PETALS-BC-EJB

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

PEtALS Team

FABRE Olivier <olivier.fabre@ebmwebsourcing.com>

- December 2007 -

(CC) EBM WebSourcing - This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/

Table of Contents

List of Figures

List of Tables

PETALS-BC-EJB

This binding component allows to interact with external EJB (Enterprise Java Beans). EJB 2.1 and EJB 3 on Jonas and Jboss application servers have been tested.

A JBI MessageExchange sent to a ServiceEndpoint (mapped to an EJB) is transformed into an EJB call through RMI. InOut and InOnly patterns are supported.

If you want more information about EJB3 development with Easybeans and deployment on Jonas see : http://wiki.easybeans.org/xwiki/bin/view/Main/Documentation

Exemples of EJB3 are available in PEtALS SVN repository under "petals-demos/petals-ejb" folder.

Chapter 1. Features

The petals-bc-ejb is based on the petals-cdk v3.0-beta2.

RMI client libraries are not included in the petals-bc-ejb component libraries because it depends on the application server where your EJBs are running. You can bundle the client libraries for your application server in a Shared Library (see ejb-chapter4).

It provides support of :

- Expose EJB as JBI Services. Has been tested with EJB 2.1 and EJB 3.
- Can handle JBI security subject by using JAAS Login Modules.
- Bind xml input and output tags to Java object thanks to Xstream. Xstream aliases can be used to simplify XML messages

Chapter 2. Component Configuration

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

```
<jbi version="1.0" xmlns='http://java.sun.com/xml/ns/jbi'
 xmlns:petals='http://petals.ow2.org/extensions'
 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
 <component type="binding-component"
  component-class-loader-delegation="parent-first">
  <identification>
   <name>petals-bc-ejb</name>
   <description>
    an EJB Binding Component sending messages to local or
    distant EJB instances
   </description>
  </identification>
  <component-class-name>
   org.ow2.petals.bc.ejb.EjbBC
  </component-class-name>
  <component-class-path></component-class-path>
  <bootstrap-class-name>
   org.ow2.petals.component.framework.DefaultBootstrap
  </bootstrap-class-name>
  <bootstrap-class-path></bootstrap-class-path>
  <!-- Uncomment it to add your specific ejb implementation
   library as a Shared Library -->
  <!-- <shared-library>sl-jonas-client</shared-library> -->
  <petals:jbi-listener-class-name>
   org.ow2.petals.bc.ejb.listener.JBIListener
  </petals:jbi-listener-class-name>
 </component>
</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	-	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

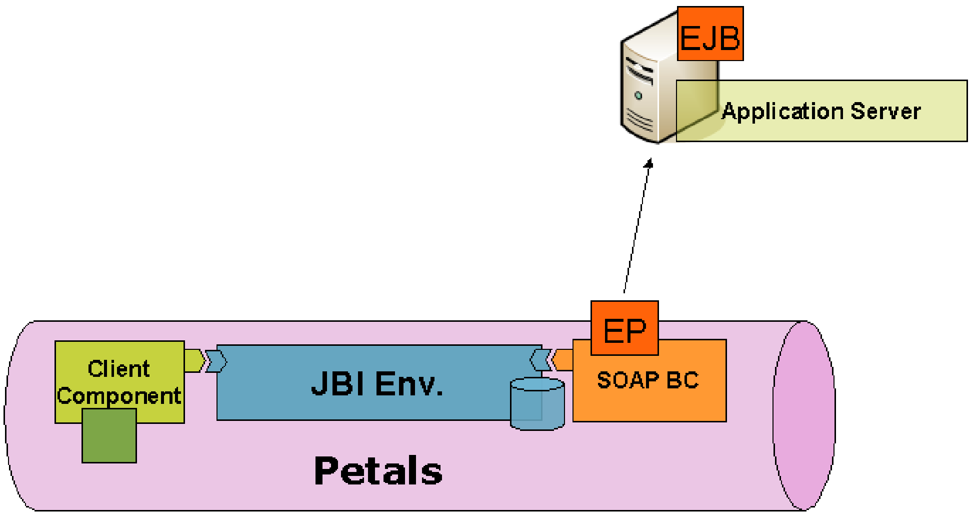
EJB binding component can only handle outgoing message (JBI --> EJB), so you can't specify an external-listener-class-name.

Chapter 3. Service Configuration

3.1. Send a JBI message to an external EJB

PROVIDE SERVICE : Expose an external EJB in the JBI environment

Figure 3.1. Provides an external EJB as a JBI service



The petals-bc-ejb component can expose an external EJB as a JBI ServiceEndpoint. This is done by deploying a Service Unit on it.

When a message is received on a EJB linked endpoint from the JBI environment, it is transformed into an RMI message and sent to the linked EJB. The JNDI name of the targeted EJB is defined in the "ejb.jndi.name" extension of the deployed Service Unit.

The RMI message is created like this :

- 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 XStream api. For more information about XStream see XStream documentation.
- The JBI message exchange operation local part is used as operation name for the RMI call
- If a security subject is provided in the JBI message it is used as authentication information during the RMI call. For more information about JAAS Authentication see Chapter 6, *JAAS Authentication for EJB calls*.

The external EJB is called and the response is processed (with XStream) and returned to the JBI environment.

3.1.1. Service Unit descriptor

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

```
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:extensions="http://petals.ow2.org/extensions"
 xmlns:sa="http://petals.ow2.org/SA"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi" version="1.0">
<jbi:services binding-component="true">
  <jbi:provides interface-name="sa:ejbSAInterface"
   service-name="sa:ejbSAService" endpoint-name="ejbSAEndpoint">
   <extensions:wsdl>SA.wsdl</extensions:wsdl>
   <extensions:params>
    <extensions:param name="ejb.jndi.name">
      SA
    </extensions:param>
    <extensions:param name="java.naming.factory.initial">
      org.objectweb.carol.jndi.spi.IRMIContextWrapperFactory
    </extensions:param>
    <extensions:param name="java.naming.provider.url">
      rmi://localhost:1099
    </extensions:param>
    <extensions:param name="ejb.version">
      2.1
    </extensions:param>
    <extensions:param name="ejb.home.interface">
      com.ebmwebsourcing.petals.usecase.soap2ejbsecurity.SAHome
    </extensions:param>
   </extensions:params>
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

Table 3.1. Service Unit attributes to provide services

Attribute	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 2.1 and 3.0.		Yes
ejb.home.interface	Used only if ejb.version is 2.1. It's the fully qualified name of the targeted EJB Home Interface.		No (Required if ejb.version is 2.1)

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

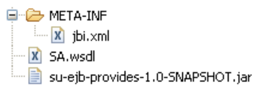
Parameter	Description	Default	Required
wsdl	Path to the WSDL document describing services and operations exposed by the provided JBI endpoints defined in the SU. The value of this parameter is : • an URL • a file relative to the PEtALS installation path If no wsdl path is specified, a basic description isl 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 Service Unit (CDK)

Parameter	Description	Default	Required
name	Name of the interceptor to use. It must refer to a component interceptor name.	-	Yes

3.1.2. Service Unit Content

Figure 3.2. Service unit folder



```
META-INF
    jbi.xml
SA.wsdl
su-ejb-provides-1.0-SNAPSHOT.jar
```

The service unit must contain a Jar archive including EJB interface (and EJB Home Interface for a 2.1 EJB) and all specific Java classes used by this interface (like Java beans...).

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

Chapter 4. Application Server specific client libraries

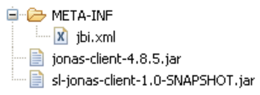
The petals-bc-eb is a generic binding component that allows to call EJBs running on different application servers. You must add your application server specific RMI client libraries to the component classpath to allow it to call your running EJBs. There are two solutions to add the libraries to the component classpath.

The first solution is to change the JBI descriptor (jbi.xml file) of your petals-bc-ejb component : simply add a path element in the component classpath section.

The second solution is to create a shared library. It's probably the best solution.

Here is an exemple of a shared library archive :

Figure 4.1. Shared library folder



Here is an exemple of shared library JBI descriptor (jbi.xml file) :

```
<jbi:jbi xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0">
 <jbi:shared-library>
  <jbi:identification>
   <jbi:name>sl-jonas-client</jbi:name>
   <jbi:description>
    A shared library including jonas-client jar.
   </jbi:description>
  </jbi:identification>
  <jbi:shared-library-class-path>
   <jbi:path-element>jonas-client-4.8.5.jar</jbi:path-element>
   <jbi:path-element>
    sl-jonas-client-1.0-SNAPSHOT.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 your petals-bc-ejb JBI descriptor (just after the bootstrap-class-path section) :

```
<shared-library>sl-jonas-client</shared-library>
```

Chapter 5. XML to Java Binding

The JBI message payload is an XML message. It must be unmarshalled to Java objects that will be used as method parameters for the EJB RMI call. In the other way (EJB response), the Java object response must be marshalled to an XML message payload.

This marshalling/unmarshalling process is provided by a Java library : XStream.

5.1. Outgoing message (unmarshalling)

The structure of the outgoing XML message payload must conform to the following pattern :

```
<params>
  <firstParams>firstValue<firstParams>
  <secondParams>secondValue<secondParams>
  <thirdParams>thirdValue<thirdParams>
  ...
<params>
```

The names of the different XML tags are not restricted the ones given as exemple. The name of the root element can be "foo", "bar" or anything else because it isn't used during the unmarshalling process.

Parameter tag names will be used by XStream to instantiate Java classes that will be used as EJB RMI call parameters. XStream will search in the SU classpath for a class with a fully qualified name similar to the XML tag name. For exemple if the XML tag name is "com.foo.Bar", the mapped class will be "com.foo.Bar". Here is an exemple of an outgoing message :

```
<params>
  <com.foo.Bar>
    <name>foo<name>
    <address>bar<address>
  <com.foo.Bar>
  <string>foo<string>
  <integer>22<integer>
<params>
```

In this exemple, the method call parameters will be : a "com.foo.Bar" object (including two strings attributes "name" and "address"), "foo" and "22". The parameters types will be : "com.foo.Bar", "java.lang.String" and "java.lang.Integer". If you want to simplify XML tag names you can use XStream aliases.

5.2. Incoming message (marshalling)

EJB Java object response will be automaticaly marshalled to XML response payload by the EJB BC. The marshalling process is similar to the unmarshalling process. Fully qualified class name will be used as XML tag name. For exemple if the response class name is "com.foo.Bar", the XML tag name will be "com.foo.Bar". Here is an exemple of an EJB response :

```
<com.foo.Bar>
  <name>foo<name>
  <address>bar<address>
<com.foo.Bar>
```

In this exemple, EJB response is a "com.foo.Bar" object, with two attributes ("name" and "address"). If you want to simplify XML tag names you can use XStream aliases.

5.3. XStream aliases

By default, XStream uses fully qualified class name as name for XML tags. So, if there are a lot of complex types in RMI requests or responses, XML document can become very verbose.

To simplify XML documents you can use XStream aliases. An alias is an abreviation for a class name. For exemple, you can define a simple alias like "bar" linked to the "com.foo.Bar" class name. In this case, the XML tag name will be "bar" instead of "com.foo.Bar".

XStream is bundled with some predefined aliases for primitive Java types like java.lang.String (alias "string"), java.lang.Integer (alias "integer")...

To define aliases for a specific JBI Endpoint, you can add extensions to the related service unit descriptor (jbi.xml). Here is an exemple of alias extensions :

```
<!-- Xstream aliases -->
<extensions:param name="xstream.alias.address">
 com.ebmwebsourcing.petals.usecase.soap2ejbsecurity.model.Address
</extensions:param>
<extensions:param name="xstream.alias.id">
 java.lang.Integer
</extensions:param>
<extensions:param name="xstream.alias.street">
 java.lang.String
</extensions:param>
```

In this exemple, three aliases have been defined :

XML tag name	Class name
address	com.ebmwebsourcing.petals.usecase.soap2ejbsecurity.model.Address
id	java.lang.Integer
street	java.lang.String

For more information about XStream aliases follow this link.

Chapter 6. JAAS Authentication for EJB calls

The EJB binding component is JAAS enabled : you can handle security subject from your JBI platform to your Application Server to perform authentication and role based EJB method restriction.

6.1. JAAS configuration

JAAS authentication is based on a configuration file where you can specify all login modules that have to be used to perform the whole authentication. Here is an exemple of a JAAS configuration file :

```
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";
};
```

Only one configuration is defined in this file. "jonas" is the configuration identifier. You can define multiple configuration in a same JAAS configuration file.

Petals must be setup to use this file has default JAAS configuration file at startup. To do it, you must add a JVM properties "java.security.auth.login.config" defined to the absolute location of your JAAS configuration file : assuming that "PETALS_HOME" is an environment variable pointing on your PEtALS installation folder and your JAAS configuration file is called "jaas.conf" and resides in your PEtALS installation folder, you can specify the JVM property by adding the following option to the PEtALS startup command -Djava.security.auth.login.config=="%PETALS_HOME%/jaas.conf".

6.2. Login Modules configuration

In your JAAS configuration file you can specify the list of LoginModules. Each specified login modules will be used for the whole authentication process.

You can create your own LoginModule by implementing javax.security.auth.spi.LoginModule interface. In the previous JAAS configuration file exemple, two login modules are defined. The first one (org.ow2.petals.bc.ejb.security.WSSUserPasswordLoginModule) is used to make the authentication (based on user/ password information) and the second one (org.objectweb.jonas.security.auth.spi.ClientLoginModule) is used to propagate the login context to the application server (Jonas). Login module classes must be in the SU class path to allow authentication during EJB call. For more information about JAAS LoginModule implementation follow this link.

To specify the JAAS configuration to use for JAAS authentication during an EJB call, you must add a "security.name" extension to the service unit descriptor (jbi.xml). For exemple, if I want to use the "jonas" JAAS configuration defined in the previous exemple, I must add the following SU extension :

```
<extensions:param name="security.name">
 jonas
</extensions:param>
```