PETALS-BC-EJB

This document explains how to install, configure and use the petals-bc-ejb JBI component.

PEtALS Team
Olivier FABRE <olivier.fabre@ebmwebsourcing.com>
Charles CASADEI <charles.casadei@ebmwebsourcing.com>

- February 2009 -

Table of Contents

List of Figures

List of Tables

PETALS-BC-EJB

This Binding Component (BC) allows to interact with external Enterprise Java Beans (EJB) running on an external JEE container.

This component has been successfully tested with the following EJB specifications :

- 2.0
- 2.1
- 3.0
- 3.1

On the following JEE container :

- JOnAS
- JBoss
- OpenEJB
- OC4J

This component acts only as a service provider. A JBI message exchange sent to a ServiceEndpoint (mapped to an EJB) is transformed into an EJB call through RMI.

`InOut` and `InOnly` patterns are currently supported.

If you need more informations about EJB3 developement with EasyBeans and deployment on JOnAS feel free to read the EasyBeans documentation, available online at : http://wiki.easybeans.org/xwiki/bin/view/Main/Documentation.

Chapter 1. Features

The petals-bc-ejb component is based on the PEtALS CDK v4, PEtALS-JAXB-Databinding v1.0 and XStream 1.3.1.

It provides the following features :

- Expose EJB (2.0, 2.1, 3.0 & 3.1) as JBI Services.
- Handle JBI security subject by using JAAS Login Modules.
- Binds XML request and response messages to/from Java Objects, by using the PEtALS-JAXB-Databinding library.

Chapter 2. Component configuration

Before installing the bc-ejb component, you must check in your $PETALS_HOME/conf/server.properties configuration file if the property "petals.classloaders.isolated=true" is set and uncommented. The bc-ejb component need the isolated classloaders to work correctly.

The component can be configured through its JBI descriptor file like this :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi" version="1.0">
 <jbi:component type="binding-component" component-class-loader-delegation="parent-first">

  <jbi:identification>
   <jbi:name>petals-bc-ejb</jbi:name>
   <jbi:description>an EJB Binding Component sending messages to local or distant EJB
 instances</jbi:description>
  </jbi:identification>

  <jbi:component-class-name>org.ow2.petals.bc.ejb.EjbBC</jbi:component-class-name>
  <jbi:component-class-path><jbi:path-element/></jbi:component-class-path>

 <jbi:bootstrap-class-name>org.ow2.petals.component.framework.DefaultBootstrap</jbi:bootstrap-class-
name>
  <jbi:bootstrap-class-path><jbi:path-element/></jbi:bootstrap-class-path>

  <petalsCDK:acceptor-pool-size>5</petalsCDK:acceptor-pool-size>
  <petalsCDK:processor-pool-size>10</petalsCDK:processor-pool-size>
  <petalsCDK:ignored-status>NOTHING_IGNORED</petalsCDK:ignored-status>

  <shared-library>petals-sl-ejb</shared-library>


 <petalsCDK:jbi-listener-class-name>org.ow2.petals.bc.ejb.listener.JBIListener</petalsCDK:jbi-
listener-class-name>
 </jbi:component>
</jbi:jbi>
```

This component doesn't have any specific configuration parameters.

You can customize the component configuration by changing the following common parameters.

Table 2.1. Configuration of the component (CDK)

Parameter	Description	Default	Required	Scope
acceptor-pool-size	The size of the thread pool used to accept Message Exchange from the NMR. Once a message is accepted, its processing is delegated to the processor pool thread.	5	Yes	Runtime
processor-pool-size	The size of the thread pool used to process Message Exchanges. Once a message is accepted, its processing is delegated to one of the thread of this pool.	10	Yes	Runtime
performance-notifications	Enable the performance notifications in the component. The CDK proposes to a performance notification feature to the component implementor. If you enable this feature, you must use the related method accessible in the `AbstractComponent` class.	-	No	Runtime
performance-step	When the performance notification feature is enabled, it is possible to define a step on the notifications. When there is an heavy message traffic, it is recommanded to increase this step to avoid performance disturbance.	-	No	Runtime
properties-file	Name of the file containing properties used as reference by other parameters. Parameters reference the property name in the following pattern `${myPropertyName}`. At runtime, the expression is replaced by the value of the property. The value of this parameter is : • an URL • a file relative to the PEtALS installation path • an empty value to stipulate a non-using file	-	No	Installation
ignored-status	When the component receives an acknowledgement message exchange, it can skip the processing of these message according to the type of the acknowledgment. If you decide to not ignore some acknowledgement, the component listeners must take care of them. Accepted values : `DONE_AND_ERROR_IGNORED`, `DONE_IGNORED`, `ERROR_IGNORED` or `NOTHING_IGNORED`	`DONE_AND_ERROR_IGNORED`	Yes	Component
jbi-listener-class-name	Qualified name of the class extending **AbstractJBIListener**	-	Yes	Component
external-listener-class-name	Qualified name of the class extending **AbstractExternalListener**	-	No	Component

Caution

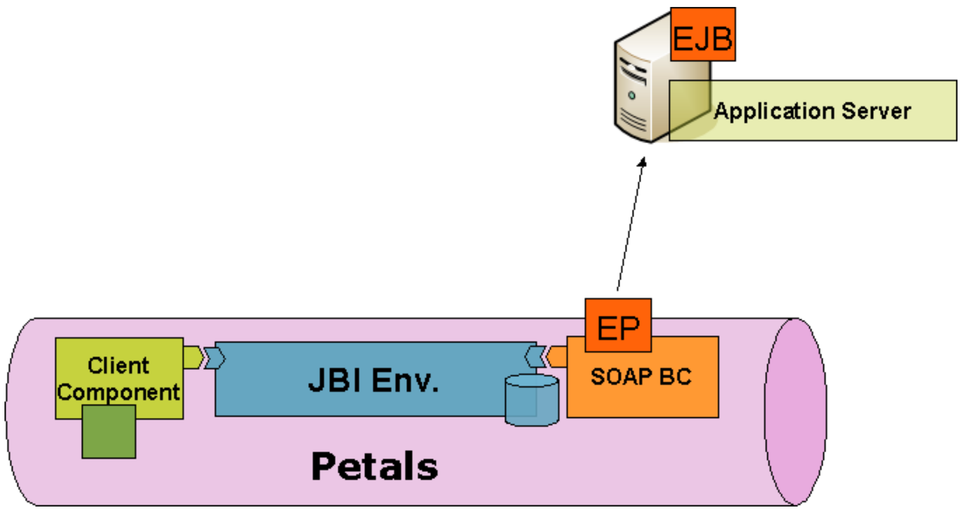
EJB binding component can only handle outgoing message (JBI --> EJB), so you can't specify an external-listener-class-name.

Chapter 3. Service configuration

3.1. Send a JBI message to an external EJB

When a JBI message is received on an endpoint linked to an EJB, the message is transformed into a RMI message, then sent to the linked EJB.

Figure 3.1. Provides an external EJB as a JBI service



The RMI message is created following these steps :

1. The JBI message payload is mapped to Java objects. These objects (and their types) are used as operation parameters for the RMI call. The mapping is done thanks to the PEtALS-JAXB-Databinding library. For more information about XML databinding feel free to read the chapter entitled XML to Java binding.
2. The JBI message exchange operation local part is used as the EJB method to invoke.
3. If a security subject is provided by the JBI message it is used as authentication information during the RMI invokation.

Note

For more information about JAAS read the chapter : JAAS authentication for EJB calls

In order to reach the remote EJB, the component need to get an RMI stub of the EJB from a JNDI server. The JNDI name of the target EJB is defined in the parameter `ejb.jndi.name`.

The external EJB is called and the response is processed by the PEtALS-JAXB-Databinding library and then returned to the JBI environment.

3.2. Service Unit descriptor

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 JBI descriptor for the PEtALS' "petals-bc-ejb" component (EJB).
 Originally created for the version 1.1 of the component.
-->
<jbi:jbi version="1.0"
```

```
  xmlns:ejb="http://petals.ow2.org/components/ejb/version-1.1"
  xmlns:generatedNs="http://application.localisation.watersupply.petals.ow2.org/"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi"
  xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:provides
    interface-name="generatedNs:LocalisationFinderBusinessServicePortType"
    service-name="generatedNs:LocalisationFinderBusinessService"
    endpoint-name="LocalisationFinderBusinessServiceEndpoint">

   <!-- CDK specific elements -->
   <petalsCDK:wsdl>Localisation.wsdl</petalsCDK:wsdl>

   <!-- Component specific elements -->
   <ejb:ejb.jndi.name>LocalisationFinderBusinessService</ejb:ejb.jndi.name>

 <ejb:java.naming.factory.initial>org.jnp.interfaces.NamingContextFactory</
ejb:java.naming.factory.initial>
   <ejb:java.naming.provider.url>jnp://localhost:1099/</ejb:java.naming.provider.url>
   <ejb:ejb.version>2.1</ejb:ejb.version>

 <ejb:ejb.home.interface>org.ow2.petals.watersupply.localisation.application.LocalisationFinderBusinessServic
ejb:ejb.home.interface>
   <ejb:marshalling.engine>jaxb</ejb:marshalling.engine>

   <ejb:security.name />
   <ejb:security.principal />
   <ejb:security.credencials />
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

Table 3.1. Configuration of a Service Unit to expose an EJB onto PEtALS

Parameter	Description	Default	Required
ejb.jndi.name	The JNDI name of the targeted EJB	-	Yes
java.naming.factory.initial	The name of the targeted JNDI Initial Context Factory	-	Yes
java.naming.provider.url	The URL of the targeted JNDI service	-	Yes
ejb.version	Implemention version of the targeted EJB. Supported versions are 2.0, 2.1, 3.0 and 3.1	-	Yes
ejb.home.interface	Fully qualified name of the targeted EJB Home Interface. Used only with ejb 2.0 and 2.1.	-	Only if ejb.version is 2.0 or 2.1
security.name	Fully qualified name of the security module used.	-	No
security.principal	Username	-	No
security.credencials	Password	-	No

Table 3.2. Configuration of a Service Unit to provide a service (CDK)

Parameter	Description	Default	Required
wsdl-imports-download	If false, the external imports declared in the service WSDL won't be downloaded, so they won't be replaced by their content.	True	No
wsdl	Path to the WSDL document describing services and operations exposed by the provided JBI endpoints defined in the SU. The value of this parameter is : • an URL • a file relative to the root of the SU package If not specified, a basic WSDL description is automaticaly provided by the CDK.	-	No
timeout	Timeout in milliseconds of a synchronous send. this parameter can be used in conjunction with the `sendSync(Exchange exchange)` method of the Listeners. Set 0 for an infinite timeout.	-	No
org.ow2.petals.messaging.provider.noack	Check PEtALS container document for further details. This propety activates the bypass of acknowledgment messages destinated to this SU.	-	No

Table 3.3. Interceptors configuration for component (CDK)

Parameter	Description	Default	Required
class	Name of the interceptor class. This class must extend the abstract class org.ow2.petals.component.common.interceptor.Interceptor. This class have to be present in the component classloader, in one of the component jars or in a shared library jar.	-	Yes
name	Name of the interceptor. This name will be used for activation or additional configuration in the SU.	-	Yes
active	Interceptor is activated for all deployed SUs.	-	Yes

3.3. Service Unit content

The service unit must contain a JAR archive including the EJB Interface (and EJB Home Interface for a 2.x EJB) and all specific Java classes used by this interface.

It is also highly recommended to provide a WSDL description of your EJB interface. This WSDL description will be used as Service Description for the JBI Endpoint linked to your EJB.

```
my-su-ejb.zip
+META-INF
    - jbi.xml
-my-ejb.jar
-my-ejb-dependency1.jar
-my-ejb-dependency2.jar
```

Chapter 4. Packaging EJB container RMI client libraries

Since the petals-bc-ejb is a generic binding component that allows to call Enterprise Java Beans running on different kind of application servers, you must add your application specific RMI client libraries to the component classpath. There are three solutions to add the libraries to do so :

- add the libraries directly in the component classpath (bad)
- add the libraries to each deployed service unit (average)
- add the libraries to a shared library deployed before component startup (good)

By default this component uses a shared library called "petals-sl-ejb" which must contains the RMI client libraries of the EJB targeted EJB container with its JEE EJB specification.

A shared library archive may look like this :

```
petals-sl-ejb.zip
+META-INF
    - jbi.xml
-my-ejb-container-rmi-client.jar
-my-ejb-container-rmi-client-dependency1.jar
-my-ejb-container-rmi-client-dependency2.jar
-my-ejb-container-ejb-specification.jar
```

Adding jar files to the archive is not sufficient. Each jar contained by the shared library must be declared within its JBI descriptor.

Here is an exemple of such files (jbi.xml file) :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:petals="http://petals.ow2.org/extensions"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi" version="1.0">

 <jbi:shared-library class-loader-delegation="self-first" version="my-ejb-container-versionXYZ">
  <jbi:identification>
   <jbi:name>petals-sl-ejb</jbi:name>
   <jbi:description>SharedLibrary EJB for my-ejb-container-versionXYZ</jbi:description>
  </jbi:identification>

  <jbi:shared-library-class-path>
   <jbi:path-element>my-ejb-container-rmi-client.jar</jbi:path-element>
   <jbi:path-element>my-ejb-container-rmi-client-dependency1.jar</jbi:path-element>
   <jbi:path-element>my-ejb-container-rmi-client-dependency2.jar</jbi:path-element>
   <jbi:path-element>my-ejb-container-ejb-specification.jar</jbi:path-element>
  </jbi:shared-library-class-path>
 </jbi:shared-library>
</jbi:jbi>
```

Finally, to add this shared library to the component classpath you must add the following section to the JBI descriptor of the component (here with the default shared library for the EJB component) just after the end of the "bootstrap-class-path" element:

```
<shared-library>petals-sl-ejb</shared-library>
```

Chapter 5. XML to Java binding

Since the JBI message payload is a XML message, the component must provide a way to transform Java objects into XML (marshalling) an XML to Java objects (unmarshalling). The message payload containing an EJB call is unmarshalled to Java objects that will be used as method parameters for the EJB call through RMI. The EJB response is intercepted by the component and then marshalled to an XML payload.

This marshalling / unmarshalling process is provided by the PEtALS-JAXB-Databinding library.

5.1. PEtALS JAXB Databinding

The PEtALS-JAXB Databinding library provides a way to marshall / unmarshall Java objects to XML. This library uses a WSDL file (generated from your service class with Apache-CXF or OW2-Java2EasyWSDL from the EasyWSDL toolbox) to bind Java classes to XML tags.

5.1.1. Request message

The incoming JBI message payload is unmarshalled by JAXB using the WSDL provided in the service unit. XML messages are transformed to Java Objects which are used to perform a RMI call on the EJB.

Figure 5.1. An EJB call request which conforms to the provided WSDL

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:q0="http://application.localisation.watersupply.petals.ow2.org/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <q0:getBureauDistributeurInfoByCommuneId>
   <q0:arg0>452</q0:arg0>
  </q0:getBureauDistributeurInfoByCommuneId>
 </soapenv:Body>
</soapenv:Envelope>
```

5.1.2. Response message

The EJB response is intercepted by the component and then marshalled by JAXB conforming to the provided WSDL.

Figure 5.2. An EJB response marshalled conforming to the WSDL

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <getBureauDistributeurInfoByCommuneIdResponse
    xmlns="http://businessinfo.localisation.watersupply.petals.ow2.org"
    xmlns:ns2="http://application.localisation.watersupply.petals.ow2.org/">
   <ns2:return>
    <BureauDistributeurInfo>
     <code>code 0</code>
     <id>452</id>
     <libelle>libelle 0</libelle>
    </BureauDistributeurInfo>
    <BureauDistributeurInfo>
     <code>code 2</code>
     <id>452</id>
     <libelle>libelle 2</libelle>
    </BureauDistributeurInfo>
    <BureauDistributeurInfo>
     <code>code 3</code>
     <id>452</id>
     <libelle>libelle 3</libelle>
    </BureauDistributeurInfo>
   </ns2:return>
  </getBureauDistributeurInfoByCommuneIdResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Chapter 6. JAAS authentication

The EJB binding component is JAAS enabled : it can handle security subjects from your JBI platform to your application server to perform authentication and role based EJB method restrictions.

Caution

When using JAAS (or any security feature) you MUST ensure that all the JVM are compliant. In other words, the JVM running PEtALS MUST be fully compliant with the one running your application server. Both JVM must came from the same vendor, using the same kind of architecture (32 bits or 64 bits), cryptography libraries and so on.

6.1. JAAS configuration

JAAS authentication is based on a configuration file which specifies all the login modules to be used during the authentication process, as shown below.

```
jonas {
    // Login Module to use for the example jaasclient.

    //First, use a LoginModule for the authentication
    org.ow2.petals.bc.ejb.security.WSSUserPasswordLoginModule required
 org.ow2.petals.users="users.properties" org.ow2.petals.roles="roles.properties";

    // Use the login module to propagate security to the JOnAS server
    // globalCtx is set to true in order to set the security context
    // for all the threads of the client container instead of only
    // on the current thread.
    // Useful with multithread applications (like Swing Clients)
    org.objectweb.jonas.security.auth.spi.ClientLoginModule  required  globalCtx="true";
};
```

In this file, only one configuration “jonas” (which is the configuration identifier) is defined. You can define several configurations in the same JAAS configuration file.

Caution

PEtALS must be configured to use this file as default JAAS configuration file at startup. To do so, you must set up the JVM property “java.security.autho.login.config” to the absolute path of your JAAS configuration file. Assuming that “PETALS_HOME” is an environment variable pointing onto your PEtALS installation folder and your JAAS configuration file is called “jaas.conf” and resides in your PEtALS installation folder, you can set this JVM property by adding the following option to the PEtALS startup command –Djava.security.auth.login.config==”$PETALS_HOME/jaas.conf”.

6.2. Login module configuration

In your JAAS configuration file you can specify a list of LoginModule, which will be used for the whole authentication process.

Note

You can write your own LoginModule by implementing the javax.security.auth.spi.LoginModule interface. To do so feel free to read the JAAS LoginModule developer’s guide.

For instance in the previous JAAS configuration file, two LoginModule were defined. The first one (org.ow2.petals.bc.ejb.security.WSSUserPasswordLoginModule) is used to make the authentication (based on user/password informations) and the second one, (org.objectweb.jonas.security.auth.spi.ClientLoginModule) is used to propagate the LoginContext to the application server (JOnAS).

Caution

LoginModule classes must be included in the service unit.

6.3. JAAS resources

- **Sun.** *JAAS Reference* , available online at : http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html
- **Sun.** *JAAS Tutorials* , available online at : http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/tutorials/index.html
- **Sun.** *LoginModule Developer's Guide*, available online at : http://java.sun.com/javase/6/docs/technotes/guides/security/jaas/JAASLMDevGuide.html
- **Bhattacharjee Rahul.** *Authentication using JAAS*, available online at : http://www.javaranch.com/journal/2008/04/Journal200804.jsp#a6