PEtALS-BC-JMS

This document explain how to install and configure the petals-bc-jms JBI component.

PEtALS Team
Adrien Louis <adrien.louis@ebmwebsourcing.com>
Nicolas Salatge <nicolas.salatge@ebmwebsourcing.com>

- June 2007 -

Table of Contents

List of Figures

List of Tables

PETALS-BC-JMS

This binding component allows to interact with an external JMS Destination (queue or topic). Service Units are used to map a JMS destination to a JBI ServiceEndpoint. A JBI `MessageExchange` sent to a ServiceEndpoint (mapped to a JMS destination) is transformed into a JMS `TextMessage` and sent to the corresponing JMS destination. A JMS message sent to a JMS Destination is transformed into a JBI MessageExchange and sent to the corresponding JBI ServiceEndpoint.

Caution

Only JMS TextMessage are recognized by the petals-binding-jms component.

Caution

The service operation defined in the JBI MessageExchange is mapped to the `operation` String property of the JMS TextMessage (both in consumming and providing mode).

Caution

If the JMS BC must use a JMS Provider other than Joram, this provider client libraries must be in the Petals classpath.

Chapter 1. Component Configuration

The following attributes can be set during the installation phase to configure the component, using the params element of the `jbi-install-component` ANT task:

no configuration for this component

Table 1.1. Component installation configuration attributes

Attribute	Description	Default	Required

Table 1.2. Advanced configuration of the component

Parameter	Description	Default	Required
pool-size	Number of threads listening to messages coming from the JBI container (JBIListeners). Int number >= 1	0	No
ignored-status	Status of messages exchanges that component must ignore. Accepted values : **DONE_AND_ERROR_IGNORED**, **DONE_IGNORED**, **ERROR_IGNORED** or **NOTHING_IGNORED**	DONE_AND_ERROR_IGNORED	No
jbi-listener-class-name	Fully qualified name of the class extending **AbstractJBIListener**		Yes
external-listener-class-name	Fully qualified name of the class extending **AbstractExternalListener**		No
properties-file	Name of the file containing values of keys used as reference by other parameters. To be able to configure a service-unit, you will use a key that has its value hosted by the component (ie. CDK documentation). The value of this parameter is : • whether an URL, • or a file relative to the directory defined by the environment variable PETALS_HOME.		No

Table 1.3. Interceptors configuration in the component

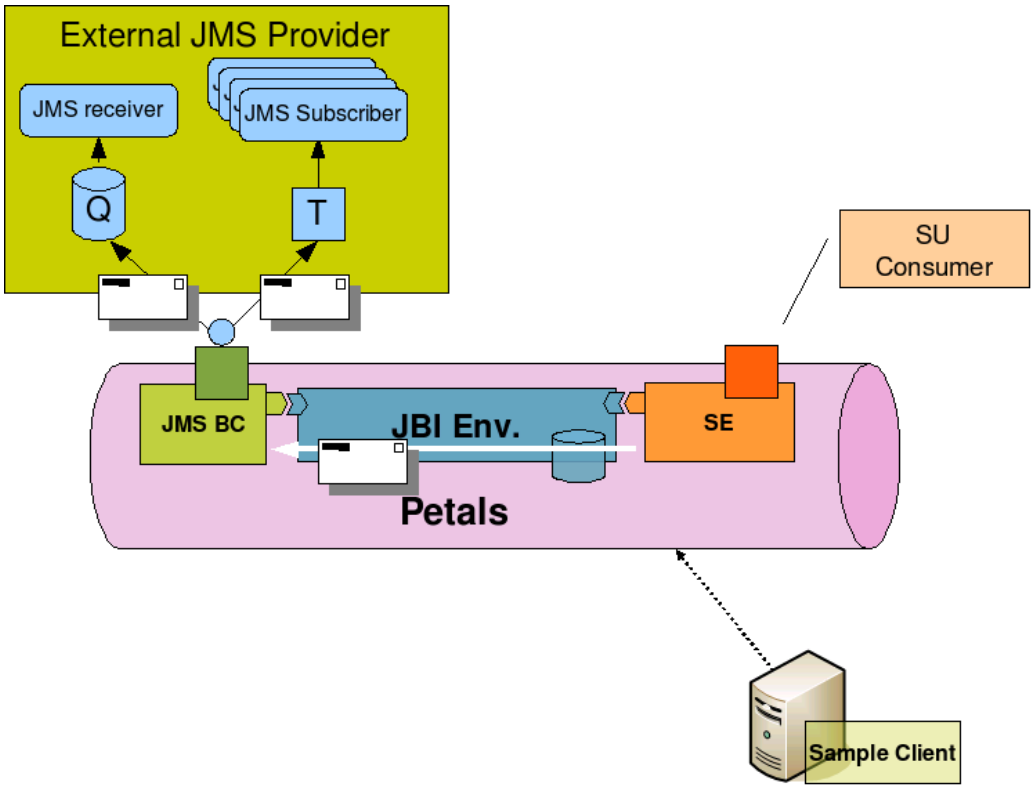
Parameter	Description	Default	Required
class	Name of the interceptor class. This class must extend the abstract class org.objectweb.petals.component.common.interceptor.Interceptor. This class have to be present in the classloader, in component or CF or in a shared library.		Yes
name	Name of the interceptor. This name will be used for additional configuration in the SU.	class name	No
active	Interceptor is active for all SU.	true	No

Chapter 2. Service Configuration

2.1. Send or publish a JBI message to a JMS Queue or Topic

PROVIDE SERVICE : Expose an external service in the JBI environment

Figure 2.1. provides a JMS queue or topic as a JBI service



The petals-bc-jms component can expose as a JBI ServiceEndpoint an external JMS Queue or JMS Topic. This is done by deploying a Service Unit on it.

When a message is received from the JBI environment, it is transform into a JMS TextMessage and sent or published on the Queue or Topic.

Caution

All MessageExchange patterns are allowed. But, due to the JMS paradigm, all exchanges are processed like InOnly exchanges. For exemple, an InOut exchange returns a default message if JMS message is sent succesfully.

Caution

The external Queue or Topic referenced by the Service Unit doesn't have to be available before you start the Service Unit. But the JNDI server where Queue or Topic are registered, must be available before you deploy the Service Unit.

Caution

The NormalizedMessage properties are automaticaly converted as JMS message properties. Be careful, Weblogic doesn't allow "." character in JMS properties names.

2.1.1. Service Unit descriptor

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

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:petals="http://petals.objectweb.org/extensions"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi" version="1.0">
 <jbi:services binding-component="false">
  <jbi:provides interface-name="petals:JMSSampleProviderInterface"
   service-name="petals:JMSSampleProviderService"
   endpoint-name="JMSSampleProviderEndpoint">

   <petals:params>
    <petals:param name="address">scn://localhost:26400</petals:param>
    <petals:param name="destination-name">queueProvider</petals:param>
    <petals:param
 name="initial-context-factory">fr.dyade.aaa.jndi2.client.NamingContextFactory</petals:param>
    <petals:param name="connection-factory">qcf</petals:param>
    <petals:param name="user">anonymous</petals:param>
    <petals:param name="password">anonymous</petals:param>
    <petals:param name="transacted">false</petals:param>
   </petals:params>
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

JMS communication attributes :

Table 2.1. Service Unit attributes to provide services

Attribute	Description	Default	Required
provides	Name of the JBI service that will be activated to expose the JMS Destination into the JBI environment. interface (qname), service (qname) and endpoint (string) name are required.		Yes
address	Address composed of the URL provider (JNDI access)		Yes
destination-name	The JMS destination name where messages will be sent (Queue or Topic JNDI name)		Yes
initial-context-factory	The initial-context-factory class name, used to create an InitalContext.		Yes
connection-factory	name of the JMS ConnectionFactory registered.		Yes
user	User name to access the JMS Destination.	""	No
password	Password to access the JMS Destination.	""	No
transacted	JMS communication transacted mode. true or false.	"false"	No
max-active	Controls the maximum number of JMS connections that can be borrowed from the pool at one time. When non-positive, there is no limit to the number of connections that may be active at one time. When maxActive is exceeded, the pool is said to be exhausted.	"10"	No
max-idle	Controls the maximum number of JMS connections that can sit idle in the pool at any time. When negative, there is no limit to the number of connections that may be idle at one time.	"5"	No
max-wait	If a positive maxWait value is supplied, the JMS component will wait for at most that many milliseconds to retrieve an available JMS connection. If maxWait is non-positive, the component will wait indefinitely.	"10000"	No
time-between-eviction-runs-millis	Indicates how long the eviction thread should sleep before "runs" of examining idle connections. When non-positive, no eviction thread will be launched.	"10000"	No
min-evictable-idle-time-millis	Specifies the minimum amount of time that a connection may sit idle in the pool before it is eligable for eviction due to idle time. When non-positive, no connections will be dropped from the pool due to idle time alone.	"2000"	No
test-while-idle	Indicates whether or not idle connections should be validated. Connections that fail to validate will be dropped from the pool.	"true"	No

Extra attributes :

Table 2.2. Advanced configuration of Service Unit (provides elements)

Parameter	Description	Default	Required
wsdl	path to a wsdl file describing services and operations offered by an endpoint activated by the SU. **This extension is only usable with provides fields.** The path can be a url "http" or "file" or relative to the r oot directory of the SU archive. Ex : "file:///user/ofabre/test.wsdl" or "/WSDL/test.wsdl" If no wsdl path is specified, a simplified description will automaticaly be written by the CF.		No

Table 2.3. Interceptors configuration in the Service Unit

Parameter	Description	Default	Required
name	Name of the interceptor to use. That's the name defined in the component.		Yes

2.1.2. Service Unit content

The Service Unit has to contain the following elements, packaged in an archive:

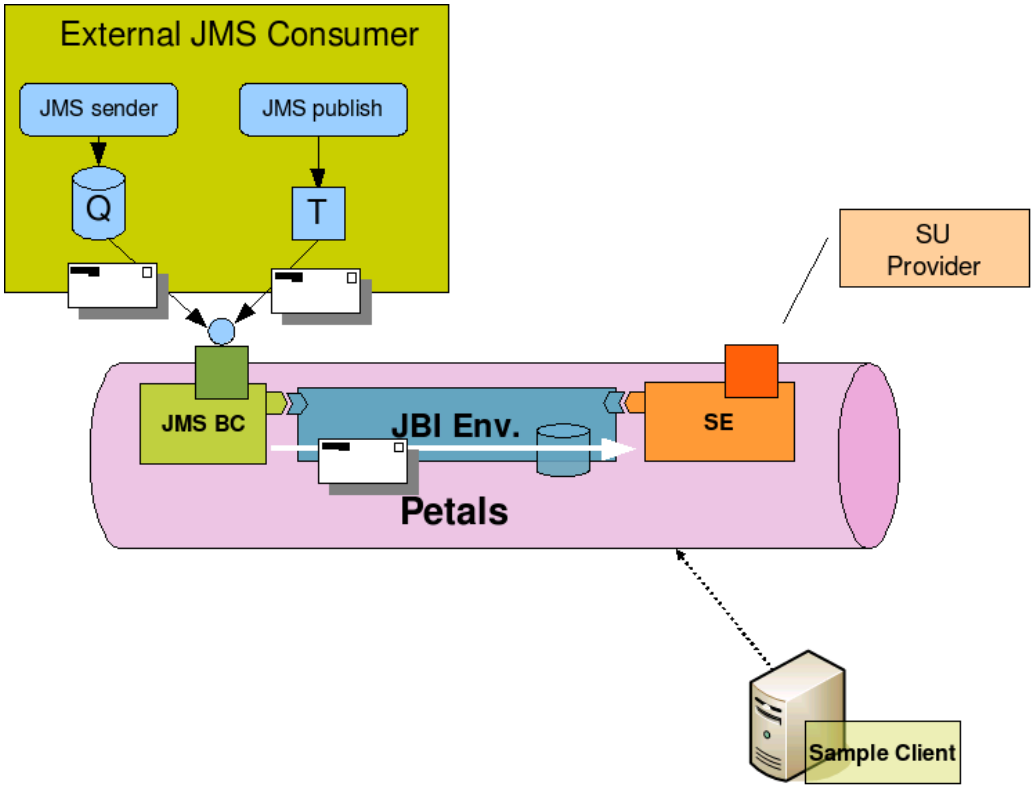
- The META-INF/jbi.xml descriptor file, has described above,
- An optional wsdl file describing the related service

```
service-unit.zip
+ META-INF
    - jbi.xml (as defined above)
    - service.wsdl (optional)
```

2.2. Send a JBI message from a received or published JMS Message

CONSUME SERVICE : Expose an internal service outside the JBI environment

Figure 2.2. Consumes a JBI service on JMS message



The petals-bc-jms component can listen to an external JMS Queue or JMS Topic and send the message to a JBI ServiceEndpoint. We say that the component consumes the JBI service.

When a message is received from the JMS server (the component listens to the JMS Queue or has subscribed to a JMS Topic), it is transform into a JBI Message and sent to the JBI ServiceEndpoint configure in the Service Unit.

Caution

All MessageExchange patterns are allowed. But, due to the JMS paradigm, all exchanges are processed like InOnly exchanges.

Caution

The external Queue or Topic referenced by the Service Unit has to be available before you start the Service Unit.

2.2.1. Service Unit descriptor

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

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:petals="http://petals.objectweb.org/extensions"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi" version="1.0">
 <jbi:services binding-component="false">
  <jbi:consumes interface-name="{http://petals.objectweb.org}HelloworldInterface">
   <petals:params>
    <petals:param name="address">scn://localhost:26400</petals:param>
    <petals:param name="destination-name">queueConsumer</petals:param>
    <petals:param
 name="initial-context-factory">fr.dyade.aaa.jndi2.client.NamingContextFactory</petals:param>
    <petals:param name="connection-factory">qcf</petals:param>
    <petals:param name="user">anonymous</petals:param>
    <petals:param name="password">anonymous</petals:param>
    <petals:param name="transacted">false</petals:param>
    <petals:param name="operation">printMessage</petals:param>
   </petals:params>
  </jbi:consumes>
 </jbi:services>
</jbi:jbi>
```

JMS communication attributes :

Table 2.4. Service Unit attributes to consume services

Attribute	Description	Default	Required
consumes	Name of the JBI service that will be called into the JBI environment. When a JMS message is received. Only the interface (qname) name can be proviTo ded (the container will choose a ServiceEndpoint for this interface), or you can only set service (qname) and endpoint (string) names, without the interface name.		Yes
address	Address composed of the URL provider (JNDI access) and the destination name to listen to.		Yes
destination-name	The JMS destination name where messages will be sent (Queue or Topic JNDI name)		Yes
initial-context-factory	The initial-context-factory class name, used to create an InitalContext.		Yes
connection-factory	name of the JMX ConnectionFactory registered.		Yes
user	User name to access the JMS Destination.	""	No
password	Password to access the JMS Destination.	""	No
transacted	JMS communication transacted mode. true or false.	false	No
operation	JMS operation property		Yes

Extra attributes :

Table 2.5. Advanced configuration of Service Unit (consumes elements)

Parameter	Description	Default	Required
mep	Message exchange pattern abbreviation. This parameter can be user in conjunction with a method of the Listeners : **createMessageExchange(Extensions extensions)**. This merthod returns a MessageExchange corresponding to the type of the specified pattern. Admitted values are : **InOnly**, **RobustInOnly**, **InOptionalOut** et **InOut**		Yes
operation	Operation to call on a service. This parameter can be used in conjuntcion with the sendXXX methods of the Listeners. If no operation is specified in the MessageExchange to send, this parameter will be used.		Yes
timeout	Timeout in milliseconds in a synchroneous send. this parameter can be used in conjunction with the **sendSync(MessageExchange exchange)** method of the Listeners. With this, a synchroneous send is done with this timeout value. 0 for no timeout int number >= 0 for a timeout	0	No
org.objectweb.petals.routing.strategy	This property defines the routing strategy. Two kind of strategy can be defines: highest or random. The others parameters represents respectively the local ponderation, the ponderation of the remote active endpoint and the ponderation of the remote inactive endpoint. The 'random' strategy chooses an endpoint in function of defined ponderations. The endpoints that have the strongest ponderation can be more easely choose in comparison with the others. The 'highest' strategy chooses the first endpoint in the list that have the strongest ponderation.		No
org.objectweb.petals.transporter.compress	The payload of a MessageExchange is an XML file. It can be interesting to compress it before messages are exchanged between two PEtALS nodes. Values are `true` or `false`. True activated the compression of the content of the message.		No
org.objectweb.petals.mep.inbound.mode	All JBI exchanges ended by a message containing a `DONE` or `ERROR` status. The consumer must accept those messages, otherwise they are accumulated in the NMR. Moreover, thoses messages cause useless trafic. Values are `true` or `false`. True make `DONE` or `ERROR` messages not sent.		No
org.objectweb.petals.transporter.qos	This property set up the policy of the Quality of Service supported by Petals Transporter. Possible values are : reliable, fast. If not specified, the reliable policy is selected by default.		No

Table 2.6. Interceptors configuration in the Service Unit

Parameter	Description	Default	Required
name	Name of the interceptor to use. That's the name defined in the component.		Yes

2.2.2. Service Unit content

The Service Unit has to contain the following elements, packaged in an archive:

- The META-INF/jbi.xml descriptor file, has described above

```
service-unit.zip
+ META-INF
    - jbi.xml (as defined above)
```

Chapter 3. Samples

Two usecases are defined in this section:

- The send of JMS Messages to the JBI Helloworld Service Engine.
- The send of JBI Messages to the JMS External provider

This section presents how install the different components and service assemblies to realize these use cases.

In each case, the external JMS Server where the client and provider queue are defined must be started in first.

3.1. Install the External JMS Server

To install the external JMS server, you must download and extracts the zip archive at this url:

Specify joram client link here

This package contains also the external jms Client and Provider used in the next sections

Inserts all jars in your classpath and start the JMS server with the command below:

```
java org.objectweb.petals.usecase.jms.common.JMSServer
```

If it is OK, you must see the lines below:

```
Start JORAM server S0...
...JORAM server S0 started
   (connected)
Initialize JMS service...
   (connect to JORAM server)
   (Creation of the queue: queueConsumer)
   (Queue created)
   (Queue bound)
   (Creation of the queue: queueProvider)
   (Queue created)
   (Queue bound)
   (disconnect from JORAM server)
JMS destinations initialized...
```

Now, the external JMS Server is ready and the client and provider queues (respectively, queueConsumer and queueProvider) have been created. You can execute the usecases below.

3.2. Send a JMS Message to the JBI Helloworld Service Engine

To send a JMS Message to the JBi HelloWorld Service Engine, you must install several components in the order listed below:

- The HelloWorld Service Engine component (Download here).
- The JMS binding component (Download here).
- The sa-jms-consume service assembly (Download here). This service assembly contains one service units:
  1. ths su-jms-consume service unit. This service unit consumes the endpoint defined by the next service unit.

Once these components are installed, you must install the JMS client.

To install the JMS client, you must download and extracts the zip archive at this url:

Specify joram client link here

By default, the JMS client is started in console mode.

Inserts all jars in your classpath and start the JMS client with the command below:

```
java org.objectweb.petals.usecase.jms.client.Client
```

If it is OK, you must see the lines below:

```
Start the Echo JMS client...
------------------------------

The parameters used for this client are listed below:
        -user=anonymous
        -password=anonymous
        -connection-factory=qcf
        -queue-name=queueConsumer
        -file=null
No file has been given in input:
You can write the xml message to send in the console or write 'quit' to exit ...(for instance:
 <text>hello world</text>)
=>
```

You can write an xml message and verify that it has been received by the helloworld component.

If you want that the JMS client send a xml file to the helloworld, you can configure it with the 'file' option as seen below:

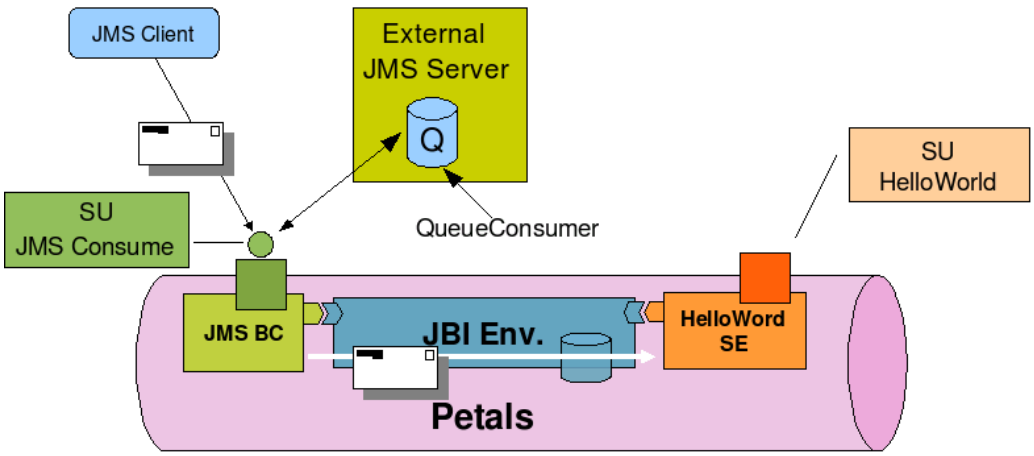
```
java org.objectweb.petals.usecase.jms.common.Client -file=test.xml
```

the test.xml file is shown below:

```
<jms>
   <text>Hello World</text>
</jms>
```

All these components can be seen on Figure 3.1, “The sa-jms-consume use case”

Figure 3.1. The sa-jms-consume use case



3.3. Send a JBI Message to the external JMS provider

To send a JBI Message to the external JMS provider, you must install several components in the order listed below:

- The Sample Client Service Engine component (Download here).

- The JMS binding component (Download here).
- The sa-jms-provide service assembly (Download here). This service assembly contains one service unit:
  1. the su-jms-provide service unit. This service unit provides an endpoint to contact the external JMS provider.

Once these components are installed, you must install the JMS provider.

To install the JMS provider, you must download and extracts the zip archive at this url:

Specify joram client link here

Inserts all jars in your classpath and start the JMS provider with the command below:

```
java org.objectweb.petals.usecase.jms.provider.Provider
```

If it is OK, you must see the lines below:

```
Start the JMS Echo services...
------------------------------

The parameters used for this server are listed below:
        -user=anonymous
        -password=anonymous
        -connection-factory=qcf
        -queue-name=queueProvider
Start the message receiver...
Message receiver started...

... JMS Echo services started...
Creating message receiver...
MsgListener ready to listen for Receiver
```

Now, the external JMS provider is ready to receive messages. You can use the sample client to send xml message to the endpoint defined by the su-jms-provde service unit.

All these components can be seen on Figure 3.2, “The sa-jms-provide usecase”

Figure 3.2. The sa-jms-provide usecase