OpenESB Enterprise Edition
JMS Connection

Document identifier:

Pymma document: 770-012

Location:

www.pymma.com

Editor:

Pymma Services: contact@pymma.com

Abstract:

This document explains how to use OpenESB Enterprise Edition to create and configure connection with the JMS broker Open MQ in the stand alone and cluster mode.

Status:

This Document is in a beta state

ABOUT PYMMA CONSULTING

Pymma Services is a technical architect bureau founded in 1999 and headquartered in London (UK). It provides expertise in service oriented integration systems design and implementation. Leader of OpenESB project, Pymma is recognised as one of the main actors in the integration landscape. It deeply invests in open source projects such as Gravitee.io. Pymma a European company based in London with regional offices in France, Belgium, Canada and Israel. (contact@pymma.com or www.pymma.com)

ABOUT OpenESB Enterprise Edition

OpenESB is today a strategic tool for IT teams who develop with success service oriented integration projects. More and more companies use OpenESB to design and develop critical projects with an enterprise scope. With OpenESB, they implemented reliable and scalable applications and process billions of messages for a high ROI. To move from departmental to Enterprise applications, OpenESB required new improvements to meet user's demands. Companies expressed strong needs for enterprise features such as security, monitoring, management with **a strong and efficient technical support.** To consider this evolution, Pymma issued a new advanced edition for OpenESB that meets these new requirements. We named it "***OpenESB Enterprise Edition (OE EE)"***. Fully compatible with the community edition, OpenESB EE embedded enterprise features demanded by OpenESB users. To take advantage of the OpenESB Enterprise Edition, please visit our web site www.pymma.com or contact us contact@pymma.com.

Copyright

Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Pymma Services LTD. This manual is provided “as is and Pymma is not responsible for and expressly disclaims all warranties of any kind with respect to third-party content, products, and services. Pymma will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services

Trademark Notice

Pymma is a registered of Pymma Engineering LTD. Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Table of Contents

1 Introduction 7
2 OpenESB and JMS Server architecture 8
3 Open MQ Installation 10
3.1 Open MQ installation Step by Step 10
3.1.1 Install JMS Connection library 11
4 Using JMS-JCA. 12
4.1 Simple tutorial 12
4.1.1 JMS Queue browser 12
4.1.2 Start OpenESB and OpenESB Studio 13
4.1.3 Install JMS Binding Component on your instance 14
4.1.4 BPEL project 19
4.1.5 Create a composite application 25
4.1.6 Create a test 28
4.1.7 Check messages in the queue 28
5 Define a connection with JNDI 30
5.1 Introduction 30
5.2 Introduction on MQ Cluster 30
5.3 Install Java DB 32
5.3.1 Start Java DB 32
5.4 Install Open MQ 33
5.4.1 Setup Open MQ Configuration files 33
5.5 Create a File Objects Store 36
5.6 Create a connection factory 37
5.7 Create a destination 39
5.8 Connect the console to the brokers (Optional) 39
5.9 OpenESB JMS BC configuration with JNDI 40
5.10 Test the high availability of the cluster 41
6 OpenESB connection to Apache ActiveMQ 43
6.1 Introduction 43
6.2 OpenESB Configuration 43
6.3 JNDI Configuration 43
7 Hornetq 44
7.1 Introduction 44
7.2 OpenESB Configuration 44
7.3 JNDI Configuration 44
8 IBM Websphere MQ 45
8.1 OpenESB Configuration 45
8.2 JNDI Configuration 45
9 Test your parameters 47
10 Conclusion 48
11 What's next 49
11.1 Reading 49

11.2 OpenESB Enterprise Edition 49
12 Help and support 50
12.1 From the community 50
12.2 From Pymma 50

1 Introduction

OpenESB Enterprise Edition (EE) is the latest OpenESB Edition. Its architecture is smart and light. Very fast and scalable, it runs without any container such as a JEE Application Server or a OSGI container but just with a simple JVM. With a very small memory footprint (from 150 MB) OpenESB fits perfectly the new architecture (virtualization, cloud) deployed in production.

OpenESB EE is independent from the application set, previously embedded with Glassfish application server such as Open MQ or Derby DB. Consequently, OpenESB users cannot rely on the container to create Database and JMS connections anymore.

The connection between JMS Messaging System and OpenESB is set up through a Binding component named JMS BC. This document provides advanced explanations on how to setup the JMS BC and provide some examples. Nevertheless, we don't discuss here about JMS use with OpenESB at the architecture level.

You will find further information in the document "JMS BC user guide" in the OpenESB web site (www.open-esb.net).

Prerequisite: you have a good knowledge on OpenESB, JMS technology and terminology.

Some reference on JMS:

http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html

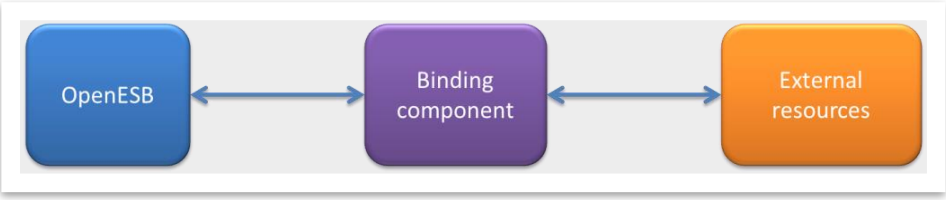
http://www.coderpanda.com/jms-tutorial/

http://download.oracle.com/otndocs/jcp/7195-jms-1.1-fr-spec-oth-JSpec/

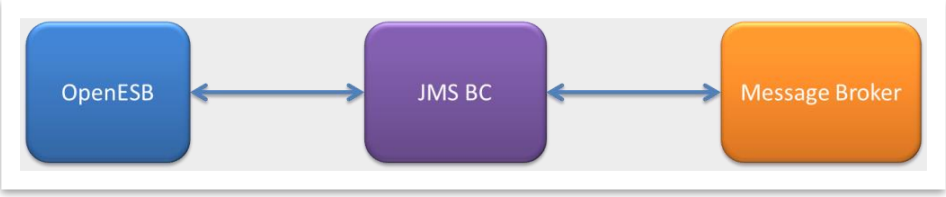
http://docs.oracle.com/cd/E19957-01/816-5904-10/816-5904-10.pdf

2 OpenESB and JMS Server architecture

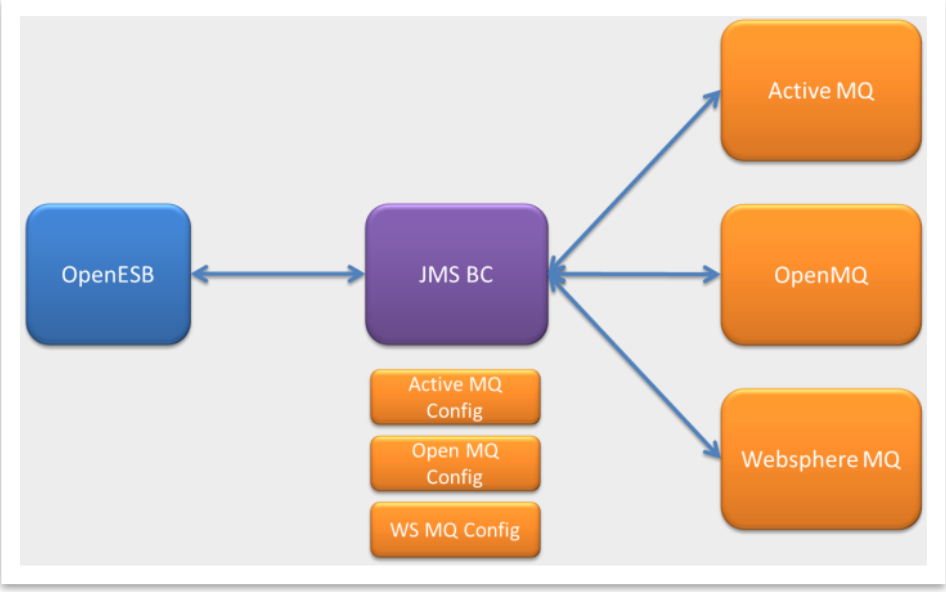
OpenESB core is not able to access by itself to any external resources such as a JMS Queue, file or FTP Server. This task is dedicated to the binding components (BC).



When a OpenESB user wants to involve an external resource in a business process, he must set up a Binding Component to create a connection to this resource. In the JMS use case, the JMS BC role is to create and manage a bidirectional connection between OpenESB and a message broker.



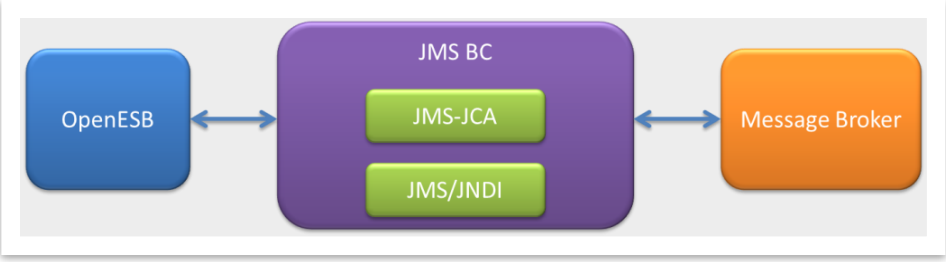
JMS BC versatility allows OpenESB to be connected to all JMS products on the market such as Open MQ, Active MQ or WebSphere MQ.



OpenESB offers two configurations to setup a connection with a message broker.

The first configuration relies on a JMS-JCA implementation embedded in the JMS Binding Component. JMS-JCA easily setups connections for development and tutorial purposes. Nevertheless, using JMS-JCA links the development with the JMS Client configuration, and it is up to the developer to setup a correct configuration for a JMS customer in place of a JMS expert.

The second way to setup a configuration is to use JNDI features to create JMS connection. A bit more complex to understand, JMS/JNDI is more versatile and efficient for connecting redundant and highly available JMS architectures. Moreover, it is compatible with all JMS products.



We highly recommend you JMS/JNDI configuration for your future deployment in production.

3 Open MQ Installation

OpenESB Enterprise Edition (OE EE) does not contain any additional software such as a JMS broker or a database server to keep its installation light and simple. You can install any tiers JMS product to work with OpenESB SE. Nevertheless, since OpenESB legacy users were used with Open MQ (installed by default with the Glassfish Applications Server), we provide a short guide on Open MQ installation. In OpenESB legacy edition, Open MQ was also often named IMQ.

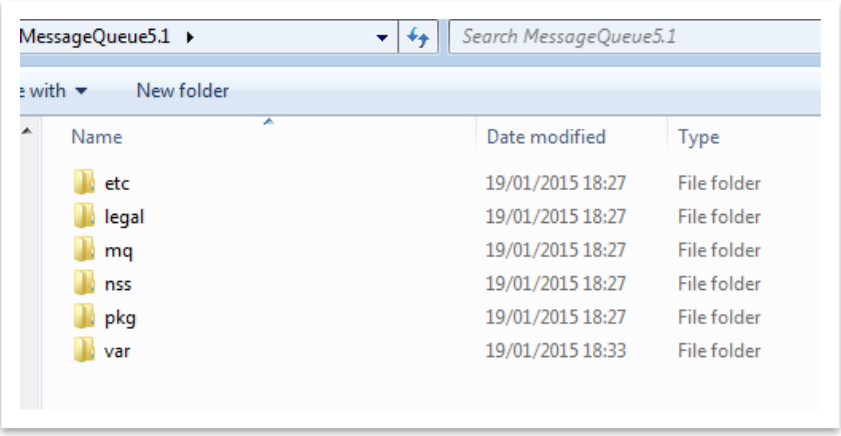
Open MQ can be downloaded from the website https://mq.java.net/ which is the official web site for Open MQ. Let's say a few words on Open MQ:

Open MQ is the reference implementation of JMS specifications. So, if you use it for your development, we are sure to be compliant with the latest JMS specifications and you can take advantage of the latest JMS upgrades. For example, today (2016 Q1) Open MQ 5.x is compliant with JMS 2.0.

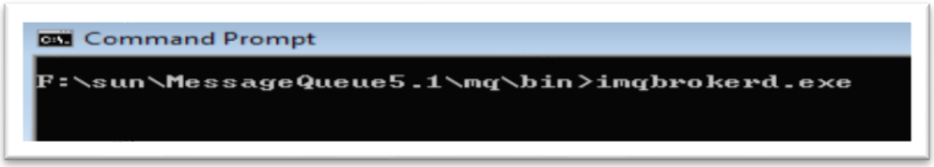
Easy to install and to use, Open MQ is reliable, powerful, scalable and well documented. It can be configured in a cluster mode for high availability purposes. In that case, a database is required to support the best high availability configuration. Many companies install Open MQ on production and process billions of messages every month. If Open MQ is not one of the fastest or the smartest JMS tools on the market it is certainly one of the most reliable such a Land Rover in the savannah.

3.1 Open MQ installation Step by Step

Download Open MQ from the https://mq.java.net/. Get the zip file and unzip it in your disk.



Open a console and go to the directory "mq_directory"/mq/bin and start the program **imqbrokerd**.



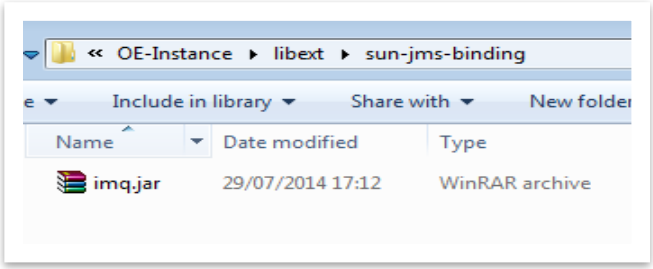
After a few seconds, your message broker is ready.

```
[#|2017-04-23T20:59:33.241+0100|FORCE|5.1|imq.log.Logger|_ThreadID=1;_ThreadName=main;|[B1039]: Broker "imqbroker@PymmaTH510:7676" ready.
|#]
```

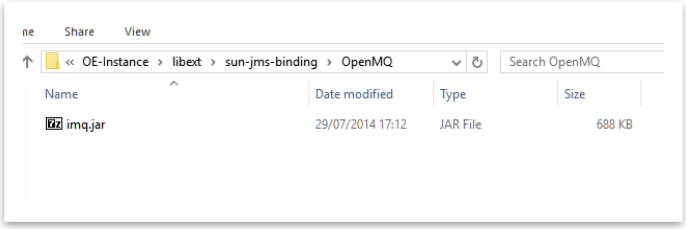
If the broker does not start as expected, please have a look at the documentation and especially the Message Queue Technical Overview and the Message Queue Administration Guide. (The documents can be found there: https://mq.java.net/downloads/index.html)

3.1.1 Install JMS Connection library

Before starting JMS Binding component, you must add the JMS Client libraries in the component class path. This Jar file contains the Java classes used by the JMS BC to access to the message broker. In Open MQ, the client jar file is named imq.jar. Copy the file imq.jar. Go to the directory ${OpenESB-HOME)/OE-Instance/libext. Create a sub directory jms-sun-binding. Then past imq.jar to jms-sun-binding directory. To consider the new class path, the Binding Component must be shut down and restarted



The OpenESB version 3.1.0 and upper, allows a selective organisation of the jar files. You can create subdirectories to organise the external libraries. Go to the directory ${OpenESB-HOME)/OE-Instance/libext/jms-sun-binding and create a sub directory openMQ.



4 Using JMS-JCA.

JMS-JCA is a library abstracting the differences between JMS servers and is used in many application servers such as Glassfish, JBoss and Weblogic. Used by JEE application servers, JMS-JCA was a very useful tool to connect Java POJO or EJB to a message broker. JMS-JCA was initially a part of the OpenESB project and implemented in the component JMS Binding Component when OpenESB was a part Sun Microsystems' Java EE stack. JMS-JCA is easy to use, especially for development and testing purposes.

However, in a production environment, we highly advise to use an external JNDI context to create and manage JMS connections.

Using JMS-JCA to create connection is easy; the developer has to provide an URL with the broker protocol, the address and the port of the broker. Example: By default, the Open MQ URL is: mq://localhost:7676.

JMS-JCA supports other protocols such as t3 for Weblogic, wmq for WebSphere MQ.

The table below lists the protocols supported by OpenESB implementation

JMS Provider	ConnectionURL
STCMS	stcms://, stcmss://
OpenMQ / JMQ / Sun Java System Message Queue	mq://, mqtcp://, mqssl://, httpjms://, httpsjms://
JMS Grid	stream://, tcp://, ssl://, http://
WebSphere MQ/MQ Series	wmq://, wmq5://
WebLogic JMS	t3://
JBoss JMS	jboss://
STCMS 4.5.3 / SRE	stcms453://
Generic JNDI	jndi://

Please, note that JMS-JCA does not support some recent JMS Brokers such as Active MQ. In place, for these brokers, the OpenESB developers use JNDI to setup for JMS Connections. So, they will take advantage of the last features available from their JMS Brokers.

In this document, we develop a simple example to explain how to use JMS-JCA.

4.1 Simple tutorial

4.1.1 JMS Queue browser

Before starting, we recommend you download the utility QBrowser. This application is used to read, create, delete messages in JMS Queues.

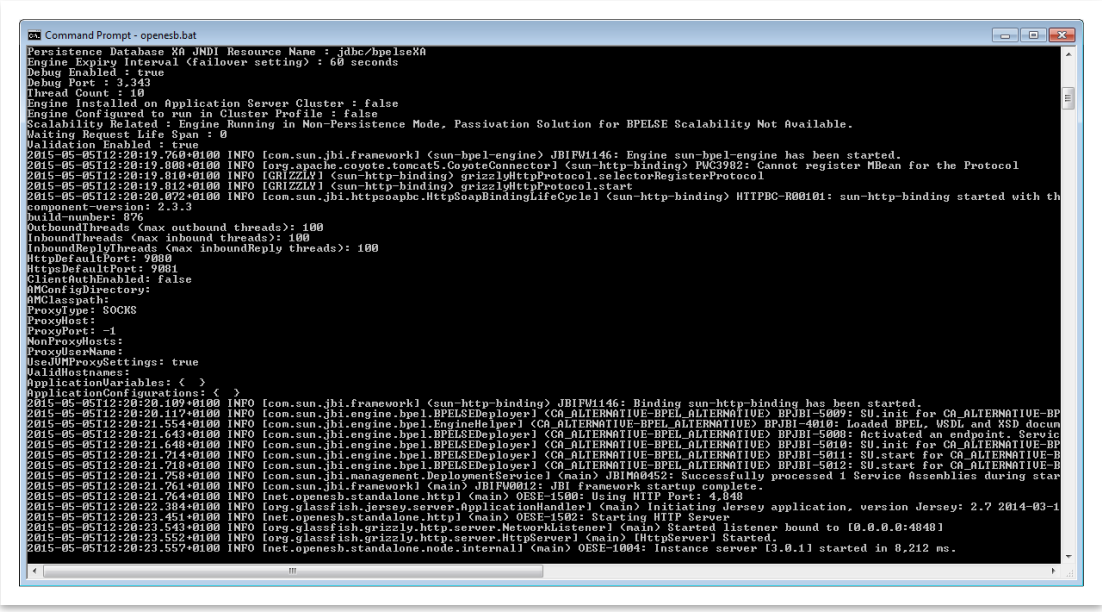
QBrowser can be downloaded here http://sourceforge.net/projects/qbrowserv2/ .

A similar tool named JMS Browser can be used as well: http://jmsbrowser.com/

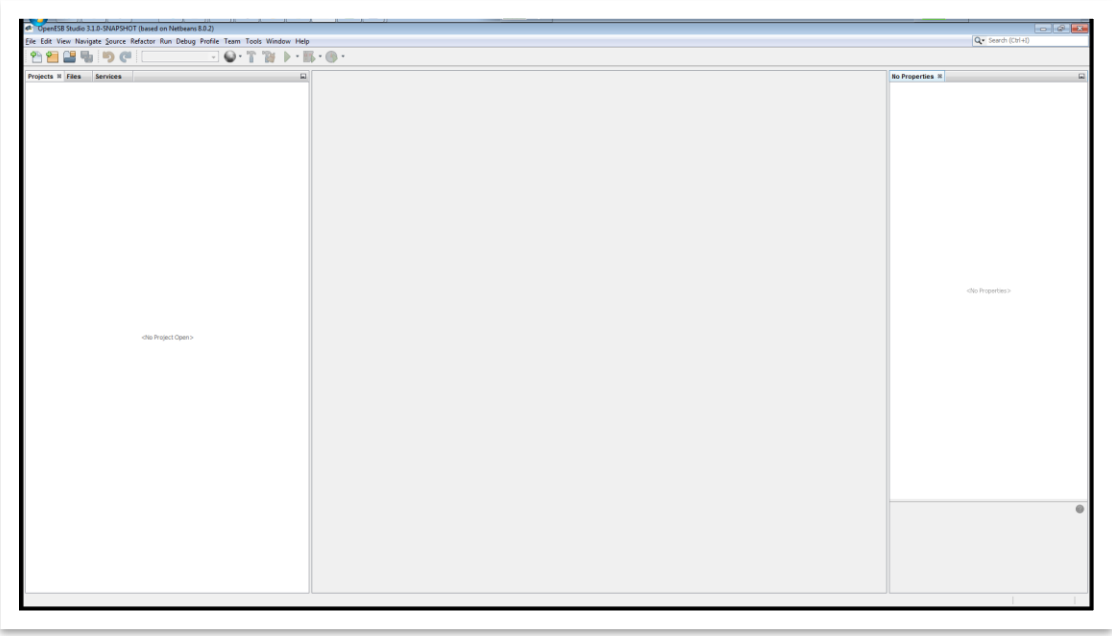
4.1.2 Start OpenESB and OpenESB Studio

The first step in this tutorial is to start an OpenESB Enterprise Edition instance and then OpenESB studio to develop and deploy this tutorial. If you are an OpenESB beginner, please, have a look on the documentations 770-001 “Start with OpenESB Enterprise Edition” and 770-002 “OpenESB Enterprise Edition: Hello World”.

For now, we suppose your OpenESB instance is started and OpenESB studio opened.



OpenESB instance has been started

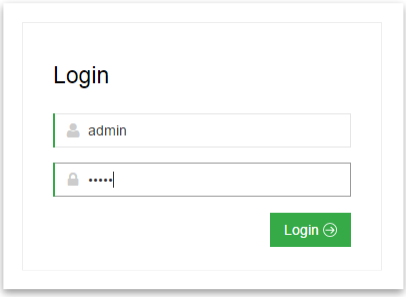


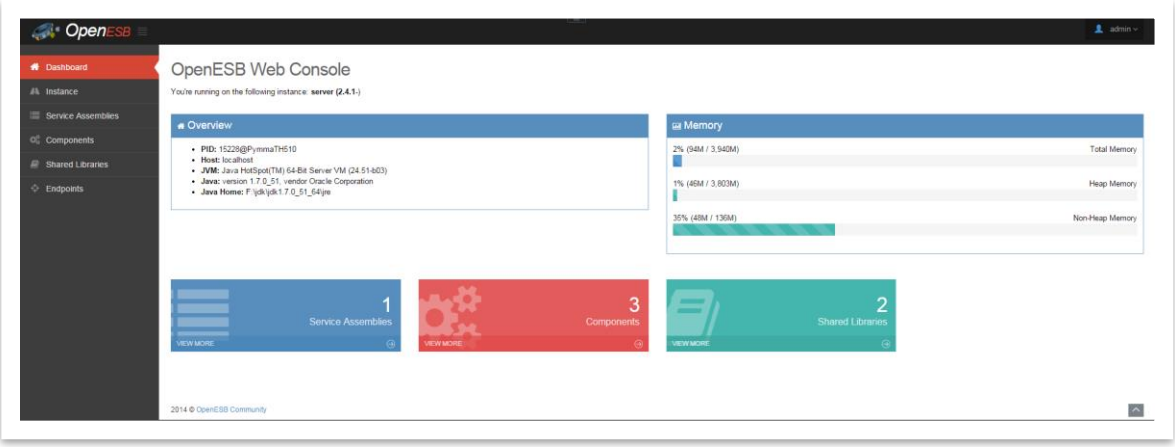
OpenESB Studio is open

4.1.3 Install JMS Binding Component on your instance

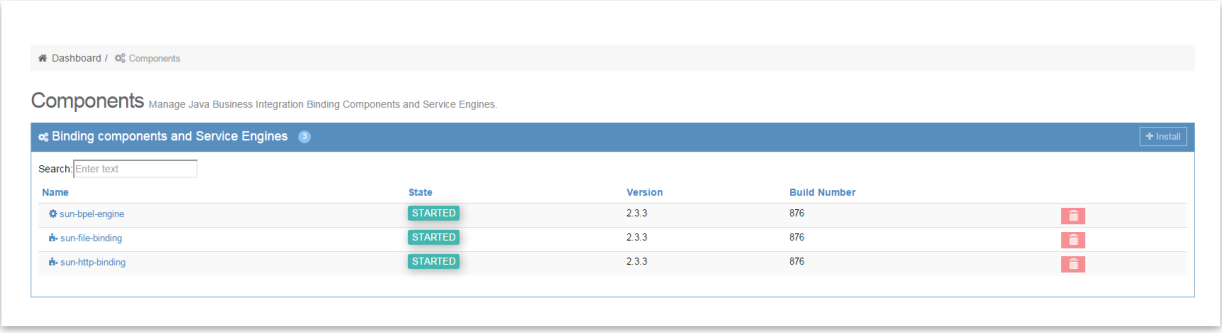
To use JMS resources with your integration project, the JMS BC must be installed on your OpenESB instance. To do it, we use the OpenESB administrative web console (For more information on the web console see the document 770-003 “Administrative web console”)

Open a browser and type the address of your instance web console (By default: the web console address is “http://localhost:4848/webui” and the login and password are “admin” and “admin”)



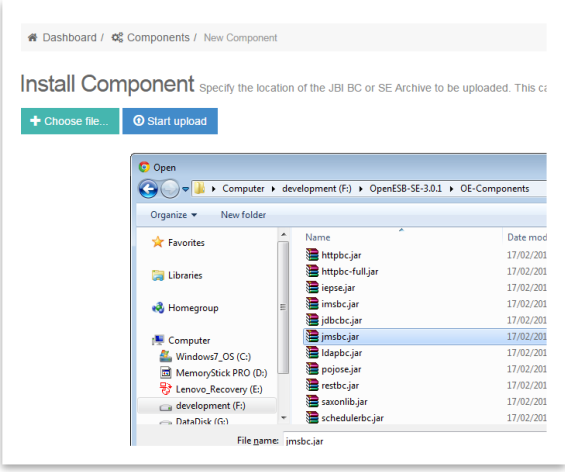


On the left menu, select components

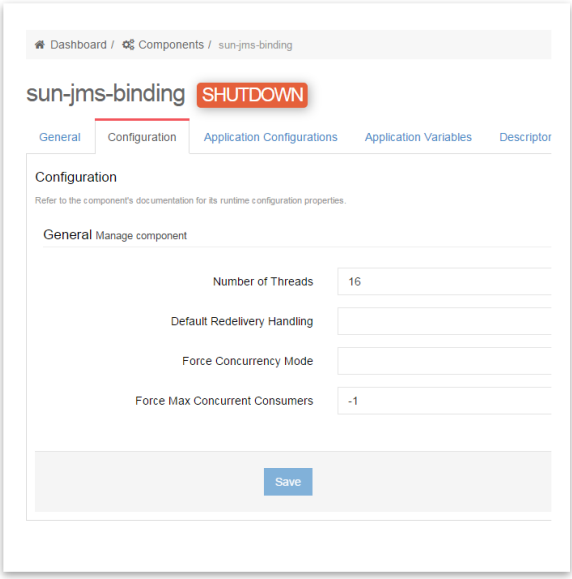


To deploy the tutorial, BPEL and Http Binding Components must be installed and started. If not, please, install them before going forward. To install the JMS BC component, click on the button install on the right side of the screen.

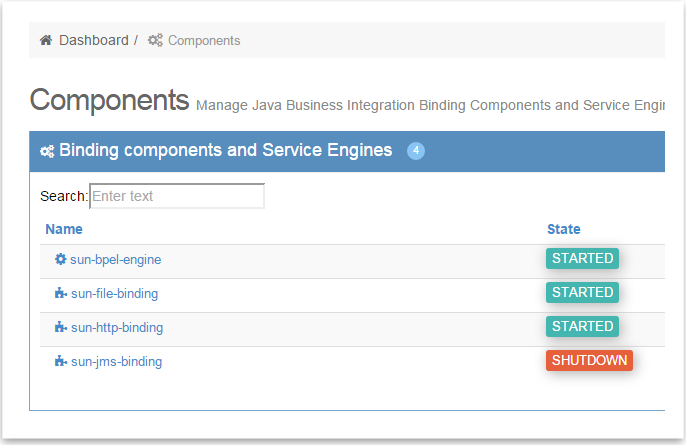
Then click on “Choose file”. Select the file JMSBC.jar in the directory ${OESE-HOME}\OE-Components where ${OESE-HOME} is the directory where you installed OpenESB. (See: OpenESB Enterprise installation documentation).



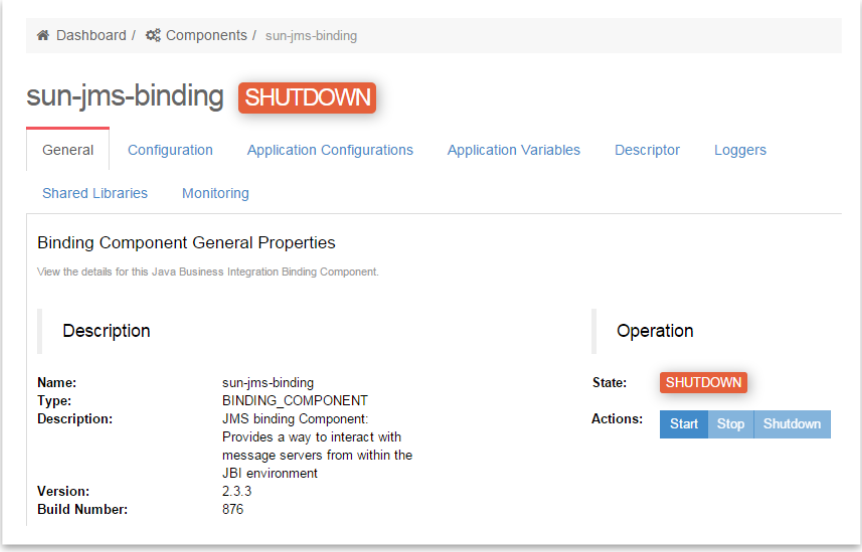
Then click on the button Start Upload. After a few seconds, the component is installed.



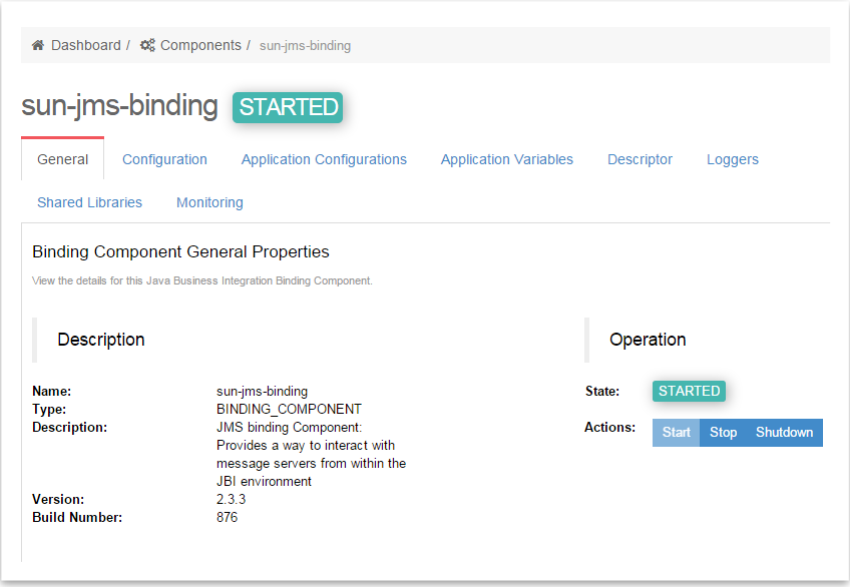
On the breadcrumb trail, click on components.



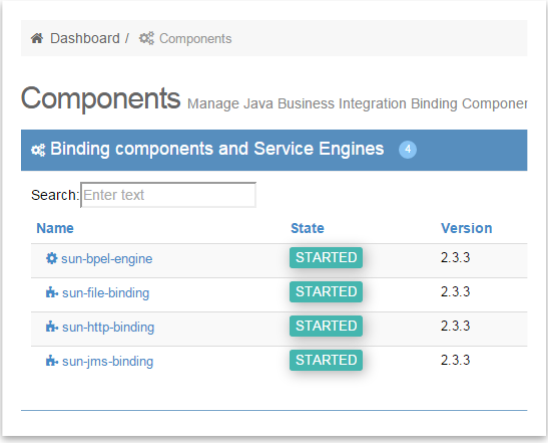
The JMS Binding component is installed but not start. Click on sun-jms-binding.



Click on the action “Start”.



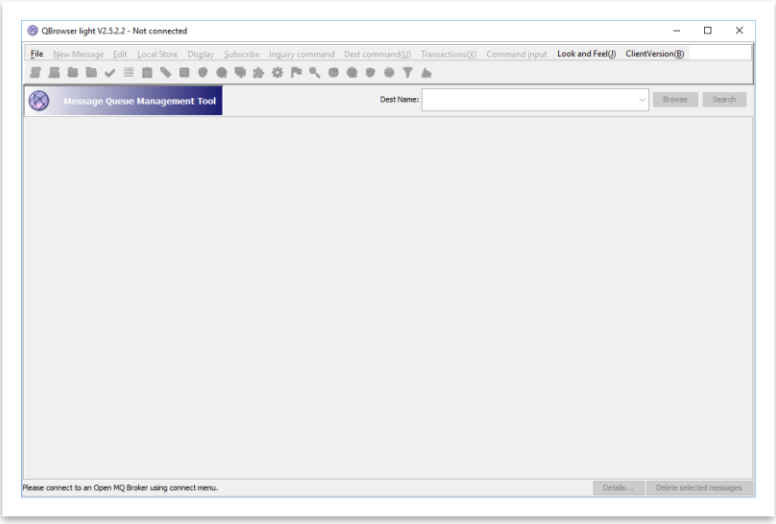
The component is started now.



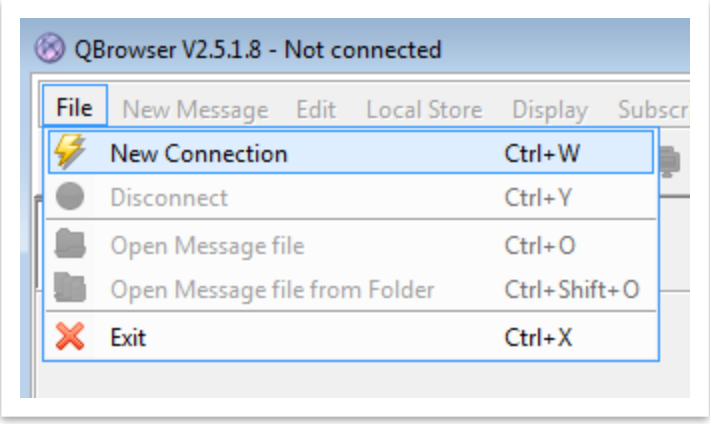
4.1.3.1 Create a queue with QBrowser

To Create a queue in the message broker, we use QBrowser.

In the QBrowser directory regarding your Operating System, run the batch file “run_open_mq.bat” or “run_open_mq.sh”.

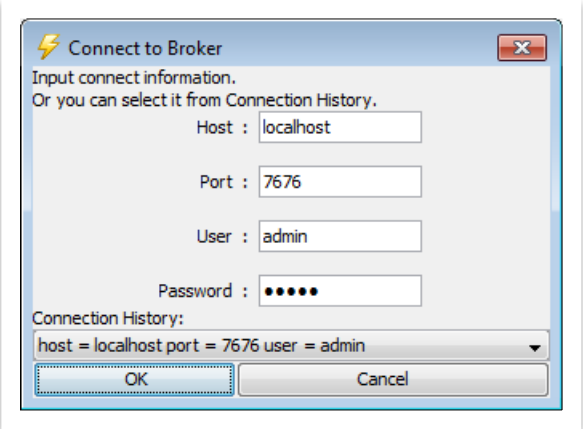


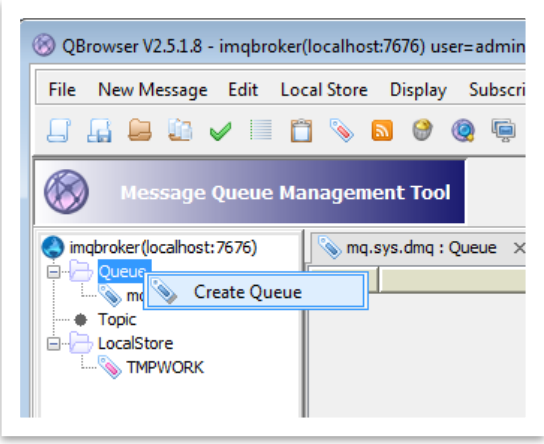
Once QBrowser Open, create a new connection.

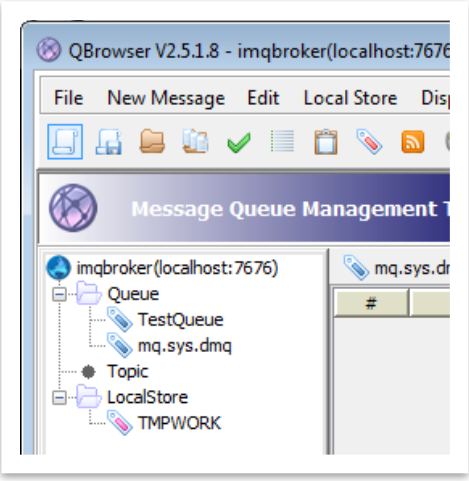


Enter the connection detail of the Open MQ broker.

Once the connection is successful, create a queue and name it “TestQueue”



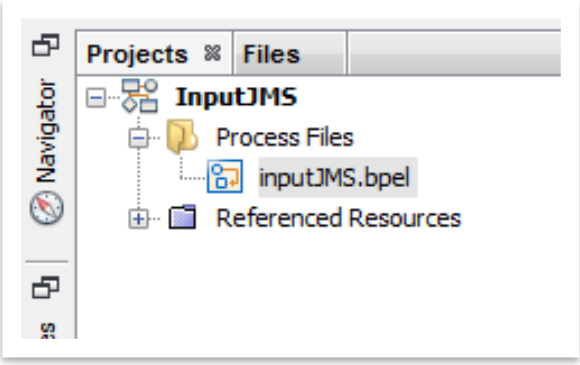




The queue is created, now, let's go back to OpenESB!

4.1.4 BPEL project

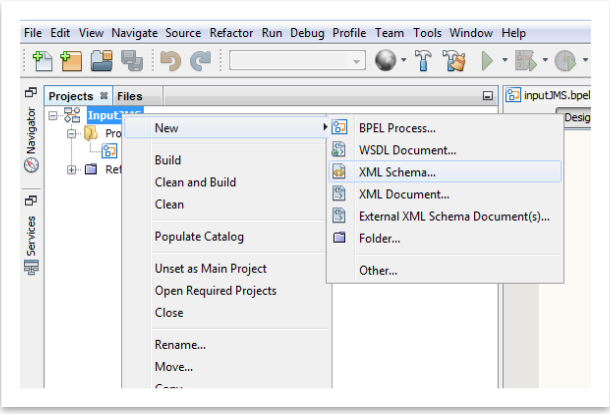
In OpenESB studio, create a new BPEL project and name it "InputJMS"

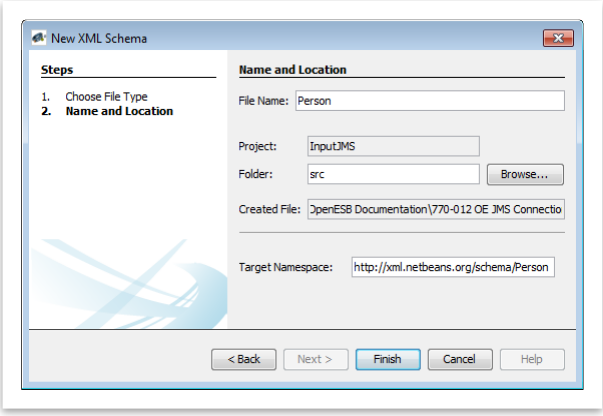


4.1.4.1 Create a schema

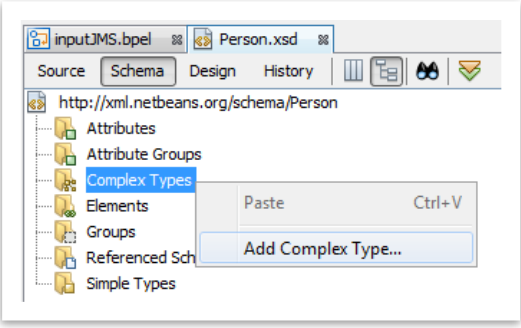
In OpenESB development process, the first stage is "create a schema". The schema is the message structures that will be exchanged between the partners. The schemas will be used in the second stage by the WSDL.

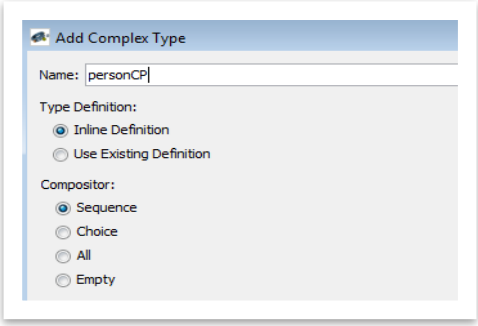
Click right on the project and select "New →XML Schema". Name the schema "PersonSchema" then click on Finish.





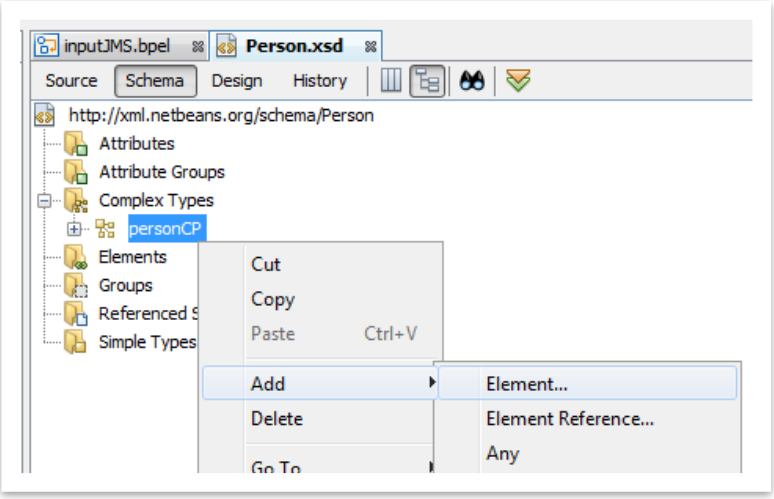
Then add a complex type and name it “personCP” then click on Finish

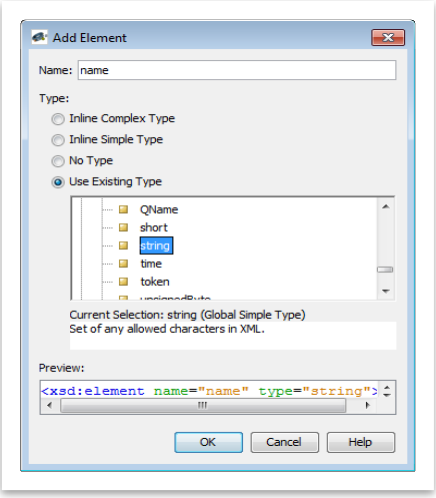




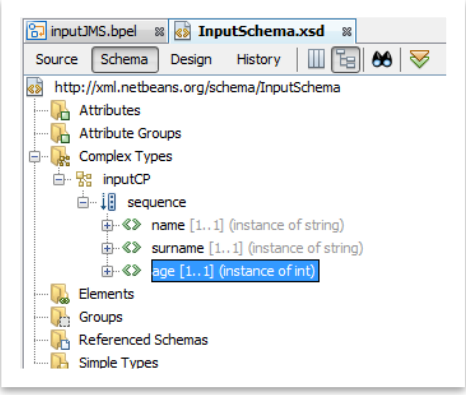
Create the element following elements

Elements	Type
name	String
surname	String
age	int

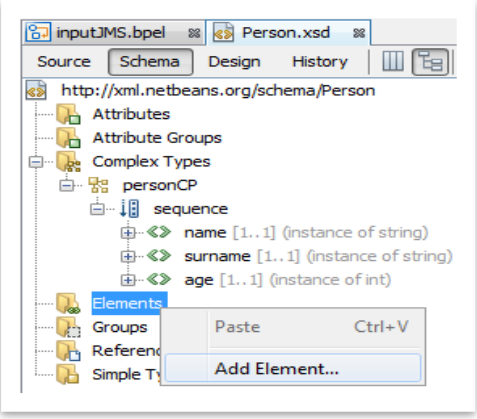


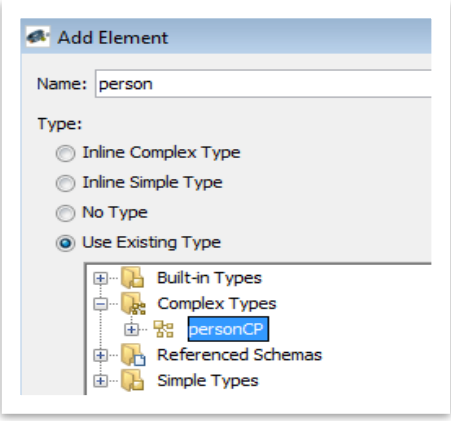


At the end, the Complex Type is as follows:

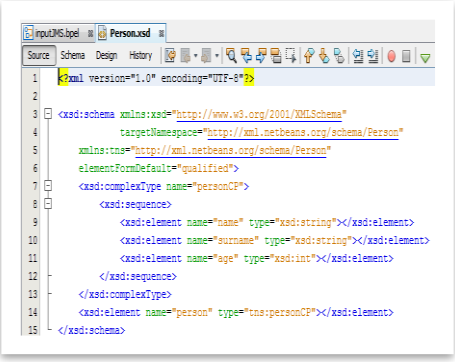


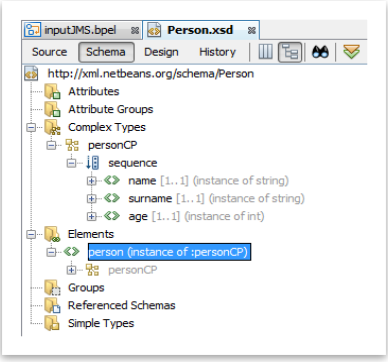
Then create the element person with the complex type personCP

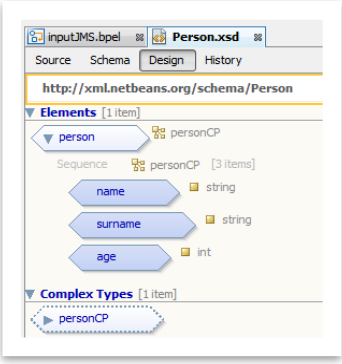




The final schema is as follows:



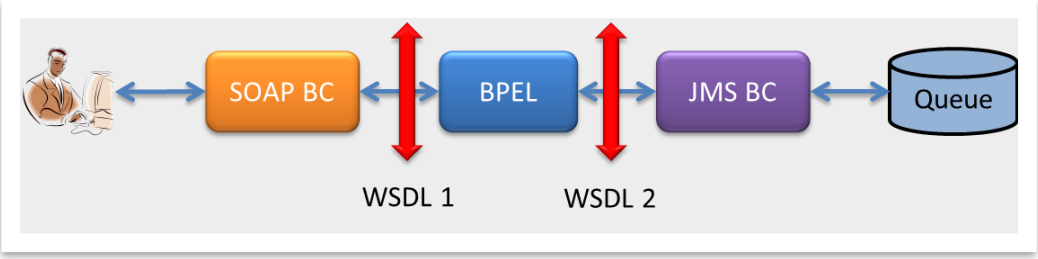




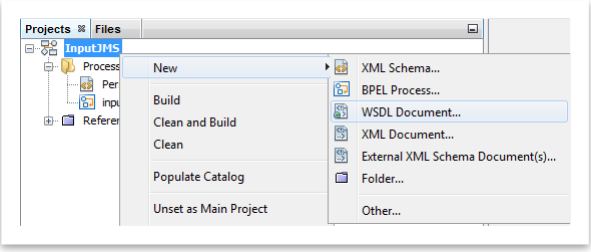
OpenESB schema editor comes with 3 views: Source, Schema and Design.

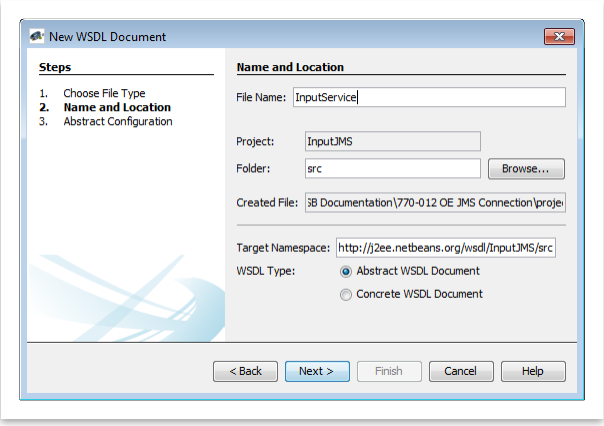
4.1.4.2 Create contract of services: WSDLs

In the second stage of this tutorial, you must create two contracts of services or WSDLs. The first defines the inbound service that will receive the input message from a SOAP call and the second, the outbound service that put the message into the queue.

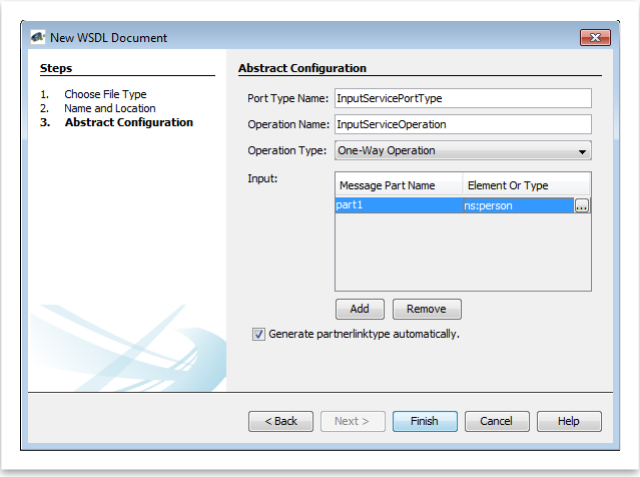


Click right on the project, select new→WSDL document and Name the WSDL"InputService"

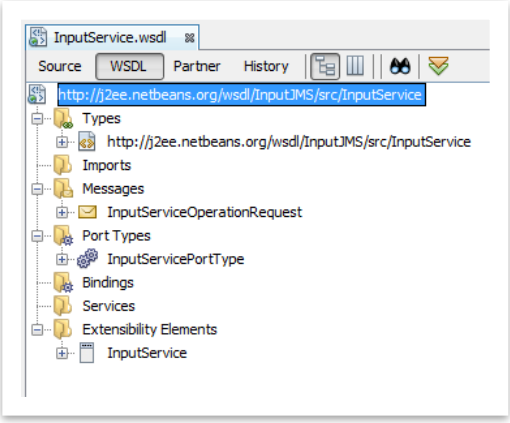




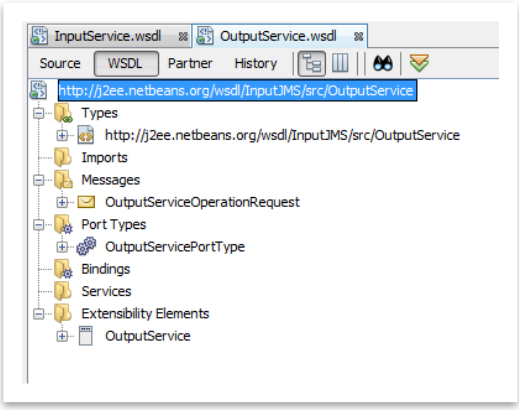
Choose the WSDL Type as Abstract and then click on Next.



Select the operation type: One-Way-Operation and person as Element for part1. Then click Finish. Your WSDL is as follows:



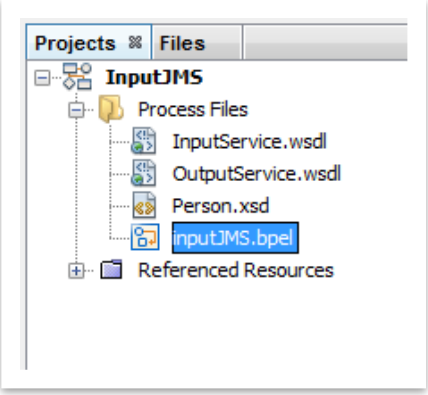
Create the second WSDL and name it “OutputService”. Follow the process you used to create InputService to define OutputService. The second WSDL must be as follows:



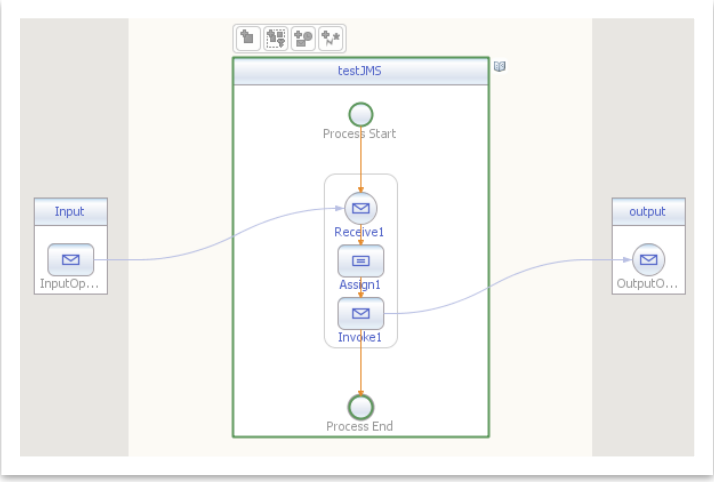
Note we did not associate any protocol to the WSDLs. It’s the reason why they have been named them “abstract WSDL”.

4.1.4.3 Create a BPEL

Double click on the node inputJMS.bpel. The BPEL editor opens then create the following process.

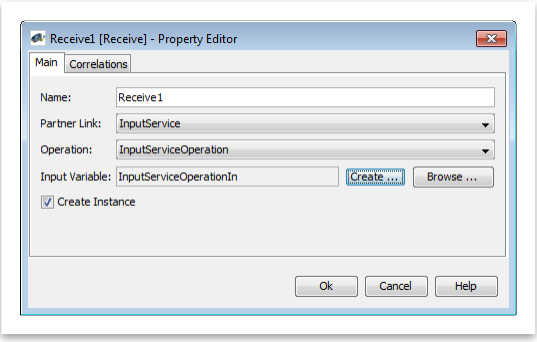


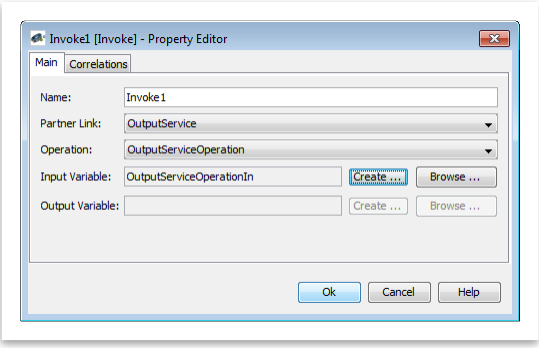
Create the following process:



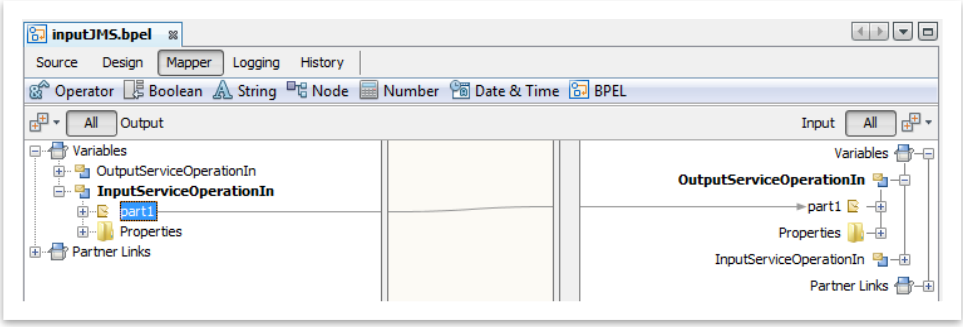
Drag the WSDL InputService on the left lane of the BPEL editor. Drag the WSDL OutputService on the right lane of the BPEL editor. Add in the BPEL process, one receive activity, one assign activity, and one invoke activity as described in the picture above.

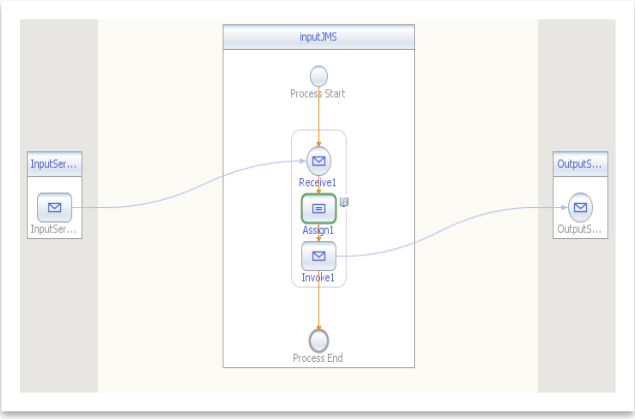
Then connect inputService with Receive1 activity and OutputService with Invoke1 activity. Then rename partnerLink1 with Input and PartnerLink2 with Output.

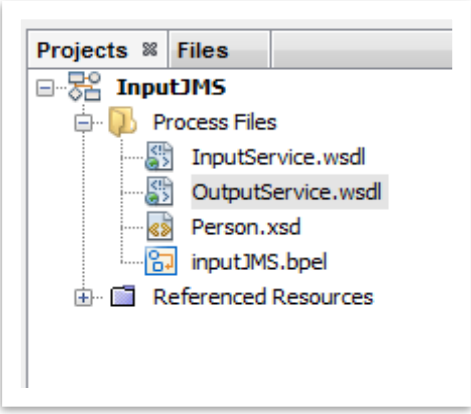




Double click the assign activity and map the variable as follows:



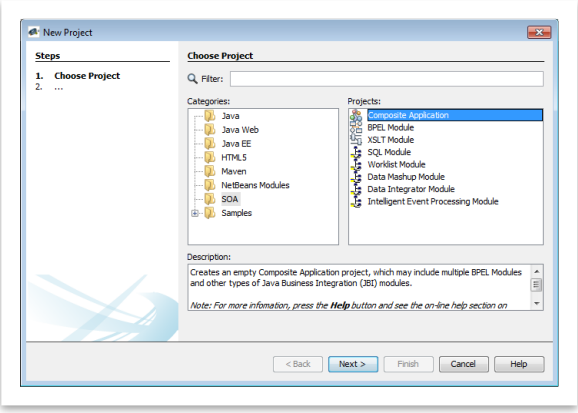


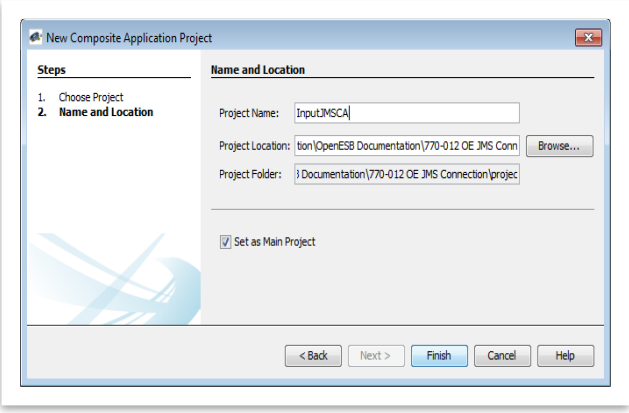


4.1.5 Create a composite application

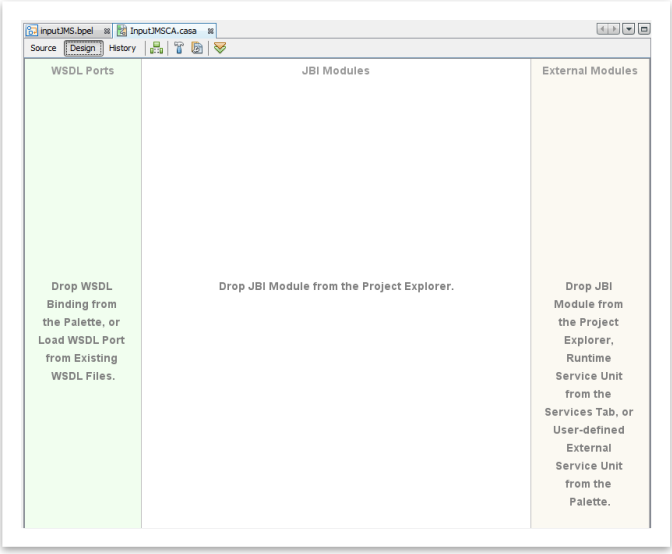
Composite application has many purposes; one of them is to assign the protocols to the endpoints of the BPEL process.

Create a Composite Application project and name it InputJMSCA.

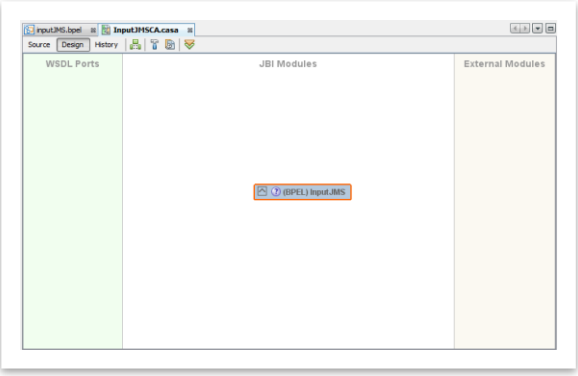


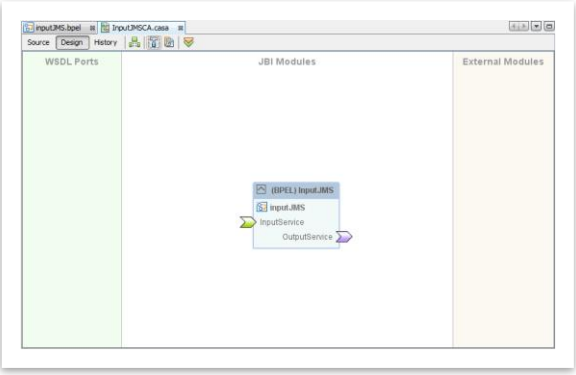


Select the design view of the casa editor.

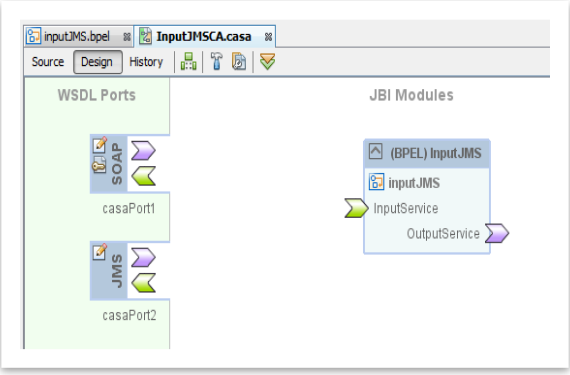


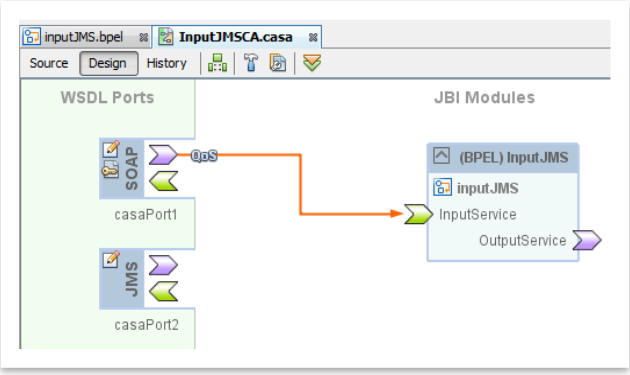
Drag you BPEL project in the white central lane of the editor. Then build the project





Drag the SOAP and JMS Binding on the green left lane of the CASA Editor.

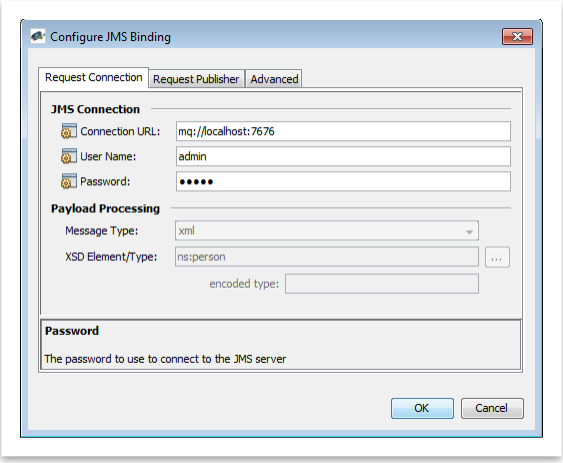


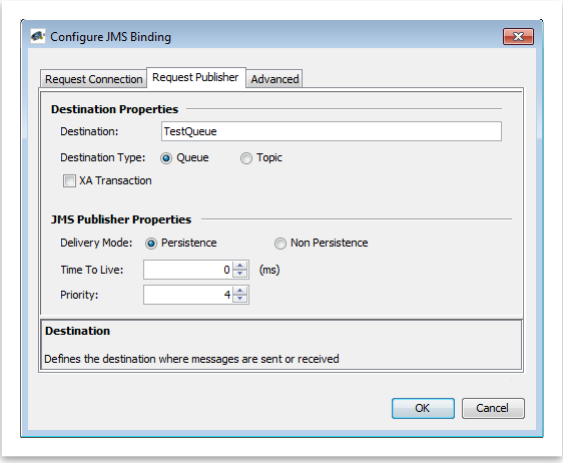


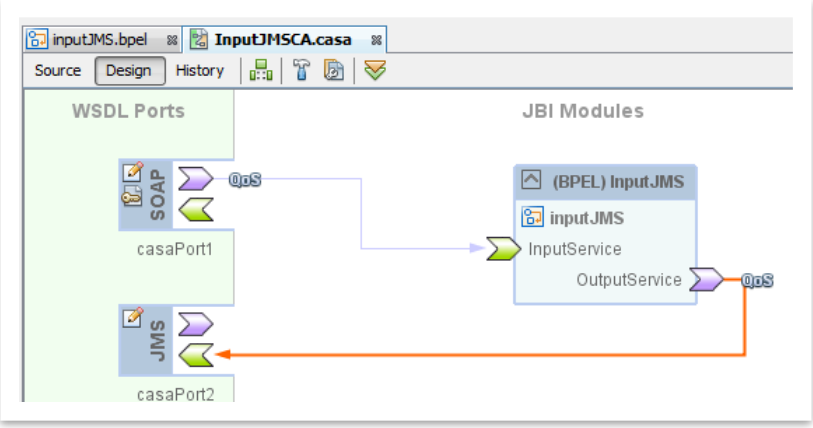
Drag the SOAP purple arrow to the BPEL green arrow. You assign a SOAP port to access to the service InputService.

Drag the BPEL purple arrow to the JMS green arrow. Then a new screen opens. On request Connection tab, left the URL unchanged, set the user name and password to “admin”.

On the “Request Publisher” tab set the destination to “TestQueue” Then click OK



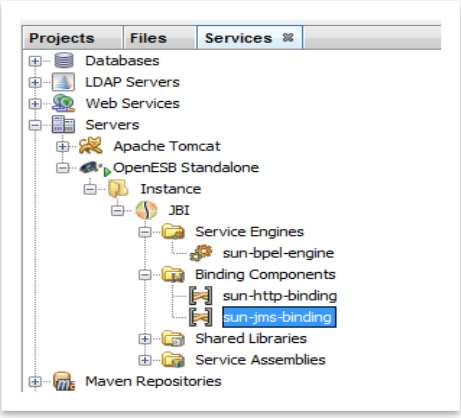


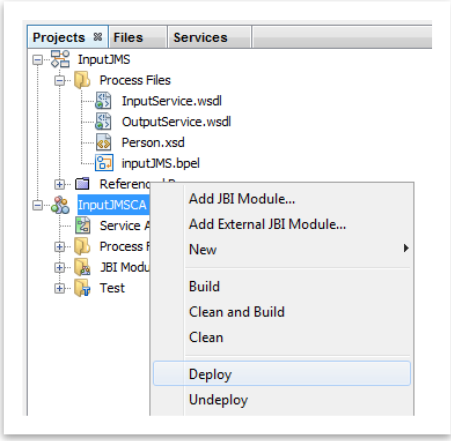


The project is ready to be deployed.

- Check if OpenESB instance is started already and HTTP BC, JMS BC and BPEL SE too.
- Check if Open MQ is started.
- Check if OpenESB studio connects to the OpenESB instance.

Click right on InputJMSCA project and deploy it

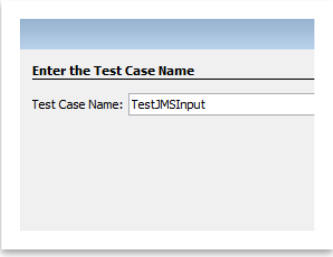


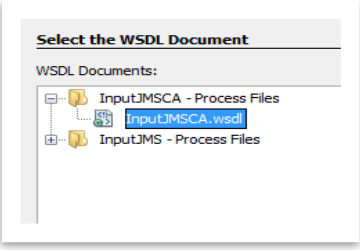


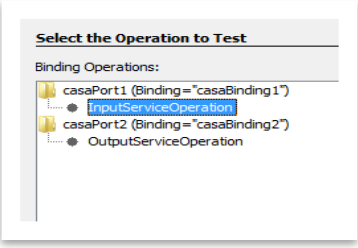
4.1.6 Create a test

In the Composite Application project, click right on the “test node” and create a new test and name it testJMSInput.

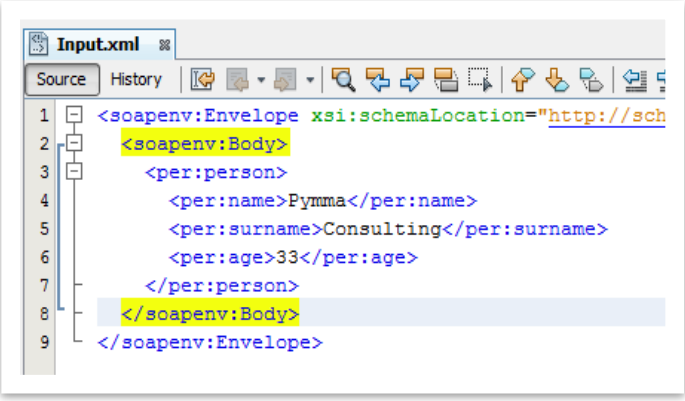
Then select InputJMSCA.wsdl. At least select InputServiceOperation and click on Finish.

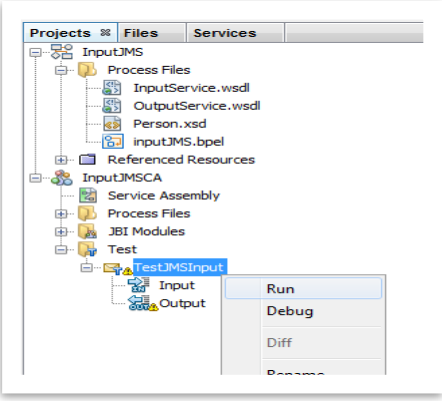






Modify Input.xml as follows, save it (Ctrl+S) and run the test.

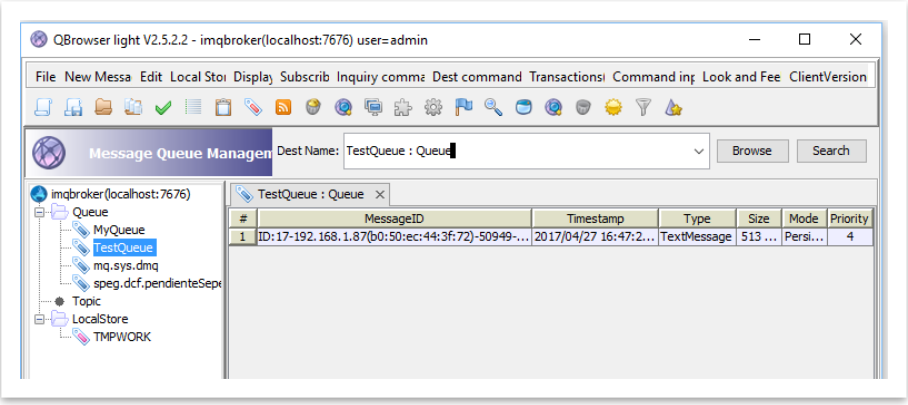




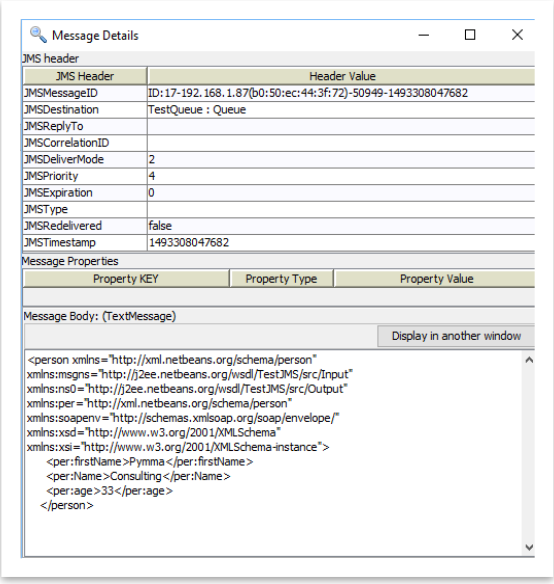
The test must not return any message.

4.1.7 Check messages in the queue

Let’s go back to QBrowser.



TestQueue has a new message. Double click on the message:



The message sent by OpenESB is now stored in the Queue

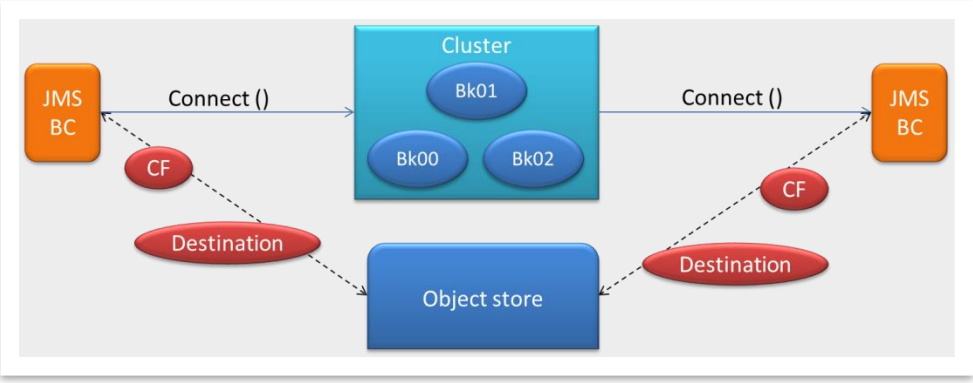
5 Define a connection with JNDI

5.1 Introduction

There is another way to set up a JMS BC for High availability outside JMS-JCA, to do it, we use the Java Naming API (JNDI). JNDI is the “standard” way to create a connection between a Java application and external Resources such as a JMS message broker. OpenESB JMS Binding component use JNDI to lookup connection factories and destinations (“Administered Object”) and then create a JMS connection. JNDI stores the connection information in an “Object Store”.

See: http://docs.oracle.com/javase/7/docs/technotes/guides/jndi/.

Open MQ supports 2 “object store” types. Connection Factories and Destinations can be stored in an LDAP server (Lightweight Directory Access Protocol directory server) or in a file (in the local file system or a network clustered file system SAN). “Administered objects are placed in a readily available object store where they can be accessed by client applications by means of the Java Naming and Directory Interface (JNDI)”.



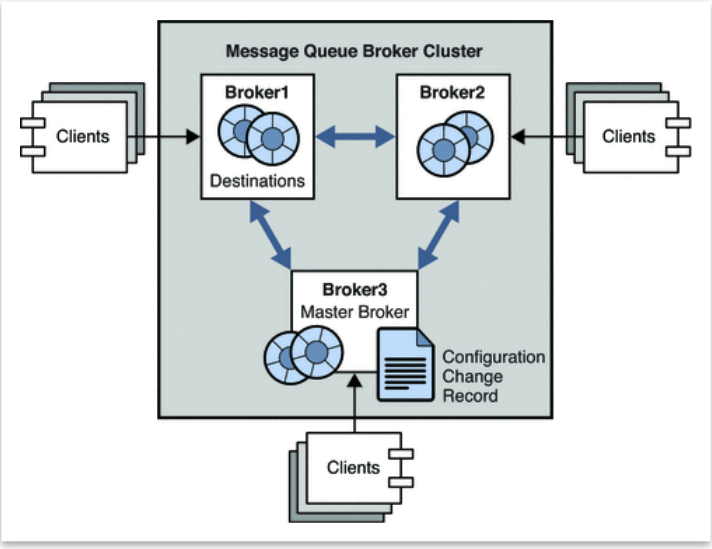
The component OpenESB JMS BC uses the object store to access to the brokers. So, we decorrelate the connection configuration from OpenESB implementation and broker expert can be involved in the connection definition and not in OpenESB development.

In that document, we detail how to create a connection with a set of brokers in the cluster. So, we must define an external “**objects store**” where factories and destinations will be defined and stored.

As explained above, there are two object store types usable by Open MQ. The first is the “file Object Store” that relies on shared files to store the objects. The second is the “LDAP objects store” that relies on LDAP to store the objects. To keep the example simple, we use the “file objects store” in a local drive.

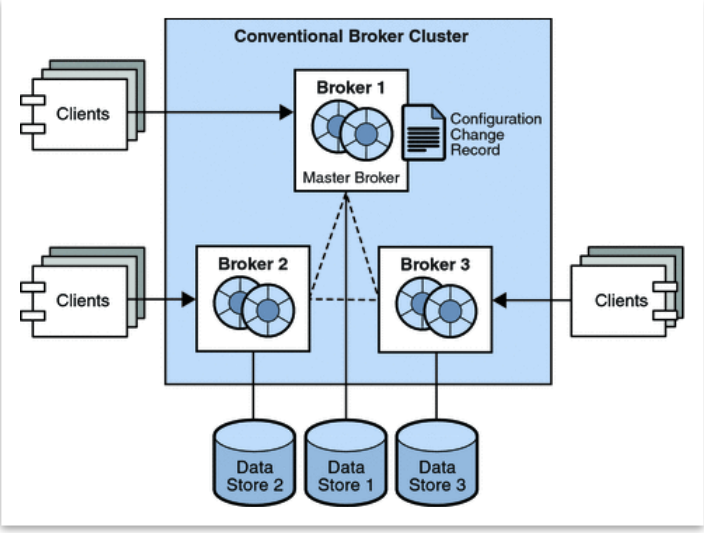
5.2 Introduction on MQ Cluster

Open MQ supports the broker clusters. It is a group of brokers working together to provide message delivery services to clients. Clusters enable an administrator to scale messaging operations with the volume of message traffic by distributing client connections among multiple brokers.

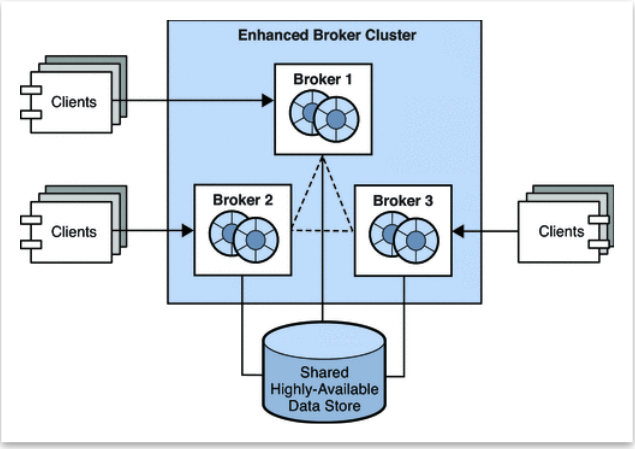


Open MQ offers two cluster architectures, depending on the degree of availability you want.

- **Conventional clusters** provide service availability, but not data availability. If one broker in a cluster fails, clients connected to that broker can reconnect to another broker in the cluster, but may be unable to access some data while they reconnect to the alternate broker.
-



- **High availability** cluster provides both service availability and data availability. If one broker in a cluster fails, a client connected to that broker is automatically reconnected another broker in the cluster. Clients continue to operate with all persistent data available.



The picture above describes a High Availability (HA) Clusters configuration with a HA Database. For more information on Open MQ Cluster, please have a look at: https://glassfish.java.net/docs/4.0/mq-admin-guide.pdf

5.3 Install Java DB

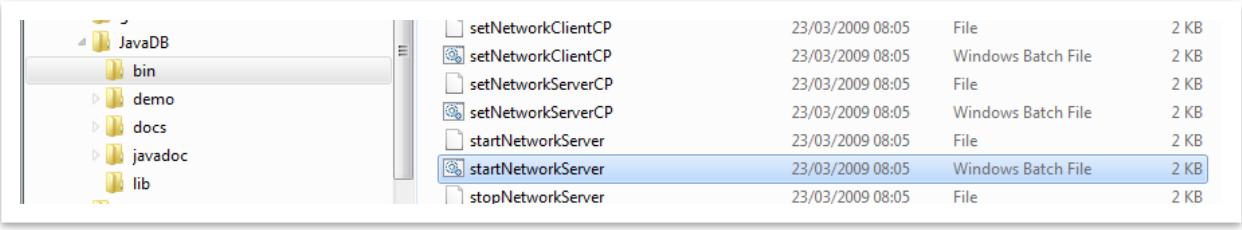
Setting up a High Availability Cluster with MQ involves a high availability database use. To reach the high availability, the database must support High Availability too. In production, Open MQ cluster in HA mode can be used with clustered databases such as Oracle RAC, DB2 Integrated Cluster, MySQL or Postgres Cluster. In this paper, we will not spend time setting a HA database but we will use a simple Java database (derby) as clustering support.

Java DB is a simple database 100% Java usually used for prototype, test and POC. Download and install Java DB.

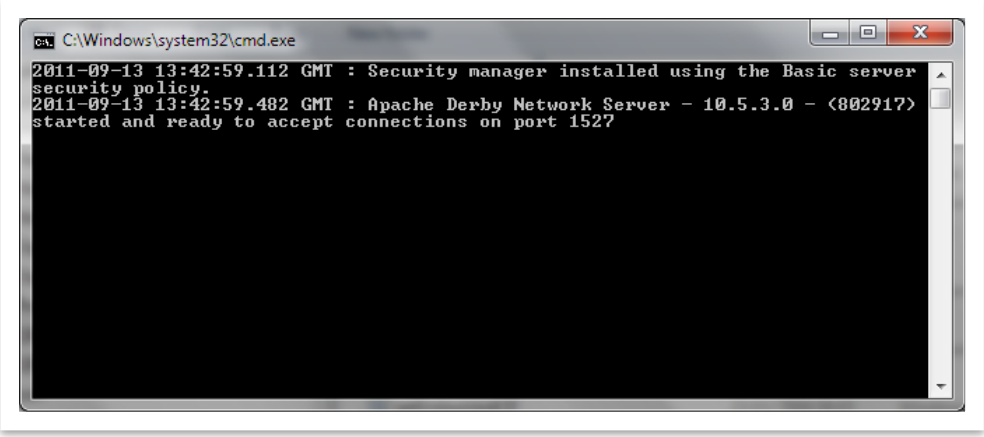
More detail on Java DB download and installation on:
http://www.oracle.com/technetwork/java/javadb/overview/index.html

5.3.1 Start Java DB

To start Java DB, run the batch file “startNetworkServer” (bat or sh) found in the directory …./javaDB/bin.



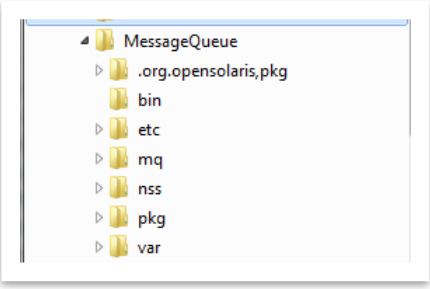
A command prompt opens. Please note the port used by the database. By default, the used port is 1527.



The database is now ready to support Open MQ HA cluster.

5.4 Install Open MQ

Download Open MQ from the site http://mq.java.net/ and install it. The installation creates the following hierarchy on your disk (windows).



On your disk, create a convenient directory to store Open MQ setting (ex: ..\JMSHAClustdirectory). In this directory, we will write the configuration files required to deploy an Open MQ HA Cluster.

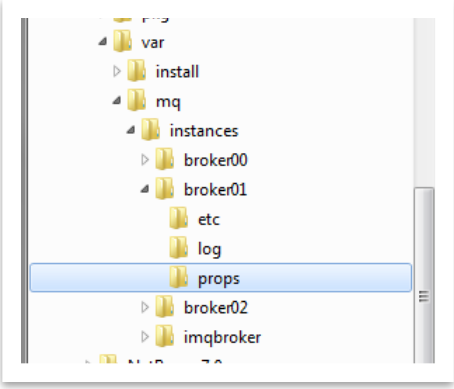
5.4.1 Setup Open MQ Configuration files

5.4.1.1 Open MQ starting process

A broker is started with the command "**imqbrokerd**". When Open MQ starts, it uses the following process to start up.

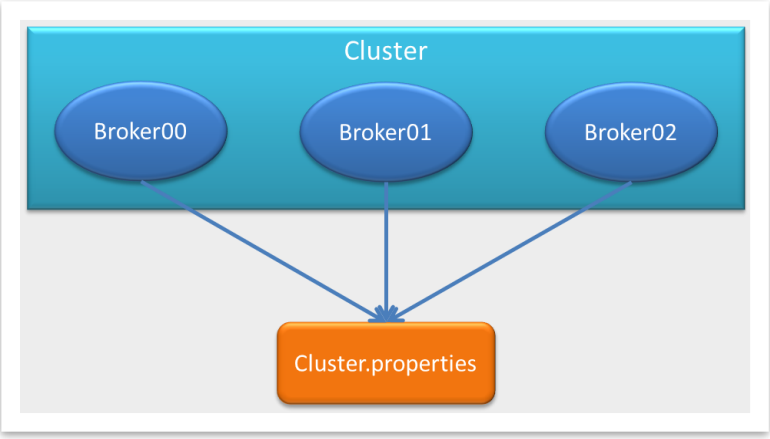
First "**imqbrokerd**" uses properties defined in the command line to set up the starting broker. Ex: with the command ***imqbrokerd -Dimq.brokerid=ID01 –name Broker01,*** the starting broker named "**Broker01**' is set up with an id=**ID01.** For more detail on imqbroker command, see: https://docs.oracle.com/cd/E19701-01/817-6024/strtbrkr.html

Then the broker reads the configuration defined in the file config.properties. By default, this file is created at the same time than the broker hierarchy and can be found within the directory props



In the config.properties file, you can set the property named imq.cluster.url. This property is used in a clustered configuration to create a shared properties file between the brokers in the same cluster.

"imq.cluster.url" allows a centralized management for the cluster configuration and avoids Copy and Paste traps. Common properties between brokers of the same cluster must be set in "imq.cluster.url".



In this POC, we set up the cluster by setting a few properties in the command line command and in a new file named "**cluster. properties**" (defined by imq.cluster.url).

On the command line, we set:

- The cluster file = `%CLUSTER_FILE%`
- The brokerID: used in the HA cluster, which identifies the broker = %`BROKER_NAME%`
- The broker's name = %`BROKER_NAME%`
- The broker port = `%BROKER_PORT%`

The command becomes:

```
%JMS_HOME%\imqbrokerd -Dimq.cluster.url=%CLUSTER_FILE% -Dimq.brokerid
=%BROKER_NAME% -Dimq.portmapper.port=%BROKER_PORT% -name %BROKER_NAME%
```

Use batch files to setup the variables regarding your configuration: Examples

```
REM this file is used to start a Broker Cluster
set JMS_HOME=F:\sun\MessageQueue\mq\bin
set CLUSTER_FILE=file:///G:/Customers/development/TestClusterJMS/cluster.properties
set BROKER_NAME=broker00
set BROKER_PORT=7600
REM start the broker
%JMS_HOME%\imqbrokerd -Dimq.cluster.url=%CLUSTER_FILE%
-Dimq.brokerid=%BROKER_NAME% -Dimq.portmapper.port=%BROKER_PORT% -name
%BROKER_NAME%
```

Batch file: broker00.bat

```
REM this file is used to start a Broker Cluster
set JMS_HOME=F:\sun\MessageQueue\mq\bin
set CLUSTER_FILE=file:///G:/Customers/development/TestClusterJMS/cluster.properties
set BROKER_NAME=broker01
set BROKER_PORT=7601
REM start the broker
%JMS_HOME%\imqbrokerd -Dimq.cluster.url=%CLUSTER_FILE%
--Dimq.brokerid=%BROKER_NAME% -Dimq.portmapper.port=%BROKER_PORT% -name
%BROKER_NAME%
```

Batch file: Broker01

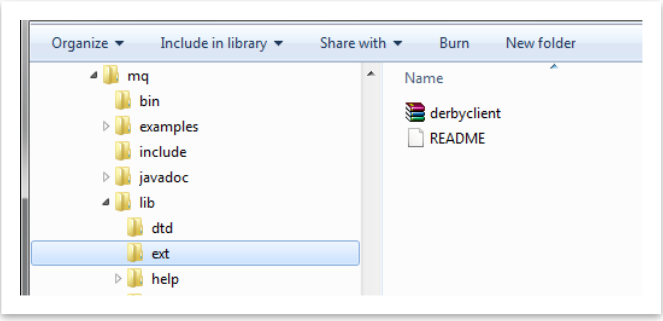
Configuration in the Cluster.properties is a bit more complex but remains simple. Set up the properties as follows

```
imq.instanceconfig.version=300
# Connection parameters
imq.cluster.ha=true
imq.cluster.clusterid=myCluster
# Database parameters
imq.persist.store=jdbc
imq.persist.jdbc.dbVendor=derby
imq.persist.jdbc.derby.createdburl=jdbc:derby://localhost:1527/imqdb;create=true
imq.persist.jdbc.derby.opendburl=jdbc:derby://localhost:1527/imqdb
imq.persist.jdbc.derby.closedburl=jdbc:derby://localhost:1527/imqdb
```

Property	Explanation
imq.cluster.ha=true	Indicates that the brokers run in High Availability mode
imq.cluster.clusterid=myCluster	Set the cluster name

imq.persist.store=jdbc	Mandatory in HA cluster
imq.persist.jdbc.dbVendor=derby *imq.persist.jdbc.derby.createdburl=jdbc:derby://localhost:1527/imqdb;create=true* *imq.persist.jdbc.derby.opendburl=jdbc:derby://localhost:1527/imqdb* *imq.persist.jdbc.derby.closedburl=jdbc:derby://localhost:1527/imqdb* *imq.persist.jdbc.derby.needpassword=false*	Use Java DB (derby) as cluster database.
imq.persist.jdbc.derby.driver=org.apache.derby.jdbc.ClientDriver	Don't forget to put the DB Client driver in …./mq/lib/ext

Don't forget to put the DB Client driver in …./mq/lib/ext



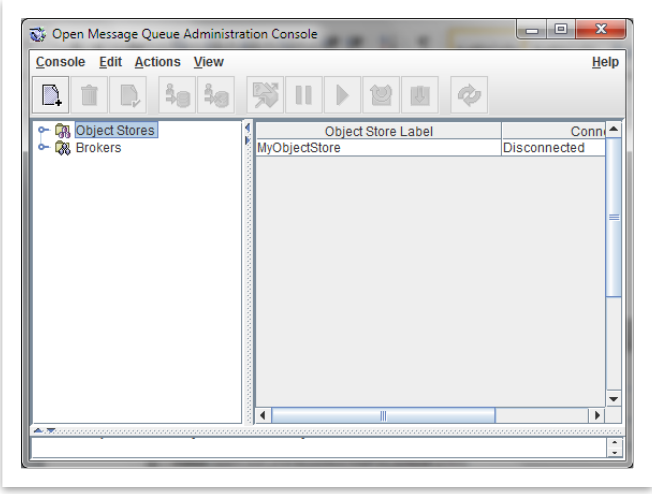
For our POC, we created 3 brokers in a HA cluster

Broker Name	Broker Address	Broker Id
Broker00	Localhost:7606	BrokerID00
Broker01	Localhost:7616	BrokerID01
Broker02	Localhost:7626	BrokerID02

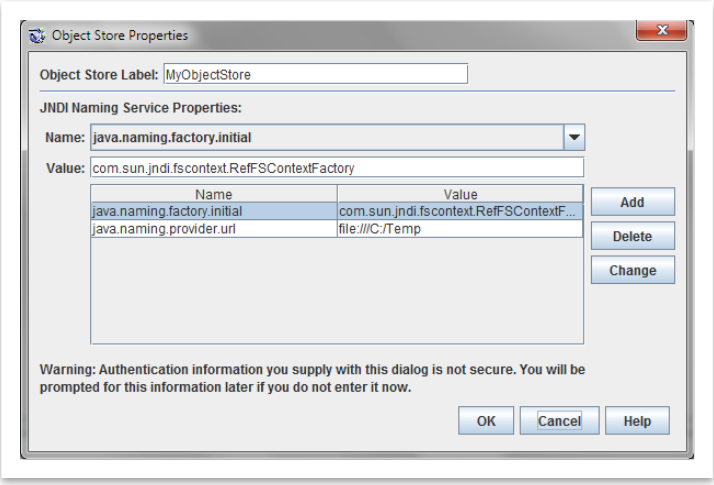
5.5 Create a File Objects Store

Open MQ provides a graphical tool "**imqadmin**" to create the Objects Store used by JMS BC to be connected to the broker.

You find this tool in the directory …/mq/bin. Just run "**imqadmin**".



Click on Actions→ Add Objects Store and create an object store as follows:



Define two Name Value properties:

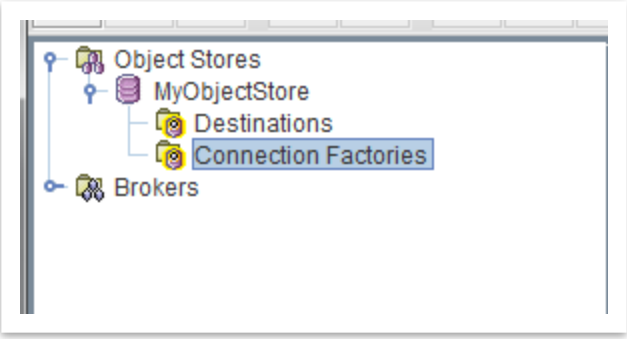
Name	Values
java.naming.factory.initial	com.sun.jndi.fscontext.RefFSContextFactory
Java.naming.provider.url	File///c:/temp *This URL is available on Windows. Set the url value regarding your OS and disk configuration.*

Select the Objects Store then click on Actions (Connect to Objects Store.

5.6 Create a connection factory

(see: http://java.sun.com/j2ee/1.4/docs/api/javax/jms/ConnectionFactory.html)

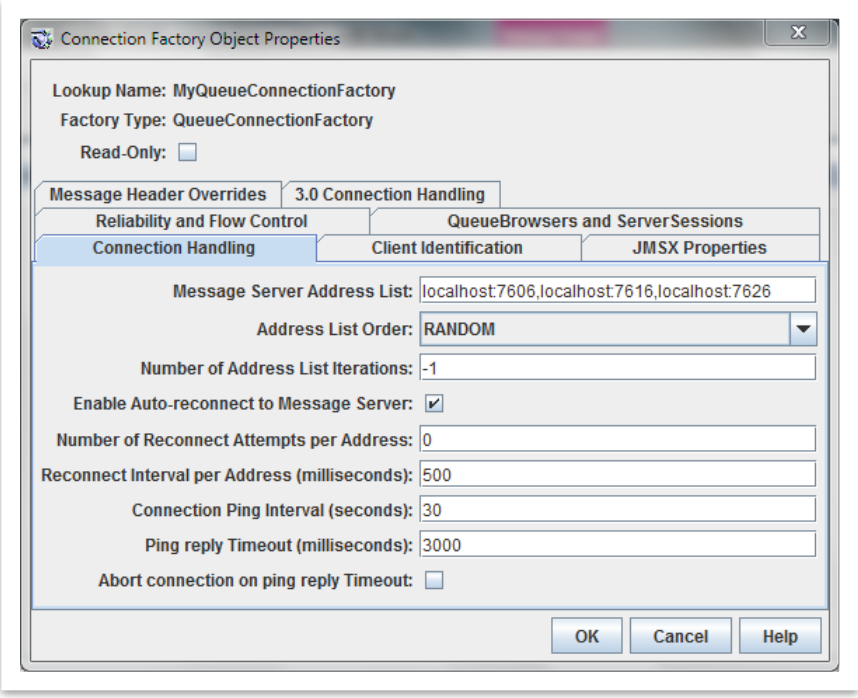
Select the item Connection Factories



Click on Actions→ Add Connection Factory Object

Name it MyQueueConnectionFactory and choose the type QueueConnectionFactory.

Fill the form as follows



Name	Values
Message Server Address List	Localhost:7606, Localhost:7616, Localhost:7626,
Address List Order	RANDOM
Number of Address List Iteration	-1
Enable Auto-Reconnect Attempts per Address	Check it
Number of Reconnect Attempts to Address	0
Reconnect Interval per Address	500
Connection Ping Interval	30
Ping Reply Timeout	3000

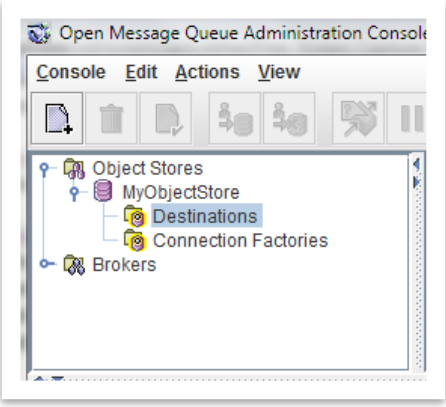
Abort connection on pin reply timeout	Uncheck

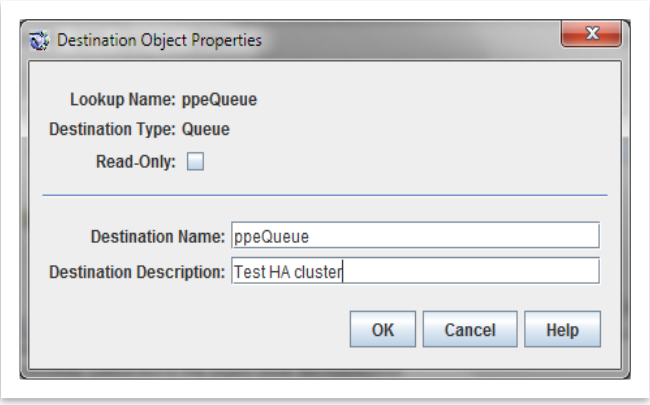
In the address list, put the broker list belonging to your cluster. With a HA cluster configuration, this list is **dynamically** updated when a broker defined in that list is added to or removed from the cluster. "Dynamically" means within the list defined here. If you add a fourth broker (ex: Localhost:7646) after starting your application, the binding component will not take it into.

5.7 Create a destination

You can define a destination (queue or Topic) through the console. Let's create a Queue and name it ppeQueue.

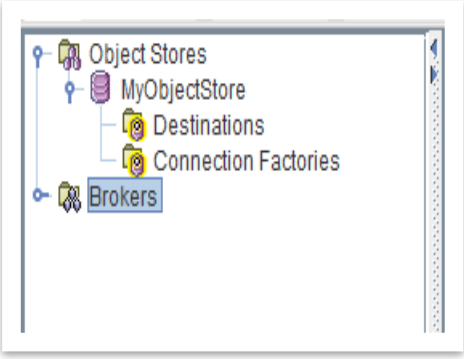
Select Destination. Click on Actions and select Add Destination Object. Feel the form as follows.

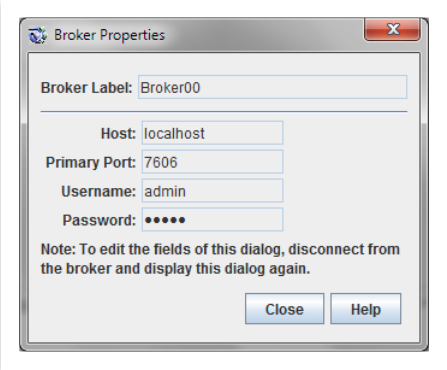




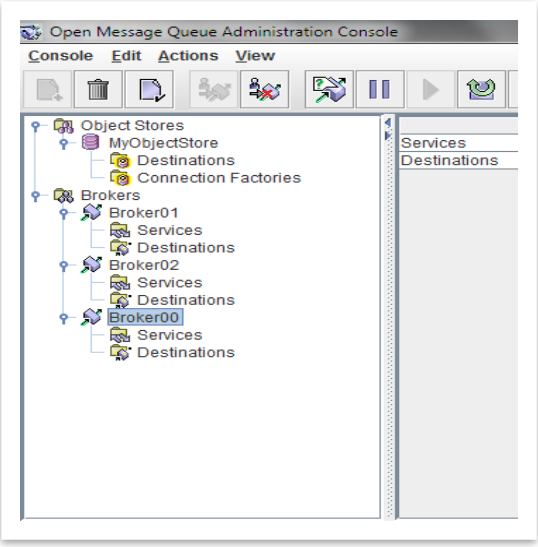
5.8 Connect the console to the brokers (Optional)

The console provides some interesting broker management. Click on the Brokers node, then, on action select "add broker".





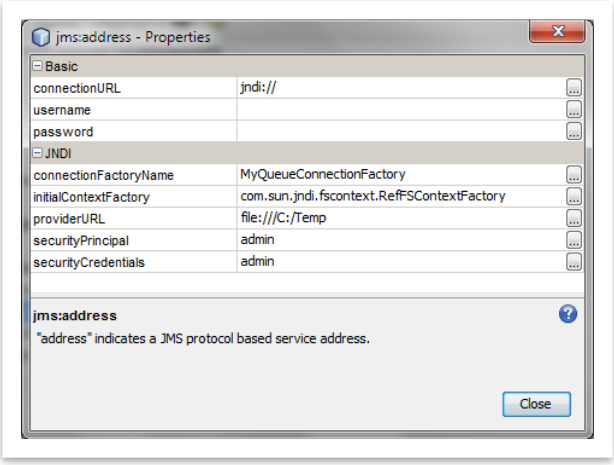
Redo the same operation for broker01 and broker02. Then the console is as follows.



Many administrative broker tasks can be started from the console: Pause, Stop, Restart the broker, change the port, change the log level, etc.

5.9 OpenESB JMS BC configuration with JNDI

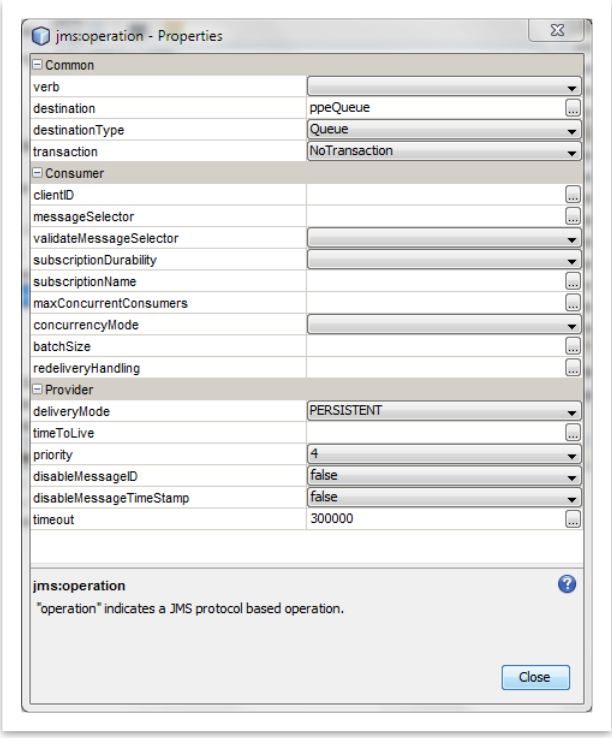
To access to the connection factories and the destination defined in the external object store, we need the reconfigure the concrete part of the WSDLs defined in our OpenESB applications. In the wsdl, set the jms:address as follows



Key	Value
ConnectionURL	JNDI://
Username	Empty
Password	Empty
ConnectionFactoryName	MyQueueConnectionFactory
initialContextFactory	com.sun.jndi.fscontext.RefFSContextFactory

ProviderURL	File:///c:/temp
securityPrincipal	Your login: admin by default
securityCredential	Your password: admin

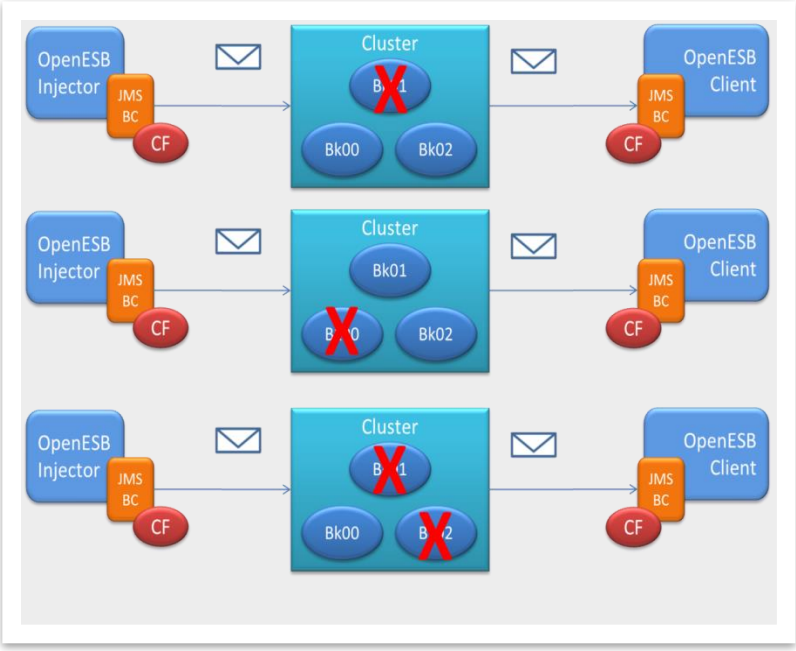
The element jms:operation must not be changed



The configuration is complete, and the test can be replayed.

5.10 Test the high availability of the cluster

Apply the same tests with this configuration and play with the high availability. Stop one broker, restart it. Stop two brokers and notice that you don't lose your messages.



Test	
Initial context	All brokers in the cluster are running
Test	We stop one or two brokers simultaneously for one or two minutes and then restart them. After a while we renew the same operation with different broker combinations.
Expectation	In case of failure, we expect that “Trigger” and “Client” will be able to reconnect to the cluster and all the messages created by the “Injector” will be present in the file.
Results	Just after a broker’s failure, the global process is stopped for few second. Then the process resumes. At the end of the test, all the messages generated by the injector are present in the file. We note that in some cases, some duplicated messages could appear.
Conclusion	HA is provided, but as expected in some cases duplicated messages appear. This is expected since distributed systems such As OpenESB provide a 1, N guarantee of delivery.

6 OpenESB connection to Apache ActiveMQ

6.1 Introduction

Apache ActiveMQ is an open source message broker written in Java together with a full Java Message Service (JMS) client. It provides "Enterprise Features" which in this case means fostering the communication from more than one client or server. Supported clients include Java via JMS 1.1 as well as several other "cross language" clients. The communication is managed with features such as computer clustering and ability to use any database as a JMS persistence provider besides virtual memory, cache, and journal persistency.

ActiveMQ supports JMS specifications and the JMS connection process defined by the specification. So, using JNDI, OpenESB JMS binding component sends to and receives from messages as it does with OpenMQ, Websphere MQ or else.

6.2 OpenESB Configuration

To connect to the broker, ActiveMQ client libraries must be added to the JMS BC classpath. If you use the OpenESB version 3.1.0 or more, you can add the jar files in the directory: ${OpenESB-HOME)/OE-Instance/libext/sun-jms-binding/activeMQ, else put the jar files to ${OpenESB-HOME)/OE-Instance/libext/sun-jms-binding.

We found simpler to add the jar file name activemq-all-XXX.jar for testing purposes. For a deployment on production, select the accurate jars for our application.

6.3 JNDI Configuration

The table below details the required JMS properties required to connect to Active MQ. Please, notice we give the ActiveMQ value per default. To change these values, please read ActiveMQ documentation to setup your broker and the OpenESB JMS BC configuration. http://activemq.apache.org/getting-started.html

Key	Value
ConnectionURL	JNDI://
Username	Empty
Password	Empty
ConnectionFactoryName	ConnectionFactory*
initialContextFactory	org.apache.activemq.jndi.ActiveMQInitialContextFactory
ProviderURL	tcp://myHost:61616
securityPrincipal	Your login: Empty by default
securityCredential	Your password: Empty by default

* (Case sensitive)

7 Hornetq

7.1 Introduction

HornetQ is an open source asynchronous messaging project from JBoss. It is an example of Message-oriented middleware. HornetQ is an open source project to build a multi-protocol, embeddable, very high performance, clustered, asynchronous messaging system. During much of its development, the HornetQ code base was developed under the name JBoss Messaging 2.0.

HornetQ supports JMS specifications and the JMS connection process defined by the specification. So, using JNDI, OpenESB JMS binding component sends to and receives from messages as it does with OpenMQ, Websphere MQ or else.

7.2 OpenESB Configuration

To connect to the broker, HornetQ client libraries must be added to the JMS BC classpath. If you use the OpenESB version 3.1.0 or more, you can add the jar files in the directory: ${OpenESB-HOME)/OE-Instance/libext/sun-jms-binding/hornetQ, else put the jar files to ${OpenESB-HOME)/OE-Instance/libext/sun-jms-binding.

The chapter “the client classpath2 in the HornetQ documentation list the required jars to use for a Java-JMS-JNDI client They are:

- hornetq-core-client.jar
- hornetq-jms-client.jar
- jboss-jms-api.jar
- jnp-client.jar

7.3 JNDI Configuration

The table below details the required JMS properties required to connect to HornetQ. Please, notice we give the HornetQ value per default. To change these values, please read HornetQ documentation to setup your broker and the OpenESB JMS BC configuration. https://docs.jboss.org/hornetq/. Notice that Hornet security is very restrictive by default and OpenESB user and role must be setup first.

Key	Value
ConnectionURL	JNDI://
Username	Empty
Password	Empty
ConnectionFactoryName	ConnectionFactory*
initialContextFactory	org.jnp.interfaces.NamingContextFactory
ProviderURL	jnp://localhost:1099
securityPrincipal	Your login: guest by default
securityCredential	Your password: guest by default

* (Case sensitive)

8 IBM Websphere MQ

IBM MQ is a family of network software products that IBM launched for the first time as an IBM product in December 1993. It was originally called MQSeries and was renamed WebSphere MQ in 2002 to join the suite of WebSphere products. In April 2014, it was renamed IBM MQ. The products that are included in the MQ family are IBM MQ, IBM MQ Advanced, IBM MQ Appliance, and IBM MQ for z/OS.

IBM MQ supports JMS specifications and the JMS connection process defined by the specification. So, using JNDI, OpenESB JMS binding component sends to and receives from messages as it does with OpenMQ, Active MQ or else.

8.1 OpenESB Configuration

To connect to the broker, IBM MQ client libraries must be added to the JMS BC classpath. If you use the OpenESB version 3.1.0 or more, you can add the jar files in the directory: ${OpenESB-HOME)/OE-Instance/libext/sun-jms-binding/ibmMQ, else put the jar files to ${OpenESB-HOME)/OE-Instance/libext/sun-jms-binding.

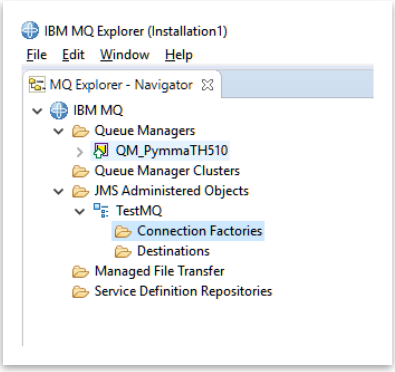
The jar files required to set up the connection are:

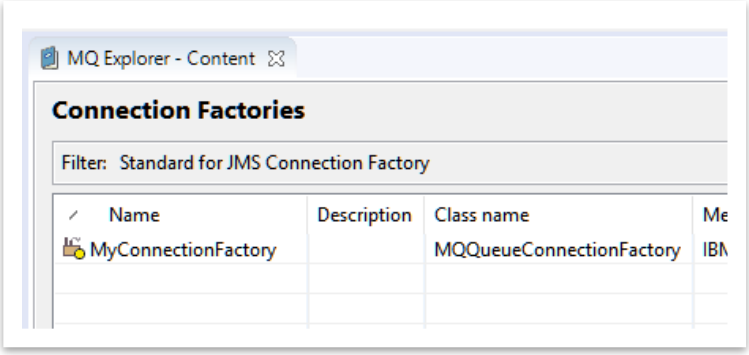
- com.ibm.mq.allclient.jar
- com.ibm.mq.traceControl.jar
- fscontext.jar
- jms.jar
- providerutil.jar

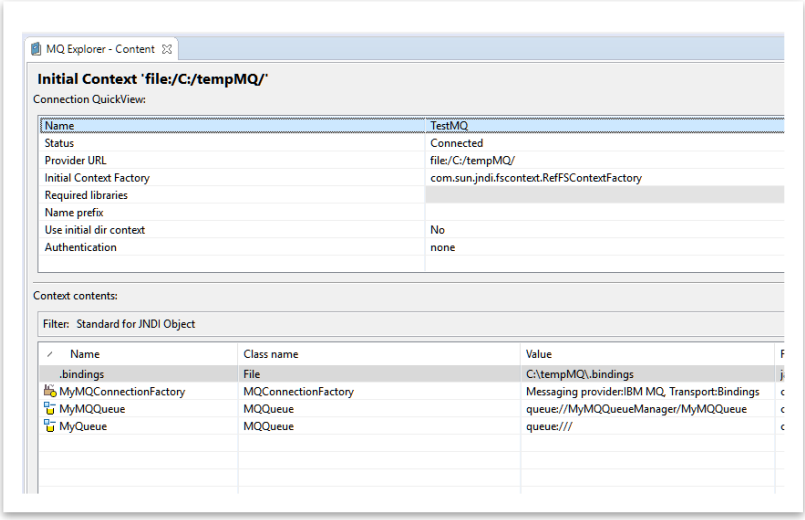
8.2 JNDI Configuration

To access to a queue or a topic hosted by IBM MQ, an Administrative work is required. First you need to Add an Initial context in the administrated object. Then, when you create a Queue or a Topic, you must create a “Matching JMS Queue”. Please refer to IBM MQ documentation for further information.

To make our example easier, the JNDI initial context is in the file system but you can use a LDAP or any other implementation







Key	Value
ConnectionURL	JNDI://
Username	Empty
Password	Empty
ConnectionFactoryName	MyConnectionFactory
initialContextFactory	com.sun.jndi.fscontext.RefFSContextFactory*
ProviderURL	file:///c:/temp
securityPrincipal	Your login
securityCredential	Your password

9 Test your parameters

Each JMS broker has its own way of working and it is often complex to evaluate if an unsuccessful connection to the broker is due to the parameters, the libraries, the parameter or the OpenESB JMS BC.

To test the parameters and the required library, try them following Java code.

```
package com.pymma.openesb.testMyMQ;
import java.util.Properties;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
public class TestMyMQ {
   public static String INITIAL_CONTEXT_FACTORY = "com.sun.jndi.fscontext.RefFSContextFactory";
   public static String PROVIDER_URL= "file:///C:/tempMQ/";
   public static String SECURITY_AUTHENTICATION    = "";
   public static String SECURITY_CREDENTIALS      = "";
   public static void main (String args[]) throws Exception {
      Properties props = new Properties();
      props.setProperty(Context.INITIAL_CONTEXT_FACTORY,TestMyMQ.INITIAL_CONTEXT_FACTORY);
      props.setProperty(Context.PROVIDER_URL,TestMyMQ.PROVIDER_URL);
      props.setProperty(Context.SECURITY_AUTHENTICATION, TestMyMQ.SECURITY_AUTHENTICATION);
      props.setProperty(Context.SECURITY_CREDENTIALS, TestMyMQ.SECURITY_CREDENTIALS);
      javax.naming.Context ctx = new InitialContext(props);
      ConnectionFactory factory = (ConnectionFactory) ctx.lookup("MyConnectionFactory");
      Connection connection = factory.createConnection();
      Session session = connection.createSession(false, QueueSession.AUTO_ACKNOWLEDGE);
      Queue queue = (Queue) ctx.lookup("MyQueue");
      connection.start();
      MessageProducer producer = session.createProducer(queue);
      Message message = session.createTextMessage("This is a text message");
      producer.send (message);
      System.out.println("Message sent");
   }
}
```

10 Conclusion

In this tutorial, we learn how to connect the component JMS to JMS broker. We defined two ways to do it. The first relies on JMS-JCA and the second on JNDI. At the design time and at the runtime, we advise OpenESB users to use the JNDI configuration that is more flexible and complete than the JMS-JCA.

Feel free to send use your feedback at contact (at) pymma.com

11 What's next

11.1 Reading

For a better understanding of component and Service assembly, please have a look at JBI specifications.

For Application configurations and application variable please have a look on our document 770-006 OpenESB Enterprise Edition Multiple environment.

11.2 OpenESB Enterprise Edition

Congratulation, you succeed in deploying OpenESB for you Proof of Concept and development. Your next step is a deployment on QA / Test and production. OpenESB Enterprise Edition comes with advanced features for your project and overall a professional technical. It is a guarantee for you and your company to get the best of OpenESB. For more detail on OpenESB Enterprise Edition please have a look on our web site www.pymma.com or contact us at contact@pymma.com

12 Help and support

12.1 From the community

You can find OpenESB documentation on the OpenESB official web site:
www.open-esb.net.

You want to ask questions or share your feedback, used OpenESB forum at:
http://**openesb**-community-forum.794670.n2.**nabble**.com

Notify a bug or propose an improvement on OE JIRA:
https://openesb.atlassian.net/secure/Dashboard.jspa

12.2 From Pymma

Today, Pymma is certainly the best knowledge centre on OpenESB. Since 2010 around 80% of the new OpenESB features come from our labs. Deeply involved in the community and proposes services and consulting on OpenESB.

Pymma offers a professional service that assists you during the design, the implementation and the ongoing management of your service-oriented integration project. Our skill and background comes from years on extensive real-world experience, industry-leading methods and best practices on OpenESB.

Feel free to contact us by email at contact@pymma.com for any further information on our OpenESB Services.