSC service thread 1-1) Compiler warnings:
17:44:02,143 ERROR [stderr] (MSC service thread 1-1) WARNING: 'org.apache.xerces.jaxp.SAXParserImpl: Property 'http://javax.xml.XMLConstants/prop
17:44:02,494 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Route: direct:{urn:soi:syhello.file:1.0}Hello s
17:44:02,685 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Route: V1CamelFileBindingModel/Hello@HelloFileB
17:44:02,738 INFO [org.jboss.as.server] (DeploymentScanner-threads - 1) JBAS018559: Deployed "syhello.file.jar" (runtime-name : "syhello.file.jar")
```

In the file system there should be a new folder called **temp** under the bin folder of the WildFly server:

c:\Programs\wildfly-8.1.0.Final\bin\temp\

Create a new file called **syhello.txt** in this folder with some text in it, e.g.:

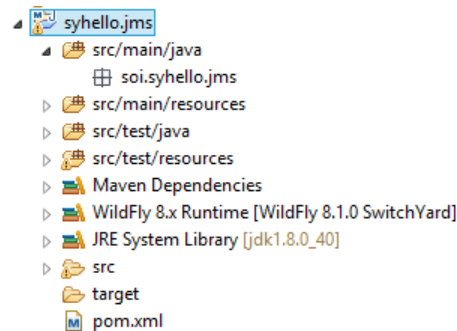
Alice & Bob

The file should be recognized and deleted by the server, and the following should appear in the server log:

```
Markers Properties Servers Data Source Explorer Snippets Console
WildFly 8.1.0 SwitchYard [JBoss Application Server Startup Configuration] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (Apr 2, 2015, 5:43:34 PM)
17:44:01,694 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Apache Cam
17:44:01,880 INFO [org.apache.camel.impl.converter.DefaultTypeConverter] (MSC service thread 1-1) Loaded 214 t
17:44:02,033 ERROR [stderr] (MSC service thread 1-1) Warning: org.apache.xerces.jaxp.SAXParserImpl$JAXPSAXPars
17:44:02,140 ERROR [stderr] (MSC service thread 1-1) Compiler warnings:
17:44:02,143 ERROR [stderr] (MSC service thread 1-1) WARNING: 'org.apache.xerces.jaxp.SAXParserImpl: Property
17:44:02,494 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Route: dir
17:44:02,685 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Route: V1C
17:44:02,738 INFO [org.jboss.as.server] (DeploymentScanner-threads - 1) JBAS018559: Deployed "syhello.file.jar"
17:50:59,483 INFO [stdout] (Camel (camel-1) thread #0 - file://temp) Hello: Alice & Bob
```

3 Hello World JMS service

Create a new **SwitchYard Project** in Eclipse. The project name should be **syhello.jms**, and the group id should be **soi**. The project should look like this:



Create a new file called **config.cli** directly under the project (next to the **pom.xml**) with the following content:

```
if (outcome != success) of /subsystem=messaging/hornetq-server=default/↵
jms-queue=HelloQueue:read-resource
    jms-queue add --queue-address=HelloQueue --entries=java:/jms/queue/HelloQueue,↵
java:jboss/exported/jms/queue/HelloQueue
end-if
```

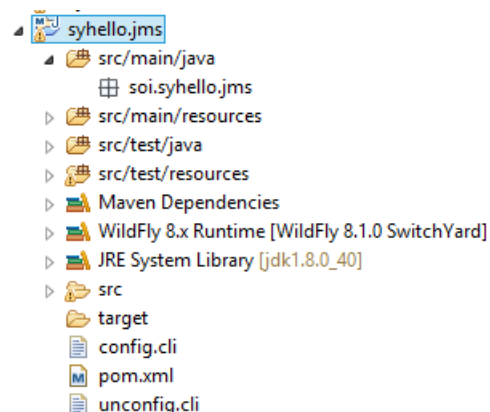
(The ↵ symbol means that the next line is not a new line but is a continuation of the current line.)

Create a new file called **unconfig.cli** directly under the project (next to the **pom.xml**) with the following content:

```
if (outcome == success) of /subsystem=messaging/hornetq-server=default/↵
jms-queue=HelloQueue:read-resource
    jms-queue remove --queue-address=HelloQueue
end-if
```

The **config.cli** command file creates a new JMS queue called **HelloQueue**, while the **unconfig.cli** command file removes this JMS queue from the server.

The project should now look like as:



Make sure that the WildFly server is running.

Open a command prompt from inside the **syhello.jms** project folder and issue the following command:

```
c:\Programs\wildfly-8.1.0.Final\bin\jboss-cli.bat --connect --file=config.cli
```

(If the connection is unsuccessful, make sure that the WildFly server is running and that the controller port number in the **jboss-cli.xml** inside the WildFly **bin** folder is the same as the management port number in the **standalone.xml** file.)

If the command is executed successfully, the following lines should appear in the server log:

```
21:48:23,330 INFO [org.hornetq.core.server] (ServerService Thread Pool -- 62) HQ221003: trying to deploy queue jms.queue.HelloQueue
21:48:23,331 INFO [org.jboss.as.messaging] (ServerService Thread Pool -- 62) JBA5011601: Bound messaging object to jndi name java:jboss/exported/jms/queue/HelloQueue
21:48:23,331 INFO [org.jboss.as.messaging] (ServerService Thread Pool -- 62) JBA5011601: Bound messaging object to jndi name java:jms/queue/HelloQueue
```

If you no longer need this queue, you can remove it from the server with the **unconfig.cli**:

```
c:\Programs\wildfly-8.1.0.Final\bin\jboss-cli.bat --connect --file=unconfig.cli
```

(Make sure to run this command from the **syhello.jms** project folder so that the **unconfig.cli** file is available.)

The **HelloQueue** should also appear in the admin console of the server after executing the **config.cli** command:

The screenshot shows the WildFly 8.1.0.Final admin console. The left sidebar has a menu with 'JMS Destinations' selected. The main content area is titled 'JMS Queue Metrics' and shows 'Metrics for JMS queues.' Below this is a 'Queue Selection' table with one entry: 'HelloQueue' with JNDI name '[java:/jms/queue/HelloQueue, java:jboss/exported/jms/que...]'. Below the table is a 'Messages' tab and a 'Consumer' tab. At the bottom, there is a bar chart titled 'In-Flight Messages' showing 'In Delivery count' on the x-axis (0 to 100) and 'Compared to Queued' on the y-axis. The bar chart shows a single bar at 100.

Name	JNDI
HelloQueue	[java:/jms/queue/HelloQueue, java:jboss/exported/jms/que...]

In-Flight Messages

In Delivery count

Compared to Queued

Create a new SCA component called **HelloBean**. Its service interface should be the **soi.syhello.jms.Hello** Java interface and the implementation should be **soi.syhello.jms.HelloBean** Java class.

The **Hello.java** interface should contain:

```
package soi.syhello.jms;

public interface Hello {
    public void sayHello(String name);
}
```

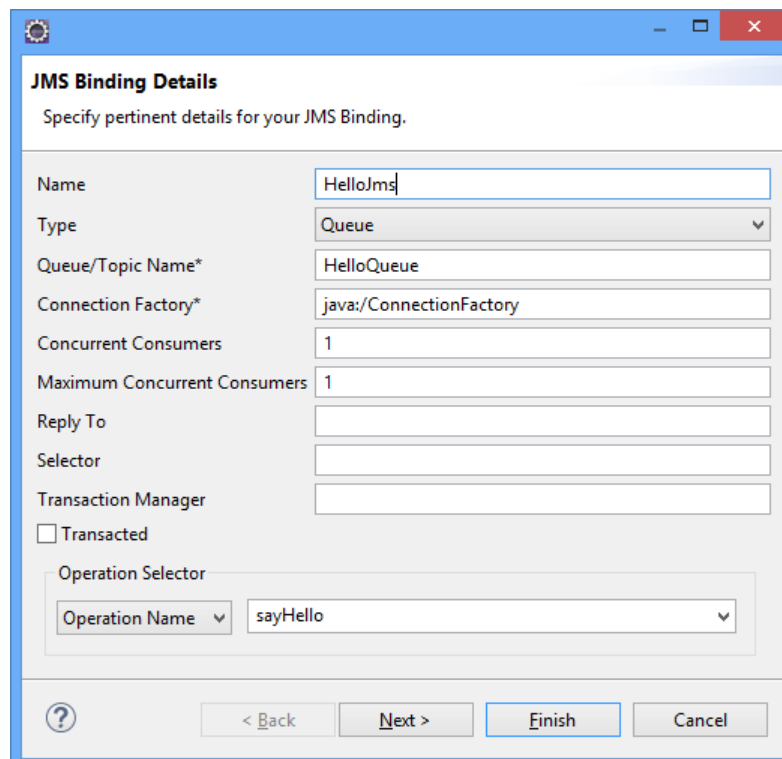
The **HelloBean.java** implementation should contain:

```
package soi.syhello.jms;

import org.switchyard.component.bean.Service;

@Service(Hello.class)
public class HelloBean implements Hello {
    @Override
    public void sayHello(String name) {
        System.out.println("Hello from JMS bean: "+name);
    }
}
```

Promote the **Hello** service interface to the **syhello.jms** composite as a service, and add a JMS binding to this composite service:

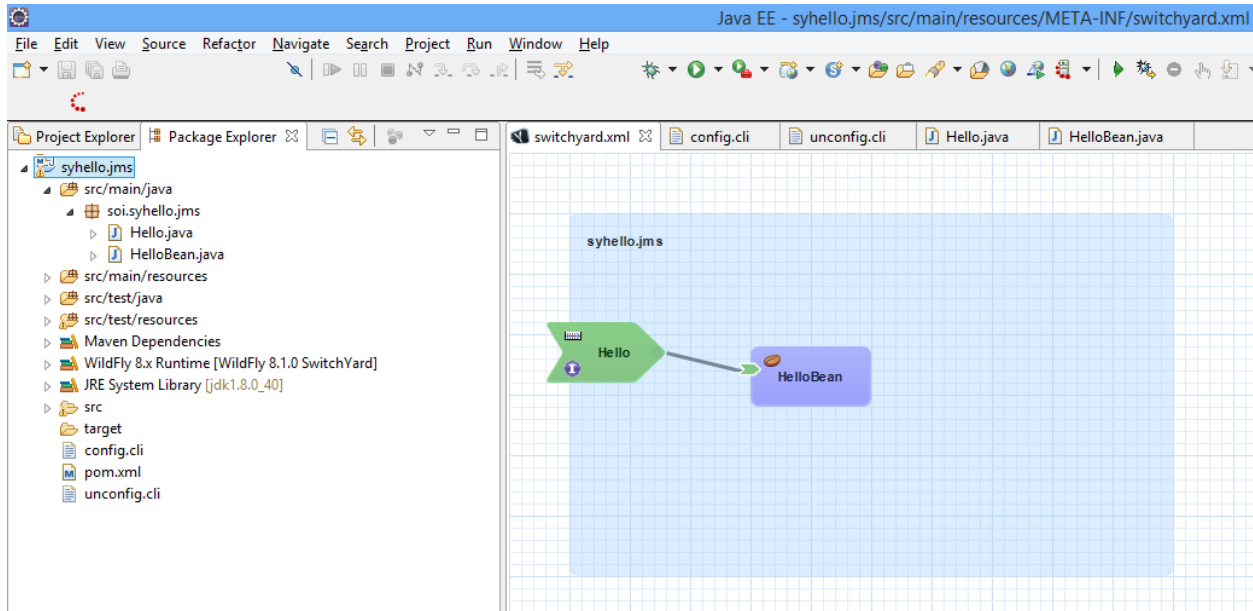


JMS Binding Details
Specify pertinent details for your JMS Binding.

Name	HelloJms
Type	Queue
Queue/Topic Name*	HelloQueue
Connection Factory*	java:/ConnectionFactory
Concurrent Consumers	1
Maximum Concurrent Consumers	1
Reply To	
Selector	
Transaction Manager	
<input type="checkbox"/> Transacted	
Operation Selector	
Operation Name	sayHello

? < Back Next > Finish Cancel

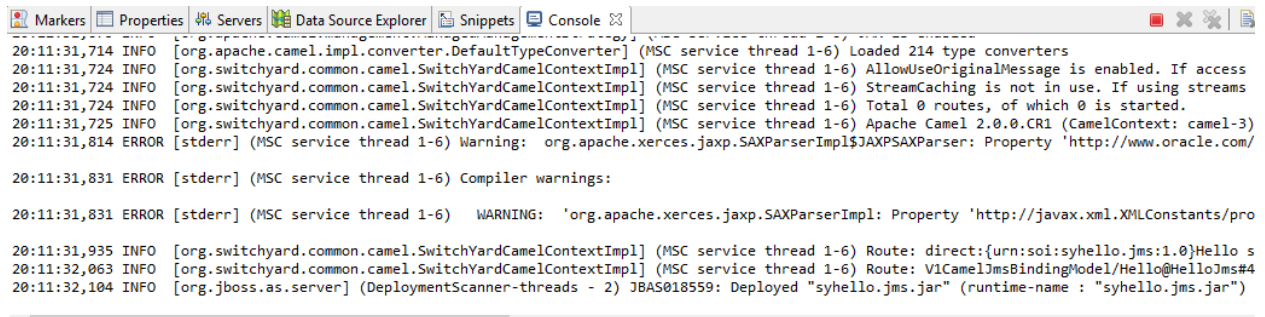
The project should look like this:



The SwitchYard XML should contain the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<sy:switchyard xmlns:bean="urn:switchyard-component-bean:config:2.0"
xmlns:jms="urn:switchyard-component-camel-jms:config:2.0"
xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
xmlns:sy="urn:switchyard-config:switchyard:2.0" name="syhello.jms"
targetNamespace="urn:soi:syhello.jms:1.0">
  <sca:composite name="syhello.jms" targetNamespace="urn:soi:syhello.jms:1.0">
    <sca:component name="HelloBean">
      <bean:implementation.bean class="soi.syhello.jms>HelloBean"/>
      <sca:service name="Hello">
        <sca:interface.java interface="soi.syhello.jms>Hello"/>
      </sca:service>
    </sca:component>
    <sca:service name="Hello" promote="HelloBean/Hello">
      <sca:interface.java interface="soi.syhello.jms>Hello"/>
      <jms:binding.jms name="HelloJms">
        <sy:operationSelector operationName="sayHello"/>
        <jms:queue>HelloQueue</jms:queue>
        <jms:connectionFactory>java:/ConnectionFactory</jms:connectionFactory>
      </jms:binding.jms>
    </sca:service>
  </sca:composite>
</sy:switchyard>
```

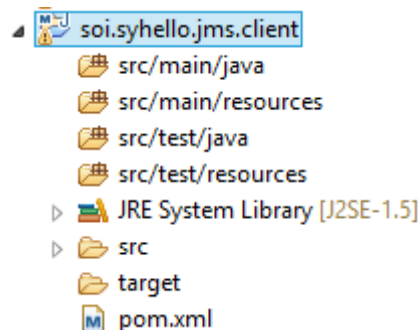

Deploy the project to the server and it should print the following to the server log:



```
20:11:31,714 INFO [org.apache.camel.impl.converter.DefaultTypeConverter] (MSC service thread 1-6) Loaded 214 type converters
20:11:31,724 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-6) AllowUseOriginalMessage is enabled. If access
20:11:31,724 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-6) StreamCaching is not in use. If using streams
20:11:31,724 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-6) Total 0 routes, of which 0 is started.
20:11:31,725 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-6) Apache Camel 2.0.0.CR1 (CamelContext: camel-3)
20:11:31,814 ERROR [stderr] (MSC service thread 1-6) Warning: org.apache.xerces.jaxp.SAXParserImpl$JAXPSAXParser: Property 'http://www.oracle.com/
20:11:31,831 ERROR [stderr] (MSC service thread 1-6) Compiler warnings:
20:11:31,831 ERROR [stderr] (MSC service thread 1-6) WARNING: 'org.apache.xerces.jaxp.SAXParserImpl: Property 'http://javax.xml.XMLConstants/pro
20:11:31,935 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-6) Route: direct:{urn:soi:syhello.jms:1.0}Hello s
20:11:32,063 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-6) Route: ViCamelJmsBindingModel/Hello@HelloJms#4
20:11:32,104 INFO [org.jboss.as.server] (DeploymentScanner-threads - 2) JBAS018559: Deployed "syhello.jms.jar" (runtime-name : "syhello.jms.jar")
```

In order to test the component, we need a client to send a message to the **HelloQueue** JMS queue.

Create a new simple Maven project (skip archetype selection) in Eclipse. The group id and the artifact id should both be **soi.syhello.jms.client**, and use the default **jar** packaging. The project should look like this:



Edit the **pom.xml** and it should have the following content:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>soi.syhello.jms.client</groupId>
  <artifactId>soi.syhello.jms.client</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <version.wildfly>8.1.0.Final</version.wildfly>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.wildfly.bom</groupId>
```

```

        <artifactId>jboss-javaee-7.0-with-tools</artifactId>
        <version>${version.wildfly}</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
    <dependency>
        <groupId>org.wildfly</groupId>
        <artifactId>wildfly-ejb-client-bom</artifactId>
        <version>${version.wildfly}</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
</dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>org.wildfly</groupId>
        <artifactId>wildfly-ejb-client-bom</artifactId>
        <type>pom</type>
    </dependency>
    <dependency>
        <groupId>org.wildfly</groupId>
        <artifactId>wildfly-jms-client-bom</artifactId>
        <type>pom</type>
    </dependency>
</dependencies>

</project>

```

Create a new class under **src/main/java** called **HelloJmsClient** under the package **soi.syhello.jms.client**:

```

package soi.syhello.jms.client;

import java.util.Properties;
import java.util.logging.Logger;

import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSContext;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class HelloJmsClient {

    private static final Logger Log = Logger.getLogger(HelloJmsClient.class
        .getName());
    // Set up all the default values
    private static final String DEFAULT_MESSAGE = "Alice & Bob";
    private static final String DEFAULT_CONNECTION_FACTORY = "jms/RemoteConnectionFactory";
    private static final String DEFAULT_DESTINATION = "java:/jms/queue/HelloQueue";
    private static final String DEFAULT_USERNAME = "guest";
    private static final String DEFAULT_PASSWORD = "guest";
    private static final String INITIAL_CONTEXT_FACTORY =
"org.jboss.naming.remote.client.InitialContextFactory";
    private static final String PROVIDER_URL = "http-remoting://127.0.0.1:18080";

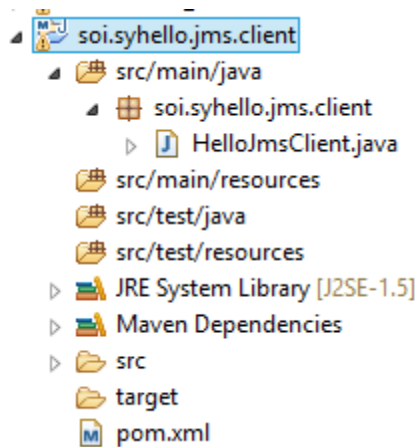
```

```

public static void main(String[] args) {
    Context namingContext = null;
    try {
        try {
            String userName = System.getProperty("username",
                DEFAULT_USERNAME);
            String password = System.getProperty("password",
                DEFAULT_PASSWORD);
            // Set up the namingContext for the JNDI lookup
            final Properties env = new Properties();
            env.put(Context.INITIAL_CONTEXT_FACTORY,
                INITIAL_CONTEXT_FACTORY);
            env.put(Context.PROVIDER_URL,
                System.getProperty(Context.PROVIDER_URL, PROVIDER_URL));
            env.put(Context.SECURITY_PRINCIPAL, userName);
            env.put(Context.SECURITY_CREDENTIALS, password);
            namingContext = new InitialContext(env);
            // Perform the JNDI lookups
            String connectionFactoryString = System.getProperty(
                "connection.factory", DEFAULT_CONNECTION_FACTORY);
            Log.info("Attempting to acquire connection factory \""
                + connectionFactoryString + "\"");
            ConnectionFactory connectionFactory = (ConnectionFactory) namingContext
                .lookup(connectionFactoryString);
            Log.info("Found connection factory \""
                + connectionFactoryString + "\" in JNDI");
            String destinationString = System.getProperty("destination",
                DEFAULT_DESTINATION);
            Log.info("Attempting to acquire destination \""
                + destinationString + "\"");
            Destination destination = (Destination) namingContext
                .lookup(destinationString);
            Log.info("Found destination \"" + destinationString
                + "\" in JNDI");
            String content = System.getProperty("message.content",
                DEFAULT_MESSAGE);
            JMSContext context = connectionFactory.createContext(userName,
                password);
            Log.info("Sending message with content: " + content);
            // Send the message
            context.createProducer().send(destination, content);
            context.close();
        } catch (Exception e) {
            Log.severe(e.getMessage());
        }
    } finally {
        if (namingContext != null) {
            try {
                namingContext.close();
            } catch (NamingException e) {
                Log.severe(e.getMessage());
            }
        }
    }
}
}

```

The project should look like this:



Run the **HelloJmsClient** and the following should be printed to the console:

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> HelloJmsClient [Java Application] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (Apr 2, 2015)
INFO: XNIO NIO Implementation Version 3.2.2.Final
Apr 02, 2015 9:53:13 PM org.jboss.remoting3.EndpointImpl <clinit>
INFO: JBoss Remoting version 4.0.3.Final
Apr 02, 2015 9:53:13 PM soi.syhello.jms.client.HelloJmsClient main
INFO: Attempting to acquire connection factory "jms/RemoteConnectionFactory"
Apr 02, 2015 9:53:13 PM soi.syhello.jms.client.HelloJmsClient main
INFO: Found connection factory "jms/RemoteConnectionFactory" in JNDI
Apr 02, 2015 9:53:13 PM soi.syhello.jms.client.HelloJmsClient main
INFO: Attempting to acquire destination "java:/jms/queue/HelloQueue"
Apr 02, 2015 9:53:13 PM soi.syhello.jms.client.HelloJmsClient main
INFO: Found destination "java:/jms/queue/HelloQueue" in JNDI
Apr 02, 2015 9:53:14 PM soi.syhello.jms.client.HelloJmsClient main
INFO: Sending message with content: Alice & Bob
```

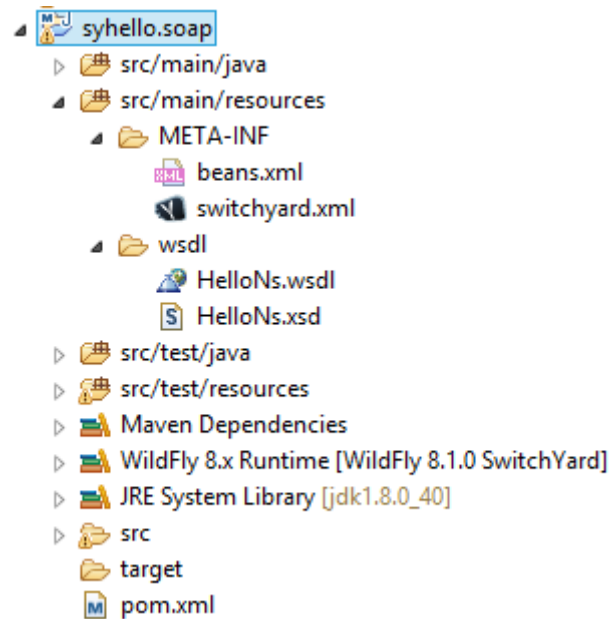
On the server side the following should appear:

```
Markers Properties Servers Data Source Explorer Snippets Console
WildFly 8.1.0 SwitchYard [JBoss Application Server Startup Configuration] C:\Program Files\Java\jdk1.8.0_40\bin\javaw.exe (Apr 2, 2015, 7:44:44 PM)
21:52:26,373 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Apache Camel 2.0.0.CR1 (CamelContext:
21:52:26,430 ERROR [stderr] (MSC service thread 1-1) Warning: org.apache.xerces.jaxp.SAXParserImpl$JAXPSAXParser: Property 'http://www.or
21:52:26,434 ERROR [stderr] (MSC service thread 1-1) Compiler warnings:
21:52:26,434 ERROR [stderr] (MSC service thread 1-1) WARNING: 'org.apache.xerces.jaxp.SAXParserImpl: Property 'http://javax.xml.XMLCons
21:52:26,515 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Route: direct:{urn:soi:syhello.jms:1.
21:52:26,551 INFO [org.switchyard.common.camel.SwitchYardCamelContextImpl] (MSC service thread 1-1) Route: V1CamelJmsBindingModel/Hello@H
21:52:26,579 INFO [org.jboss.as.server] (DeploymentScanner-threads - 1) JBAS018565: Replaced deployment "syhello.jms.jar" with deployment
21:53:14,137 INFO [stdout] (pool-25-thread-1) Hello from JMS bean: Alice & Bob
21:53:14,160 INFO [org.jboss.as.naming] (default task-4) JBAS011806: Channel end notification received, closing channel Channel ID 331d67
```

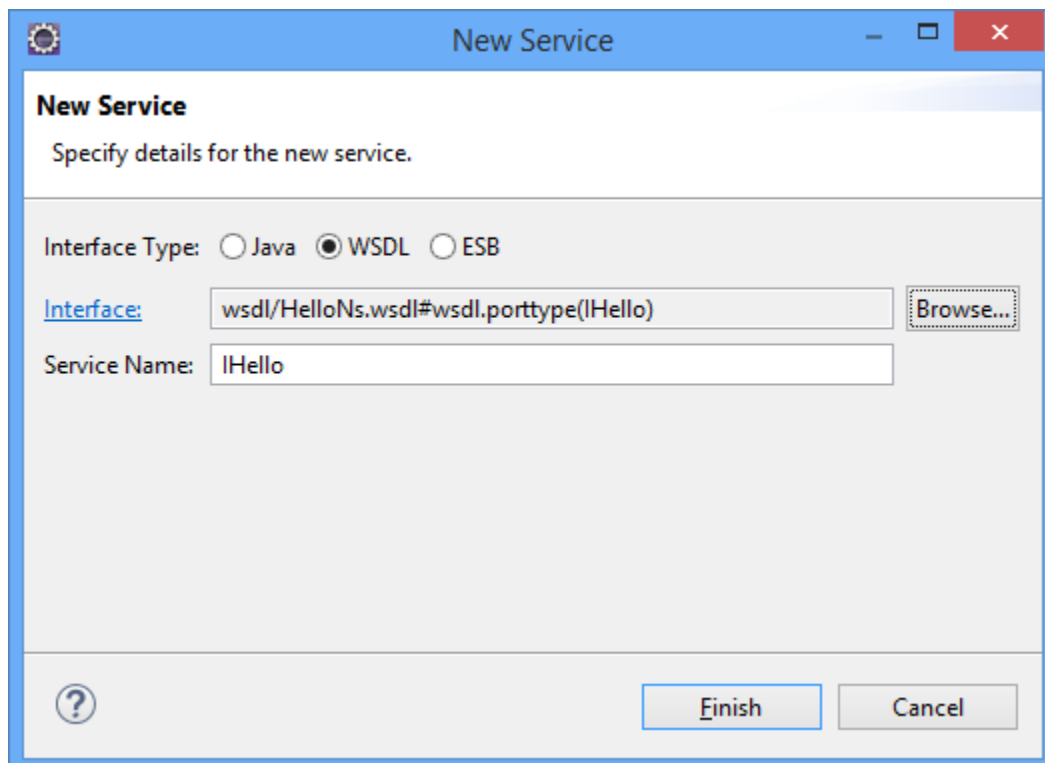
4 Hello World SOAP service

Create a new **SwitchYard Project** in Eclipse. The project name should be **syhello.soap**, and the group id should be **soi**. Copy the **HelloNS.wsdl** and **HelloNS.xsd** files into the **src/main/resources/wsdl** folder.

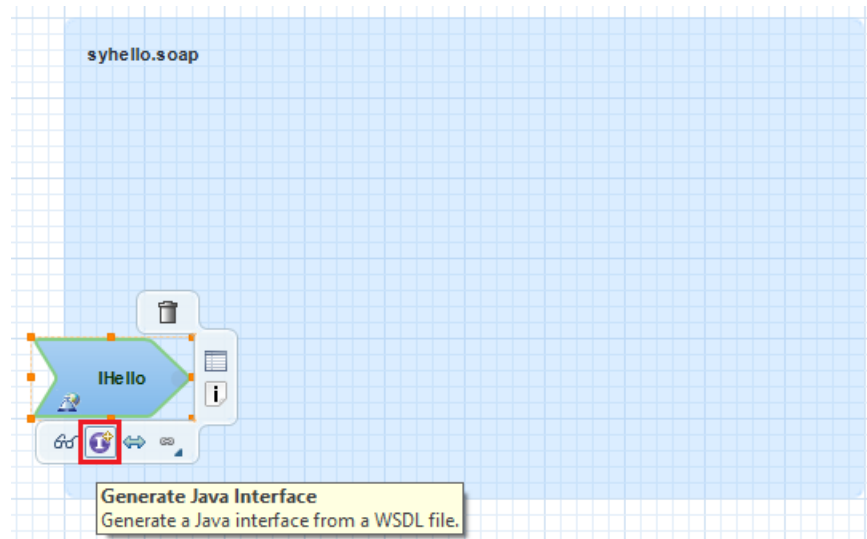
The project should look like this:



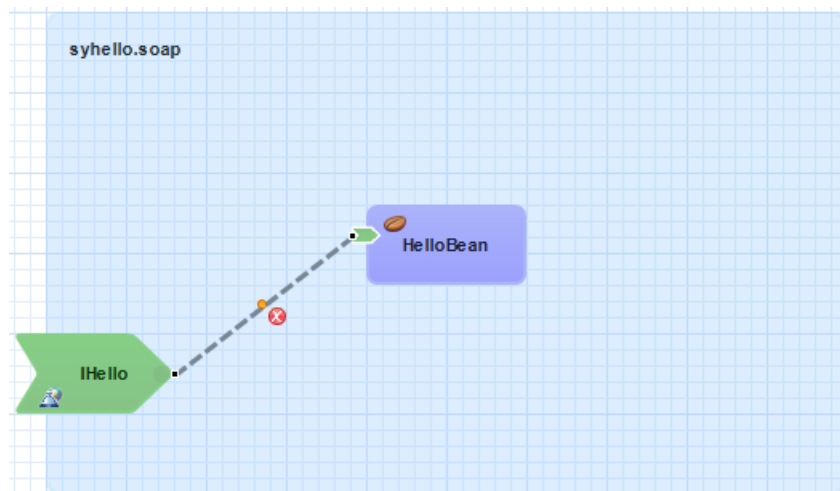
Add a new service to the **syhello.soap** composite in the SwitchYard designer and set the interface to be the **HelloNS.wsdl**:



Click **Finish**, hover over the service and generate a Java interface:



Create a new component with the generated **>Hello** Java interface, implement the component with a **HelloBean** Java class, and promote it to the external interface:



There is an error on the promote connection, since the transformation between the XML and Java objects are missing. Right click on the connection and select **Create Required Transformers**.

There are two ways to create a transformer: a Java transformer or a JAXB transformer.

The JAXB transformer works only if the request and response wrapper classes are annotated with **@XmlRootElement**, too. Unfortunately, this annotation is missing from the generated Java code, so it has to be added manually to every wrapper class. For example:

```
package soi.syhello.soap;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

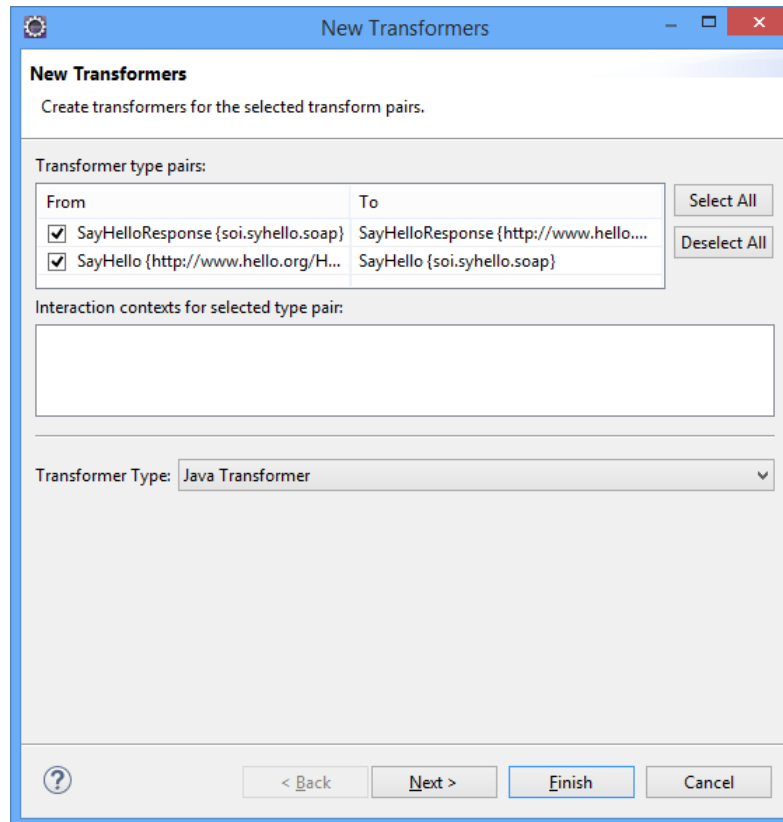
/**
 * <p>Java class for SayHello complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this class.
 *
 * <pre>
 * <complexType name="SayHello">
 *   <complexContent>
 *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       <sequence>
 *         <element name="name" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *       </sequence>
 *     </restriction>
 *   </complexContent>
 * </complexType>
 * </pre>
 *
 */
@XmlRootElement(name="SayHello")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "SayHello", propOrder = {
    "name"
})
public class SayHello {

    @XmlElement(required = true, nillable = true)
    protected String name;

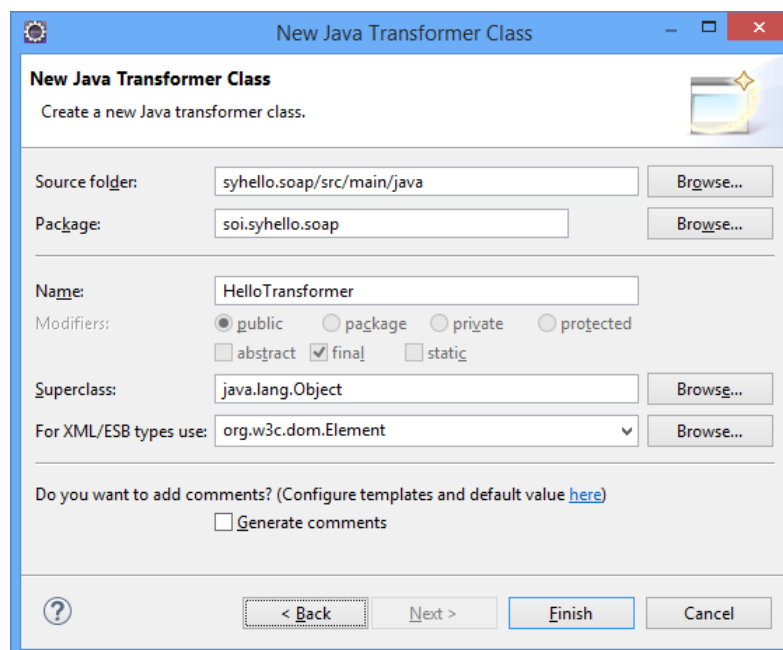
    /**
     * Gets the value of the name property.
     *
     * @return
     *     possible object is
     *     {@link String }
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the value of the name property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     */
    public void setName(String value) {
        this.name = value;
    }
}
```

The other solution is to create a Java transformer. In this case the generated code does not have to be modified. Select all the types offered:



Click **Next** and specify a class name for the transformer:



Click **Finish**. Implement the transformer as shown here:

```
package soi.syhello.soap;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import javax.xml.namespace.QName;
import javax.xml.transform.dom.DOMResult;

import org.switchyard.annotations.Transformer;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public final class HelloTransformer {
    private Element marshal(QName qname, Object obj) throws JAXBException {
        JAXBContext ctx = JAXBContext.newInstance(obj.getClass());
        Marshaller marshaller = ctx.createMarshaller();
        JAXBElement jaxbElem = new JAXBElement(qname, obj.getClass(), obj);
        DOMResult res = new DOMResult();
        marshaller.marshal(jaxbElem, res);
        Element elem = ((Document)res.getDocumentElement()).getDocumentElement();
        return elem;
    }

    private <T> T unmarshal(Element elem, Class<T> cls) throws JAXBException {
        JAXBContext ctx = JAXBContext.newInstance(cls);
        Unmarshaller unmarshaller = ctx.createUnmarshaller();
        return unmarshaller.unmarshal(elem, cls).getValue();
    }

    @Transformer(to = "{http://www.hello.org>HelloNs}SayHelloResponse")
    public Element transformSayHelloResponseToSayHelloResponse(
        SayHelloResponse from) throws JAXBException {
        return marshal(new QName("http://www.hello.org>HelloNs", "SayHelloResponse"),
            from);
    }

    @Transformer(from = "{http://www.hello.org>HelloNs}SayHello")
    public SayHello transformSayHelloToSayHello(Element from) throws JAXBException {
        return unmarshal(from, SayHello.class);
    }
}
```

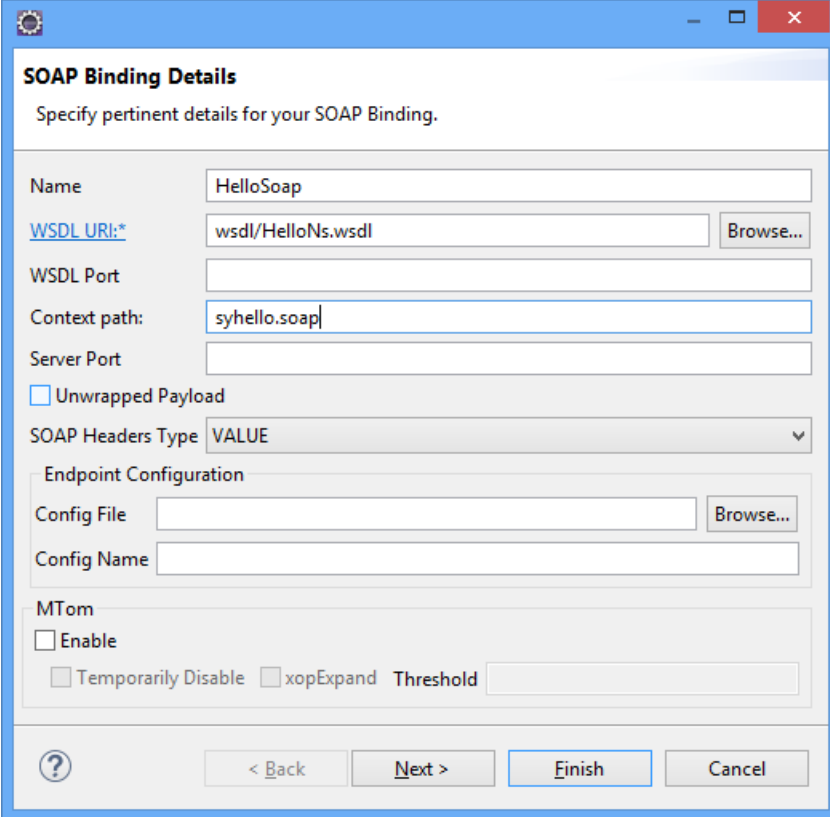
Implement the service class:

```
package soi.syhello.soap;

import org.switchyard.component.bean.Service;

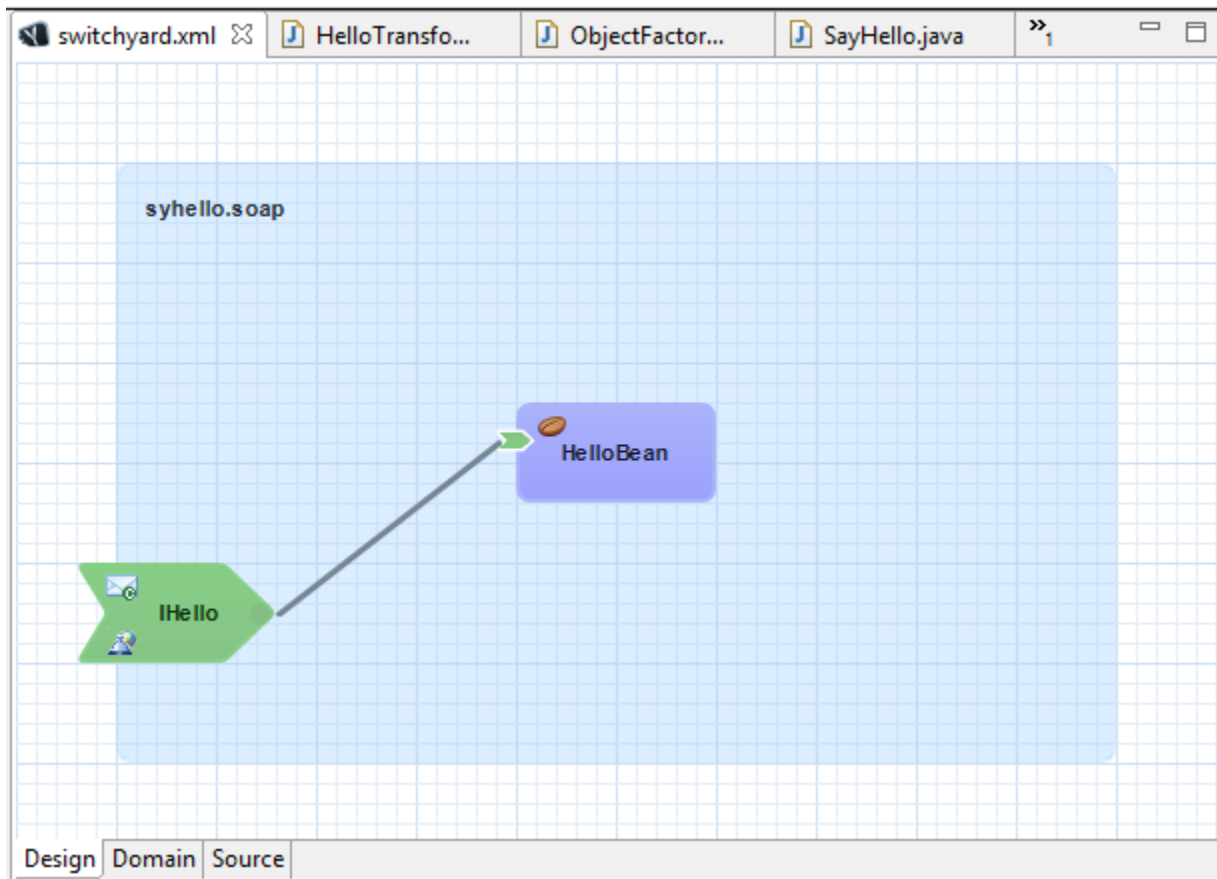
@Service(IHello.class)
public class HelloBean implements IHello {
    @Override
    public SayHelloResponse SayHello(soi.syhello.soap.SayHello parameters) {
        String result = "Hello: "+parameters.getName();
        SayHelloResponse response = new SayHelloResponse();
        response.setSayHelloResult(result);
        return response;
    }
}
```

Add a SOAP binding to the **IHello** service of the **syhello.soap** composite:

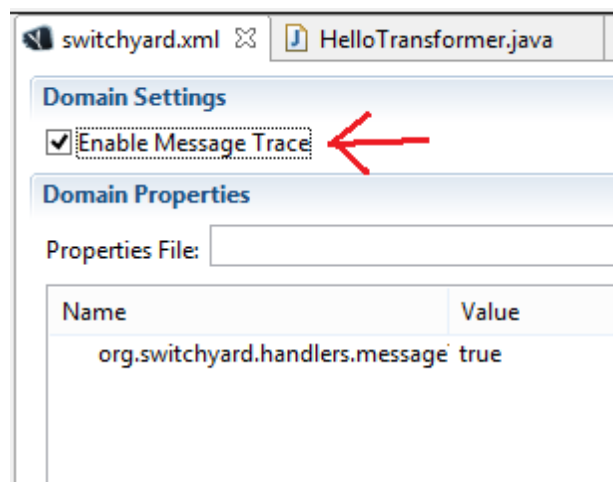
A screenshot of the 'SOAP Binding Details' dialog box in a software application. The dialog has a title bar with a gear icon and standard window controls. The main area is titled 'SOAP Binding Details' with a subtitle 'Specify pertinent details for your SOAP Binding.' Below this, there are several input fields: 'Name' with the value 'HelloSoap', 'WSDL URI:*' with 'wsdl/HelloNs.wsdl' and a 'Browse...' button, 'WSDL Port' (empty), 'Context path:' with 'syhello.soap', and 'Server Port' (empty). There is a checkbox for 'Unwrapped Payload' which is unchecked. Below that is a 'SOAP Headers Type' dropdown menu set to 'VALUE'. An 'Endpoint Configuration' section contains a 'Config File' field with a 'Browse...' button and a 'Config Name' field. At the bottom, there is an 'MTom' section with an 'Enable' checkbox (unchecked), and two sub-sections: 'Temporarily Disable' (unchecked) and 'xopExpand Threshold' (empty). The bottom of the dialog features a help icon, and four buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), and 'Cancel'.

Click **Finish**.

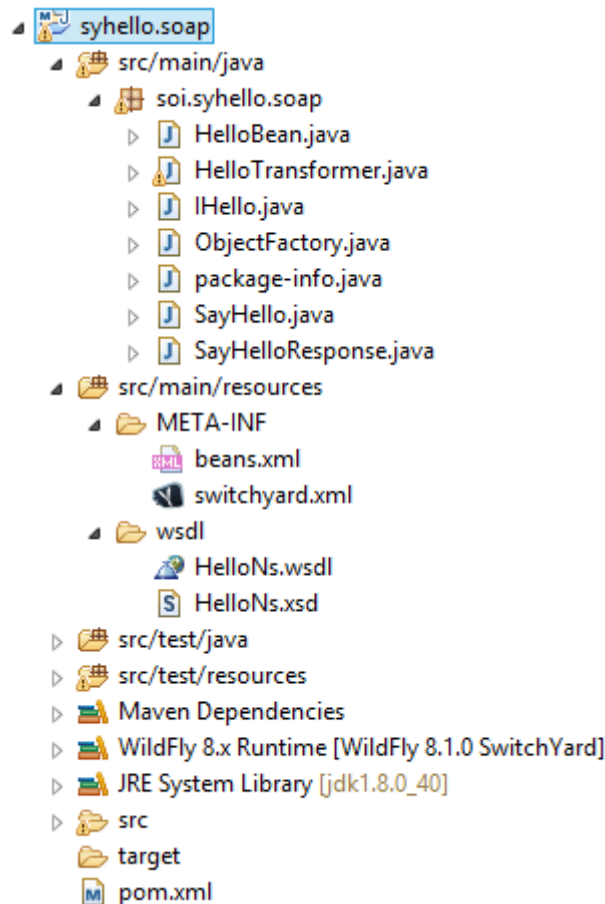
The SCA composite should look like this:



Click on the **Domain** tab at the bottom and check the **Enable Message Trace** checkbox to enable detailed message traces (this can help to solve communication problems, for example if there is a problem with the JAXB serialization):



The application should look like this:



Deploy the application and the URL of the service should appear in the server log:



Test the service with a JAX-WS client.