

# PETALS-SE-BPEL

Julien Lesbegueries

September 24, 2009

## Contents

<b>1</b>	<b>Introduction to the Maestro Service Engine</b>	<b>2</b>
<b>2</b>	<b>Using PetalsStudio</b>	<b>2</b>
2.1	Creating a BPEL Service Unit . . . . .	3
2.2	Packaging the Service Unit for Petals . . . . .	6
2.2.1	Service Unit descriptor . . . . .	6
2.2.2	Service Unit content . . . . .	7
2.3	Deployment and testing of the Service Unit . . . . .	8
<b>3</b>	<b>Advanced features</b>	<b>10</b>
3.1	Create a Proxy for Weather Web Service . . . . .	10
3.2	Service Engine Component Configuration . . . . .	13

# 1 Introduction to the Maestro Service Engine

The Maestro Service Engine allows to deploy BPEL 2.0 processes as services. It provides the support of business processes and workflows in the JBI environment (figure 1). It is composed of a CDK layer in order to manage JBI exchange, a Core layer to manage generic SOA workflows and a specific BPEL layer in order to manage the BPEL model.

A BPEL service is packaged and deployed as a BPEL Service Unit. When a request message reaches a Maestro *serviceEndpoint*, it triggers an instantiation of a new BPEL process. Once instantiated, a Maestro process interacts with the JBI environment to provide and consume the required services.

If you want to have more details about BPEL 2.0 specification, please consult this URL : <http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm>

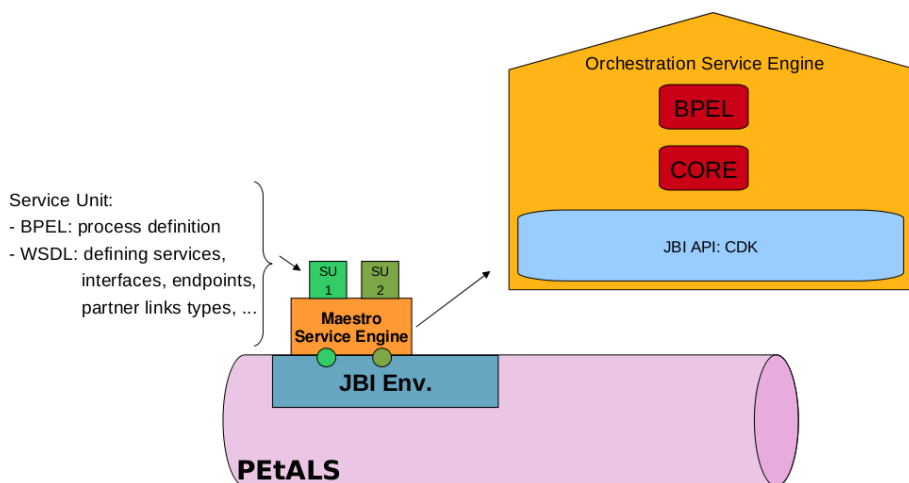


Figure 1: Maestro Service Engine.

**Note: The passover of the security context amongst the Message Exchange involved in a BPEL process is supported.**

This document is composed as follows. Section 2 describes how to use PetalsStudio in order to build a simple BPEL service unit. Section 3 describes some advanced features of the BPEL service engine.

## 2 Using PetalsStudio

This section describes an easy way for using the BPEL engine component of Petals, thanks to PetalsStudio<sup>1</sup>. This Specific Eclipse build provides plugins in order to automate the construction and configuration procedure.

The next sections explain how to create, configure and deploy a service unit (su) on PETALS with a simple use-case example.

<sup>1</sup>At the moment, you can find the Petals Eclipse plugin at <http://dist.ebmwebsourcing.com/eclipse/petals/>.

## 2.1 Creating a BPEL Service Unit

In PetalsStudio, you have to create a new project (*Other* menu). The wizard to create a BPEL project is in the “*Use a Petals technical service*” (figure 2).

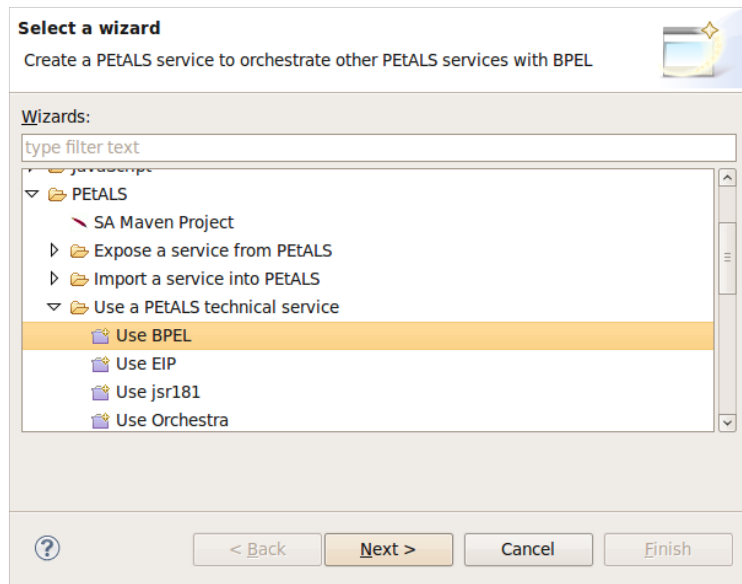


Figure 2: Step 1: New Petals technical service.

Then, you can create a new BPEL project or browse a WSDL corresponding to an existing BPEL process (figure 3). In our case we’re going to create a simple BPEL process (created by default).

A name has to be chosen and a su project is generated. Generated files are :

- the META-INF/jbi.xml of the su,
- a sample *process.bpel*
- its corresponding WSDL *process.wsdl*
- and a PetalsStudio specific file in order to save the graphic view (*process.bpelx*).

You can see a screenshot of this result on figure 4. In particular, you can visualize a BPEL process in a graphic view.

**Use the PETALS BPEL Service**  
Create a PETALS service to orchestrate other PETALS services with BPEL.

WSDL File:

Interface Namespace:

Interface Name:

Service Namespace:

Service Name:

End-point Name:

End-point Name Auto-Generated

Figure 3: Browsing the WSDL associated to the bpel process.

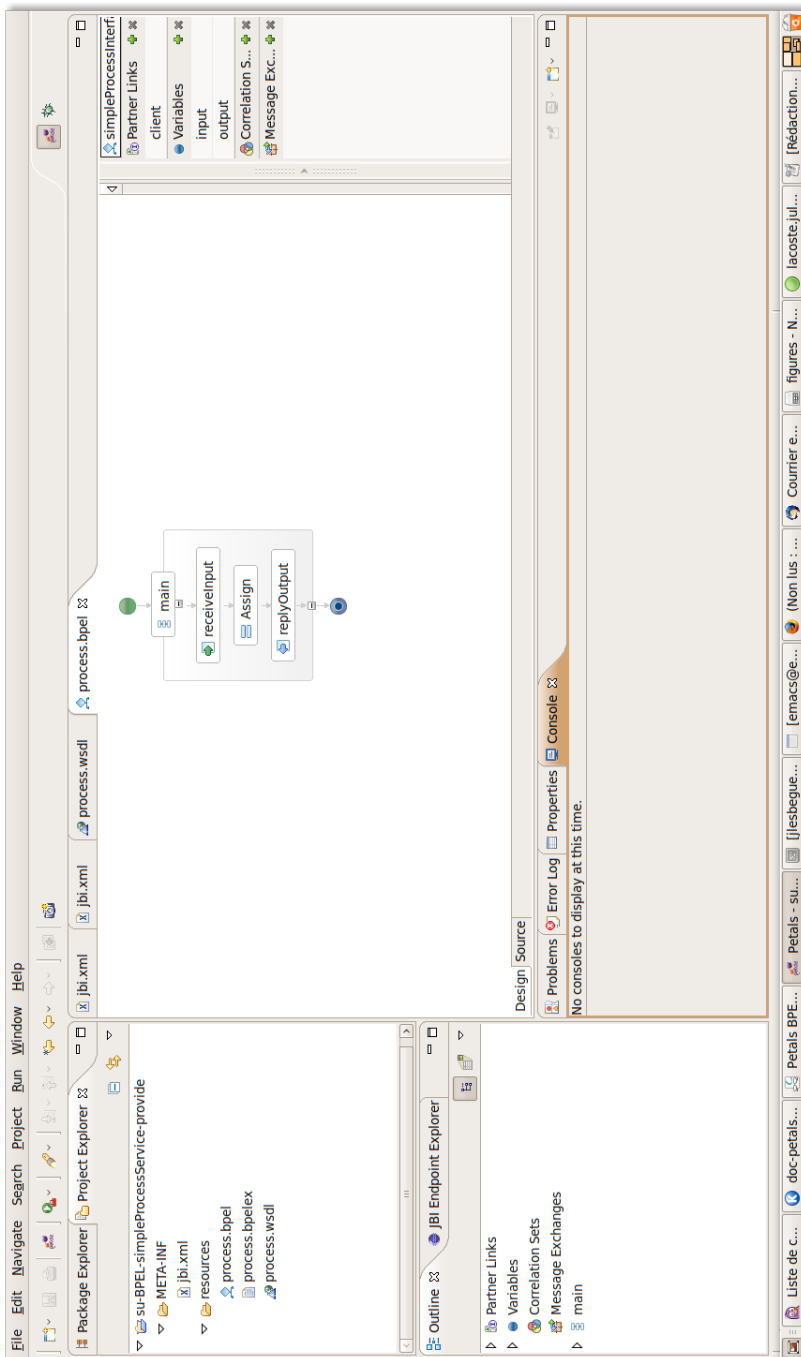


Figure 4: Files created by the Wizard.

The created files in figure 4 correspond to a default project with a very simple BPEL process example : A synchronous *Receive* → *Assign* → *Reply* where the assign action copies the string in input into the output. We can see a graphic visualization of this BPEL process.

## 2.2 Packaging the Service Unit for Petals

In order to build the service assembly (sa) (the zip file to deploy in Petals), simply right click on the su and choose *PETALS*→*Package for PETALS*.

This action creates an archive containing the jbi.xml file and some BPEL specific files that are detailed below.

### 2.2.1 Service Unit descriptor

The Service Unit descriptor file ( jbi.xml ) looks like this :

Listing 1: jbi.xml of the Service Unit

```

1 <!-- JBI descriptor for the PETALS BPEL component -->
<jbi:jbi version="1.0"
  xmlns:genNs="http://lesbegu.com"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi"
  xmlns:bpel="http://petals.ow2.org/components/bpel/version-1.0"
6  xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-5"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <jbi:services binding-component="false">
11  <jbi:provides
    interface-name="genNs:simpleProcessInterface"
    service-name="genNs:simpleProcessService"
    endpoint-name="simpleServiceSOAPPort">

    <!-- CDK specific elements -->
16  <petalsCDK:wSDL>process.wSDL</petalsCDK:wSDL>

    <!-- Component specific elements -->
    <bpel:bpel>process.bpel</bpel:bpel>

21  </jbi:provides>
  </jbi:services>
</jbi:jbi>

```

The table 1 lists and describes attributes in the jbi.xml of the Service Unit.

Table 1: Service Unit attributes to provide services.

Attribute	Description	Default	Required
bpel	The name of the BPEL file to use. The value of this parameter is:  - an URL,  - a file relative to the root of the SU package	-	Yes
wsdl	The corresponding wsdl.	-	Yes

### 2.2.2 Service Unit content

The Service Unit has to contain the following elements, packaged in an archive (table 2):

- The META-INF/jbi.xml descriptor file, has described above,
- The BPEL process file. If the BPEL is localized on an accessible URL, it does not need to be provided in the package.
- The WSDL file describing the related service and associated to the description of the BPEL process. if the WSDL is localized on an accessible URL, it does not need to be provided in the package.
- The external WSDL files describing imported partners in the description of the BPEL process.

The arborescence of the Service Unit is described in table 2.

#### Caution

The name of the BPEL process (BPEL root tag name) has to be the same that the porttype (interface) name defined in the WSDL document and the JBI descriptor.

Table 2: Files arborescence.

```

service-unit.zip
+ META-INF
  - jbi.xml (as defined above)
  - process.bpel (recommended)
  - service.wsdl (recommended)
  - externalservice(s).wsdl (required if the partner is
                             defined in the BPEL process)

```

### Caution

To bypass a classloading problem, the Maestro Component (petals-se-bpel) must be installed in PEtALS in an isolated class loader.

Please use a PEtALS server version higher or equal to 3.0 and set the `isolatedClassLoader` option at `true` in the `server.properties` file.

## 2.3 Deployment and testing of the Service Unit

You have now to launch a PEtALS (`./startup.sh` or `.bat`). Install et deploy this sa on a Petals where a `petals-se-bpel` component is installed and deployed.

In order to test our use-case we can install a `SampleClient` component (in the same way) that will find available endpoints in the PEtALS ESB. A double-click on the right endpoint will fill the form (figure 5). The “in” expected message is the following :

Listing 2: in message to send.

```
1 <les:process xmlns:les="http://lesbegu.com">
  <payload>
    <les:input>test</les:input>
  </payload>
</les:process>
```

The expected “out” message contains the same string as in input.





Figure 5: The Sample Client for testing our su.

### 3 Advanced features

This section describes first a realistic use-case, in which external Web Service is invoked by the BPEL process. Then a description of the Service Engine configuration is made, in order to help advanced users.

#### 3.1 Create a Proxy for Weather Web Service

The aim of a BPEL process is to invoke (external) Web Services. This section proposes a sample use-case with the invocation of an external “weather” Web Service.

First of all, you have to download the WSDL of the Web service<sup>2</sup> and copy it in the resources folder of your project. For now, we consider one Weather Web Service WSDL, containing a SOAP access and the operation *GetWeather* that takes as input a country name (string) and a city name (string) and returns as output a string.

Then you have to add a PartnerLink for the Weather Web Service, to your BPEL (Click on the “plus” button in the PartnerLinks panel). The figure 6 shows the result of the PartnerLink creation. Click on the “Browse” button in order to find a PartnerLinkType.

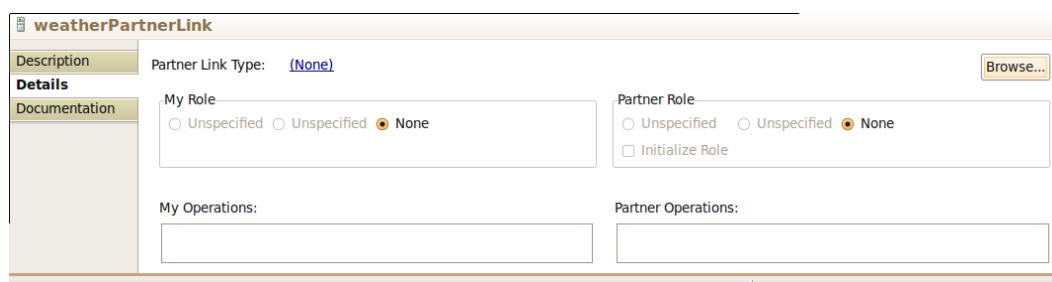


Figure 6: PartnerLink Description.

The PartnerLinkType is defined thanks to the WSDL (click on the “Add WSDL” button):

- Select the interface you want to use to contact the Web Service (double-click on the SOAP one: “GlobalWeatherSOAP”, figure 7),
- Choose the name of your PartnerLinkType,
- Choose a role for this PartnerLinkType (you can choose two roles but we won’t do this in our case),
- After finishing this creation, assign the role you created to your PartnerLink (figure 8).

---

<sup>2</sup>The Web Service can be found for instance at <http://www.webservices.net/globalweather.asmx?wsdl>.

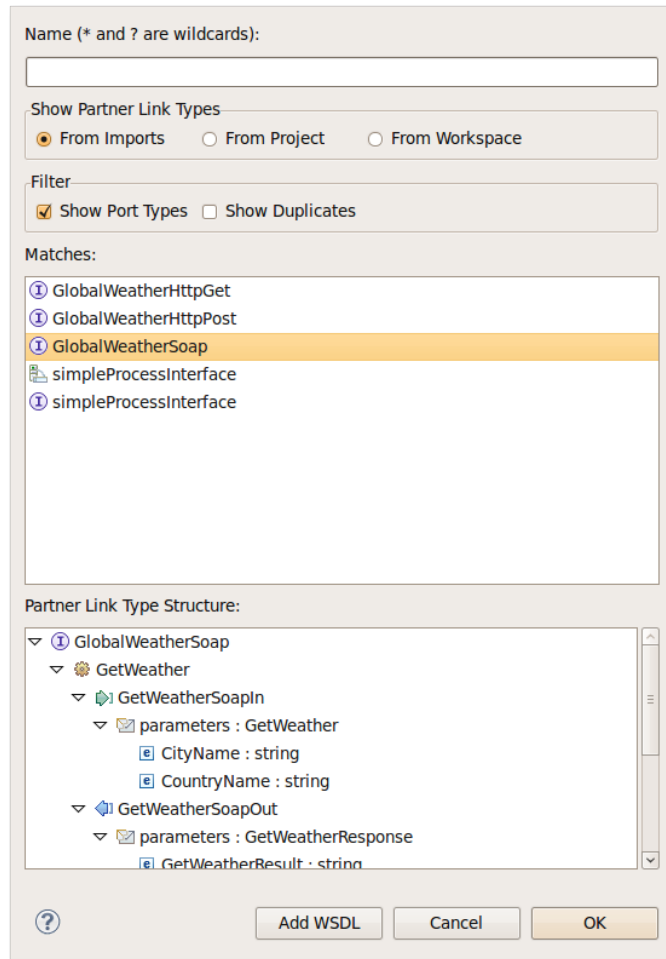


Figure 7: Choosing the PartnerLinkType.

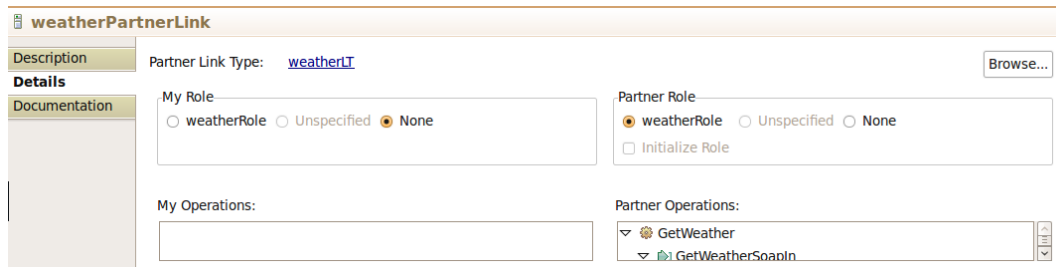


Figure 8: PartnerLink definition done.

You can add now an Invoke action after the Assign (that will be modified) and choose the operation to call.

Finally, you have to modify the input and the output surrounding this new invoke (corresponding to the new input and output of the Weather Web Service):

- You need to modify the input type of the BPEL process WSDL and change it for a sequence of a 2 strings (one for the country name, one for the city name):

Listing 3: Input type of the new BPEL process

```

5 <complexType name="simpleProcessInterfaceRequest">
  <sequence minOccurs="0" maxOccurs="1">
    <element name="CityName" type="string"/>
    <element name="CountryName" type="string" />
  </sequence>
</complexType>

```

- Modify the original Assign action in order to copy the BPEL input into the in parameter of the Web Service:

Listing 4: Modification of the first Assign action.

```

4 <assign validate="no" name="Assign">
  <copy>
    <from>
      <![CDATA[$input.payload/tns:CityName]]>
    </from>
    <to>
      <![CDATA[$weatherPartnerLinkRequest
9      .parameters/ns:CityName]]>
    </to>
  </copy>
  <copy>
    <from>
      <![CDATA[$input.payload/tns:CountryName]]>
14    </from>
    <to>
      <![CDATA[$weatherPartnerLinkRequest
19      .parameters/ns:CountryName]]>
    </to>
  </copy>
</assign>

```

- You can add an Assign action in order to put the Weather Web Service result into the original output:

Listing 5: New assign action after the call of The Weather Web Service.

```

5 <assign validate="no" name="AssignOutput">
  <copy>
    <from>
      <![CDATA[$weatherPartnerLinkResponse
      .parameters/ns:GetWeatherResult]]>
    </from>
    <to>
      <![CDATA[$output.payload/tns:result]]>
10    </to>
  </copy>
</assign>

```

Your new process is ready to run. The expected input is for instance:

Listing 6: Expected input for the BPEL process

```

<les:process xmlns:les="http://lesbegu.com">
  <payload>

```

```

4   <les:CityName>Pau</les:CityName>
    <les:CountryName>France</les:CountryName>
    </payload>
  </les:process>

```

The result is a summary of the weather for this City.

### 3.2 Service Engine Component Configuration

The component configuration is made in its `jbi.xml` (`META-INF/jbi.xml` in the `.zip`). The listing 7 is an example of `jbi.xml`. From line 3 to line 5, attributes of the component are defined :

- `type`: fixed to “service-engine” for Maestro,
- `bootstrap-class-loader-delegation`: does not need to be configured for Maestro.

From line 30 to 69, CDK specific attributes are defined (See CDK 5.0 documentation) :

- `acceptor-pool-size`
- `processor-pool-size`
- `ignored-status`
- `properties-file`
- `notifications`

From line 71 to the end, component specific attributes are defined :

- `explorer`: boolean (Incubation, research feature). If it is set to `true` the Fractal Explorer is launched in order to visualize the process.
- `internal-logger`: boolean. If it is set to `true`, the Debug mode is set to on and the component logs are printed.
- `pool-size`: *integer (To develop for optimization). Pool of instances size.*

Listing 7: `jbi.xml` of the `petals-se-bpel` component

```

4   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <jbi xmlns="http://java.sun.com/xml/ns/jbi" version="1.0">
      <component
9     type="service-engine"
      bootstrap-class-loader-delegation="parent-first">
        <identification>
          <name>petals-se-bpel</name>
          <description>
14          The PEtALS BPEL JBI service engine based on Maestro.
          </description>
        </identification>
        <component-class-name>
14          org.ow2.petals.engine.maestro.bpel.MaestroEngine
        </component-class-name>
        <component-class-path>
          <path-element>
19          petals-cdk-jbidescriptor-2.0-SNAPSHOT.jar
          </path-element>
          <!-- ... -->
        </component-class-path>
        <bootstrap-class-name>
          org.ow2.petals.engine.maestro.bpel.bootstrap.MaestroBootstrap
        </bootstrap-class-name>
24        <bootstrap-class-path>

```

```

    <path-element>
      petals-cdk-jbidescriptor -2.0-SNAPSHOT.jar
    </path-element>
    <!-- ... -->
29 </bootstrap-class-path>
    <petalsCDK:acceptor-pool-size
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
34      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
        3
    </petalsCDK:acceptor-pool-size>
    <petalsCDK:processor-pool-size
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
39      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
        10
    </petalsCDK:processor-pool-size>
    <petalsCDK:ignored-status
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
44      DONE_AND_ERROR_IGNORED
    </petalsCDK:ignored-status>
    <petalsCDK:properties-file
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
54      xmlns:petalsCDK="http://petals.ow2.org/.../version-5"/>
    <petalsCDK:notifications
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
59      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
        false
    </petalsCDK:notifications>
    <petalsCDK:jbi-listener-class-name
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
64      org.ow2.petals.engine.maestro.bpel.listener.JBIListener
    </petalsCDK:jbi-listener-class-name>
69
    <!-- Component specific elements -->
    <bpel-engine:explorer
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
74      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
        false
    </bpel-engine:explorer>
    <bpel-engine:internal-logger
      xmlns:bpel-engine="http://petals.ow2.org/.../version-1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbi="http://java.sun.com/xml/ns/jbi"
      xmlns:petalsCDK="http://petals.ow2.org/.../version-5">
79      true
    </bpel-engine:internal-logger>
84 </component>
</jbi>

```