# INDEX

JMS

Point-to-Point Message Domain

Glossary

| Abbrev. | Term | Description |
| --- | --- | --- |
| JMS | Java Messaging Service | Java message service enables loosely coupled communication between two or more systems. It provides reliable and asynchronous form of communication. |
| EIS | Enterprise Information System | Enterprise Information System (EIS) such as IBM WebSphere MQ or DB2®. |
| JCA | Java EE Connector Architecture | The Java™ Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) provides a standard way of connecting applications running in a Java EE environment to an Enterprise Information System (EIS) such as IBM WebSphere MQ or DB2®. |
| MDB | Message-driven bean | Message-driven beans **are** a special class of Enterprise Java Bean (EJB). They enable Java Platform, Enterprise Edition (JEE) applications to process incoming messages asynchronously, with WebSphere Application Server managing the transnationality and concurrency of the application. |
| SIB | Service Integration Bus | Service integration is a set of technologies that provides asynchronous messaging services. Application servers or clusters of application servers in a WebSphere Application Server cell can cooperate to provide asynchronous messaging services. Service integration provides asynchronous messaging services, and a group of servers or clusters that cooperate in this way is called a service integration bus. The application servers or server clusters in a bus are known as bus members. You can also add bus members that are WebSphere MQ servers; service integration uses these bus members to write messages to, and read messages from, WebSphere MQ queues. |

# What is messaging?

The term messaging, in the generic sense, is usually used to describe the exchange of information between two interested parties. In the context of computer science, messaging can be used to loosely describe a broad range of mechanisms used to communicate data. For example, e-mail and instant   
messaging are two communication mechanisms that could be described using the term messaging. In both cases, information is exchanged between two parties, but the technology used to achieve the exchange is different. There are two messaging types that define the mode of interaction between the sending and receiving applications:

* **Synchronous messaging**

Synchronous messaging involves tightly coupled processes, where the sending and receiving applications communicate directly and both must be available for the message exchange to occur.

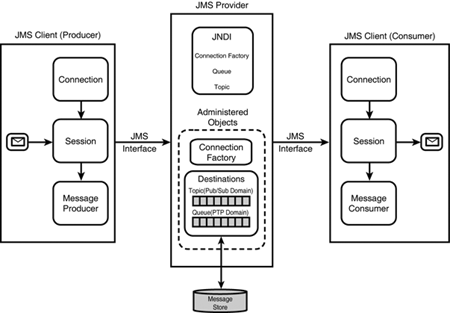
* **Asynchronous Messaging**

Asynchronous messaging involves loosely coupled processes, where the sending and receiving applications communicate through a messaging provider. The sending application is able to pass the data to the messaging provider and then continue with its processing. The receiving application can connect to the messaging provider, possibly at some later point in time, to retrieve the data.

# What is JMS?

Java message service enables loosely coupled communication between two or more systems. It provides reliable and **asynchronous form** of communication. WebSphere Application Server supports asynchronous messaging through the use of the Java Message Service (JMS).

The JMS is a specification and not an implementation. The JMS specification provides the guidelines of what to implement and how the Java based applications can interact with each other/non-Java based clients to create, send, receive and read the messages. It is the vendors who adheres to these specifications and defines how these messaging system needs to be implemented. They use the **JMS API** [javax.jms](http://docs.oracle.com/javaee/1.3/api/index.html).



## Components of JMS Architecture

**There are four elements** of the JMS architecture which are explained the below table:

|  |  |
| --- | --- |
| **Elements** | **Description** |
| JMS Provider | Vendors implement the JMS specification by supplying a JMS provider consisting of libraries that implement the JMS interfaces. This provides the core functionality for routing and delivering messages between clients, manages the administered objects and also manage, monitor, and tune the messaging service. The JMS provider ensures asynchronous communication, supports reliable message delivery and transaction control. |
| JMS Clients | The JMS clients act as a message sender or receiver. The JMS clients make API calls to send message to the queue/ topic which are managed by the JMS providers. The message receivers on the other hand receive messages from these queue/ topic. |
| Messages | The messages are objects which are used to communicate information between the clients. The JMS messages has 3 parts: (i) Header (ii) Properties (iii) Body |
| Administered objects | Administered objects are for client use, but are created by an administrator. There are two main administered objects—destinationand connectionfactories. These objects are managed by the JMS provider. The destinations are nothing but references to the physical implementation of Queues or Topics. |

The architecture diagram highlights an app server implementing the JMS specifications and messaging services. These App servers also have their own JNDI implementations.

1. The administrator creates the **physical destination** (Queues in case of PTP or Topic in case of Pub/Sub model) in the **JMS providers** from the JMS admin console. This is actually a physical location in the server.
2. In the JNDI namespace the administrator creates two administered objects **i) Destination ii) Connection factories**. The administrator configures/ binds the destination administered objects with the corresponding physical location on the JMS provider. In this example we are trying to map the physical location of the Queue1 to an object reference Q1, Queue2 to Q2.The connection factories administered objects are created in a similar way. The administrator creates/ configures a connection factory administered object through the admin console.
3. The **message sender/ producer (JMS Client)** uses the connection factory to create a new connection with the provider. It then uses a standard JNDI call to look up for the administered object queue (in this example it is referred in the JNDI namespace as Q1) for sending messages.
4. At this stage the message sender establishes a logical connection between the JMS client and the Queue in the JMS provider (in this example JMS Client A secures a logical connection with Queue 1) and is now ready for sending message. Once the background has been set the Sender sends the message to the physical destination (Queue 1).
5. The message consumer uses a JNDI call to look up the destination administered object that points to the corresponding physical destination from which it expects to get messages.
6. In this case as well the message consumer/receiver gets the message from the physical destination (Q1).

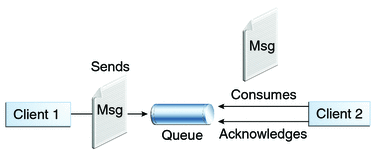
## Types of Messaging Model in JMS

There are two types of messaging models in JMS.

1. Point to Point Messaging Domain
2. Publish/Subscribe Messaging Domain

## Point-to-Point Messaging Domain

Applications are built on the concept of message queues, senders, and receivers. Each message is send to a specific queue, and receiving systems consume messages from the queues established to hold their messages. Queues retain all messages sent to them until the messages are consumed by the receiver or expire. Here there is only one consumer for a message. If the receiver is not available at any point, message will remain in the message broker (Queue) and will be delivered to the consumer when it is available or free to process the message. Also, receiver acknowledges the consumption on each message.



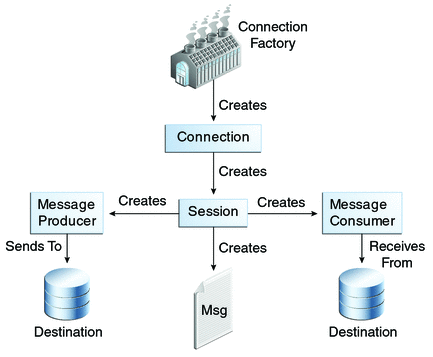
## Publish/Subscribe Message Domain

Applications send message to a message broker called Topic. This topic publishes the message to all the subscribers. Topic retains the messages until it is delivered to the systems at the receiving end. Applications are loosely coupled and do not need to be on the same server. Message communications are handled by the message broker; in this case it is called a topic. A message can have multiple consumers and consumers will get the messages only after it gets subscribed and consumers need to remain active in order to get new messages.



## Sending and Receiving Message

Administered objects (ConnectionFactory, Connection, Queue or Topic, Session, Message Producer/Sender and Message Consumer/Receiver) shown in the diagram used to sending and receiving message)

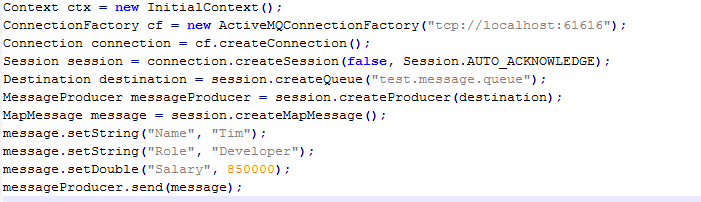


## Message Sender (Message Producer)

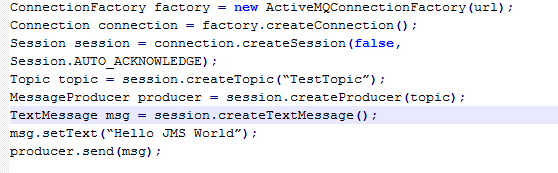
Message Sender object is created by a session and used for sending messages to a destination queue. It implements the MessageProducer interface.

First, we need to create a connection object using the ActiveMQConnectionFactory factory object. Then we create a session object.  Using the session object, we set the message broker (Queue) and create the message sender object. Here we are sending a map message object. Please see the code snippet for message sender.

**Example of sending message on Queue**



**Example of sending message on Topic**

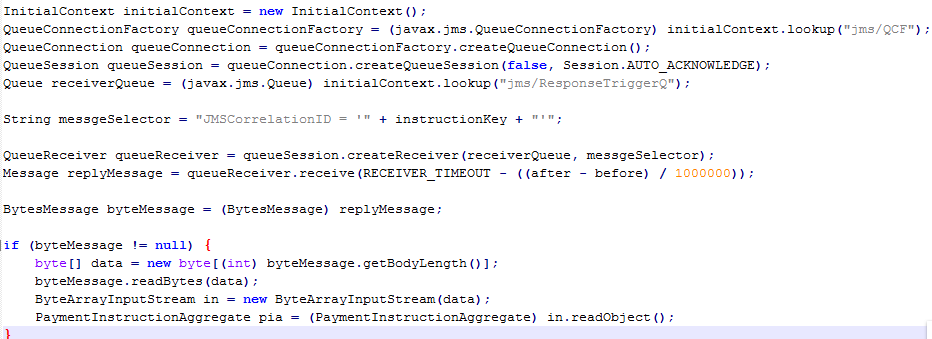


## Message Receiver (Message Consumer)

Message Receiver object is created by a session and used for receiving messages from a queue. It implements the MessageProducer interface. The JMS messages are consumed by the clients / receivers from the destination (either queue/ topic) in the following ways:

## Synchronous Mode (Receiving message in synchronous mode)

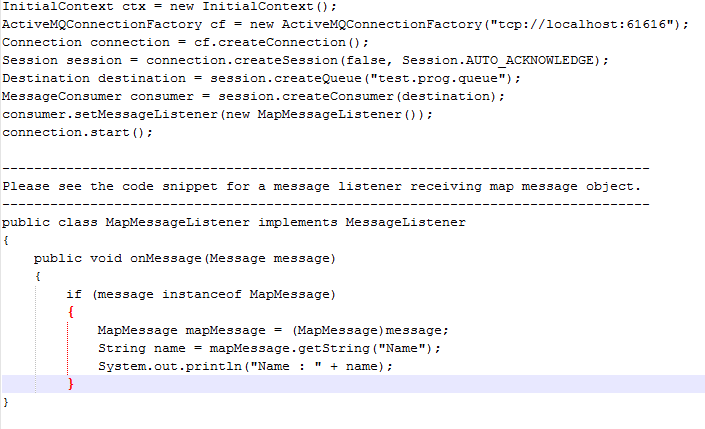
In this case the receiver fetches the message from the destination through receive() method of the javax.jms package. This method call blocks indefinitely until a message is produced or until this message receiver is closed. The receiver does not perform any other transactions during this time. However, we can also specify a timeout value through receive(timeout) method which means receiver does not perform any other transactions until a message arrives or receiver is closed within the timeout value. If there is no message received till the timeout value then the timeout expires and receiver is released from the block state. A timeout of zero never expires, and the call blocks indefinitely i.e. receive(0) is similar to receive().



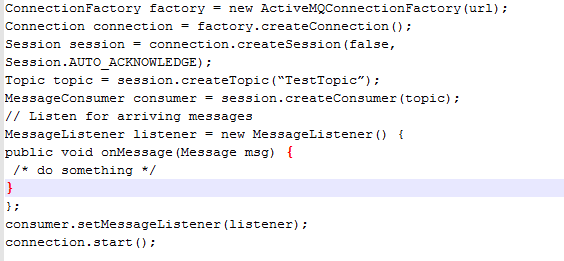
## Asynchronous Mode (Receiving message in asynchronous mode)

In this case the JMS client/ subscriber uses a messageListener object to receive messages asynchronously delivered messages. Message Listener remains active and gets invoked when the receiver consumes any message from the broker. Whenever a message arrives at its destination, the JMS provider/ message server delivers the message to the subscriber by using the *onMessage*() method. The client will not be blocked while waiting for the message.

### **Example of receiving message from Queue in Asynchronous Mode**



### **Example of receiving message from Topic in Asynchronous Mode**



## JMS with WebSphere Application Server

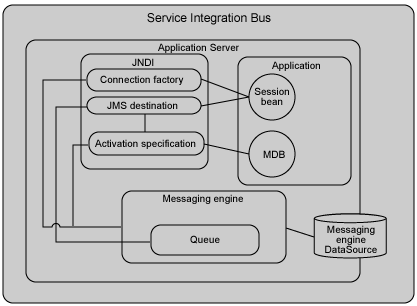
WebSphere Application Server supports asynchronous messaging through the use of the Java Message Service (JMS). The JMS API is the standard Java API for accessing enterprise messaging systems from Java programs. In other words, it is a standard API that sending and receiving applications written in Java can use to access a messaging provider to create, send, receive, and read messages. WebSphere Application Server V7.0, V8.0 and V8.5 supports JMS 1.1 applications.

Support for JMS 2.0 was added in WebSphere Application Server V9.0.

## What is Service Integration Bus?

Service integration is a set of technologies that provides asynchronous messaging services. Application servers or clusters of application servers in a WebSphere Application Server cell can cooperate to provide asynchronous messaging services. Service integration provides asynchronous messaging services, and a group of servers or clusters that cooperate in this way is called a service integration bus. The application servers or server clusters in a bus are known as bus members. You can also add bus members that are WebSphere MQ servers; service integration uses these bus members to write messages to, and read messages from, WebSphere MQ queues.

Different service integration buses can, if required, be connected. This allows applications that use one bus (the local bus) to send messages to destinations in another bus (a foreign bus). Note, though, that applications cannot receive messages from destinations in a foreign bus.



## Message Engines

A messaging engine is a component, running inside a server, that manages messaging resources for a bus member. Applications are connected to a messaging engine when they access a service integration bus.

Each service integration server or cluster bus member contains a component called a messaging engine that processes messaging send and receive requests and that can host destinations. To host queue-type destinations, the messaging engine includes a message store where, if necessary, it can hold messages until consuming applications are ready to receive them, or preserve messages in case the messaging engine fails.  
  
If the bus member is a server cluster, it can have additional messaging engines to provide high availability or workload sharing characteristics. If the bus member is a WebSphere MQ server, it does not have a messaging engine, but it lets you access WebSphere MQ queues directly from WebSphere MQ queue managers and (for WebSphere MQ for z/OS®) queue-sharing groups.

## Messaging Providers

WebSphere Application Server applications invoke asynchronous messaging services by using the Java Messaging Service (JMS) application programming interface (API) to interface to a messaging provider. WebSphere Application Server supports a variety of JMS messaging providers, including service integration (which is the default messaging provider) and WebSphere MQ as an external JMS messaging provider.

## WebSphere MQ Messaging Provider and WebSphere MQ Resource Adaptor

**WebSphere® MQ messaging provider** in WebSphere Application Server, Java Message Service (JMS) messaging applications can use your WebSphere MQ system as an external provider of JMS messaging resources.

**WebSphere MQ Resource Adaptor** allows applications running in an application server to access IBM® WebSphere® MQ resources. It supports inbound and outbound communication.

The **Java™ Platform, Enterprise Edition (Java EE) Connector Architecture (JCA)** provides a standard way of connecting applications running in a Java EE environment to an Enterprise Information System (EIS) such as IBM WebSphere MQ or DB2®.

The IBM WebSphere MQ resource adapter supports two types of communication between an application and a queue manager:

* **Outbound communication**

An application starts a connection to a queue manager, and then sends JMS messages to JMS destinations and receives JMS messages from JMS destinations in a synchronous manner.

* **Inbound communication**

A JMS message arriving at a JMS destination is delivered to an MDB, which processes the message asynchronously.

## MDB and Activation Specifications

**Message-driven beans** are a special class of Enterprise Java Bean (EJB). They enable Java Platform, Enterprise Edition (JEE) applications to process incoming messages asynchronously, with WebSphere Application Server managing the transnationality and concurrency of the application.

The MDB application uses the activation specification to connect to a WebSphere MQ queue manager for the processing of **inbound** **messages**.

**Activation specifications** are part of the JCA 1.5 specification. Activation specifications are the standardized way to manage and configure the relationship between an MDB running in WebSphere Application Server and a destination(queue) within WebSphere MQ.

Activation specifications are used by the application's message-driven bean to connect to the queue and receive messages.

Activation specifications are used to configure inbound message delivery to message-driven beans (MDBs) running inside WebSphere® Application Server. They supersede message listener ports, which are now a stabilized function. The activation specification also provides other options, such as security settings.

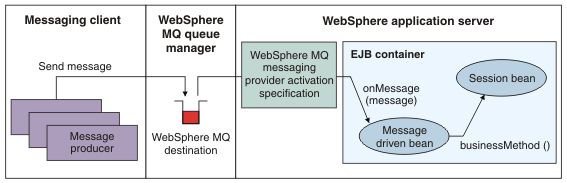
## Understanding the relationship between MDB and Activation Specification

The following figure shows how an activation specification can be used to link a WebSphere MQ queue manager destination to an MDB running within WebSphere Application Server.

WebSphere MQ Messaging Provider is the message provider with which we can create the Activation specification if we are using WebSphere MQ manager.

The process of delivering a message from a client to an MDB via a WebSphere MQ messaging provider activation specification occurs in this way:

*Figure 1. WebSphere MQ messaging provider activation specification in action*



* A messaging client, either running in a stand-alone process or within an application server environment, sends a message using JMS (or any other messaging API, such as MQI) to a WebSphere MQ queue or topic defined in a WebSphere MQ queue manager.
* A WebSphere MQ activation specification is configured to listen on that destination for messages. When the new message is detected, it is removed from the destination (potentially under an XA transaction).
* The message is then passed to an MDB that has been configured to use the activation specification through its onMessage method.
* The MDB uses the information in the message to perform the relevant business logic.

## Configure MDB with Activation Specification

MDB is treated as a JCA resource. JCA specification incorporated messaging framework APIs as well. Flow in case of JCA is:-

incoming message --> listened by Message listener --> Resource Adapter-->deliver to MDB

In case of WebSphere, you can use either SIB (Service Integration Bus) destinations for messaging OR external software like WebSphere MQ for messaging.

Step1: Define MDBBean

public abstract class **FAATInboundRequestMDBBean**implements MessageDrivenBean, MessageListener {

public void onMessage(Message msg) {

//do something

}

}

Step 2: Define MDB Bean and queue in ejb-jar.xml

<message-driven>

<ejb-name>FAATInboundRequestMDB</ejb-name>

<ejb-class>

com.clear2pay.siamph.faat.ejb.FAATInboundRequestMDBBean

</ejb-class>

<messaging-type>javax.jms.MessageListener</messaging-type>

<transaction-type>Container</transaction-type>

<message-destination-type>

javax.jms.Queue

</message-destination-type>

<activation-config>

<activation-config-property>

<activation-config-property-name>

acknowledgeMode

</activation-config-property-name>

<activation-config-property-value>

Auto-acknowledge

</activation-config-property-value>

</activation-config-property>

<activation-config-property>

<activation-config-property-name>

destinationType

</activation-config-property-name>

<activation-config-property-value>

javax.jms.Queue

</activation-config-property-value>

</activation-config-property>

<activation-config-property>

<activation-config-property-name>

subscriptionDurability

</activation-config-property-name>

<activation-config-property-value>

Durable

</activation-config-property-value>

</activation-config-property>

</activation-config>

<env-entry>

<description>

</description>

<env-entry-name>config-file</env-entry-name>

<env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>config/siamph/faat-request.xml</env-entry-value>

</env-entry>

<env-entry>

<description>

</description>

<env-entry-name>service-id</env-entry-name>

<env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>faat-request</env-entry-value>

</env-entry>

<ejb-local-ref>

<description />

<ejb-ref-name>ejb/LocatorService</ejb-ref-name>

<ejb-ref-type>Session</ejb-ref-type>

<local-home>

com.clear2pay.bph.opfcommon.locator.ejb.LocatorServiceLocalHome

</local-home>

<local>

com.clear2pay.bph.opfcommon.locator.ejb.LocatorServiceLocal

</local>

<ejb-link>LocatorService</ejb-link>

</ejb-local-ref>

<security-identity>

<description />

<run-as>

<description />

<role-name>bpelOperator</role-name>

</run-as>

</security-identity>

</message-driven>

<container-transaction>

<method>

<ejb-name>FAATInboundRequestMDB</ejb-name>

<method-name>onMessage</method-name>

<method-params>

<method-param>javax.jms.Message</method-param>

</method-params>

</method>

<trans-attribute>Required</trans-attribute>

</container-transaction>

<resource-ref id="ResourceRef\_FAATInboundRequestQ">

<description>

</description>

<res-ref-name>jms/FAATInboundRequestQ</res-ref-name>

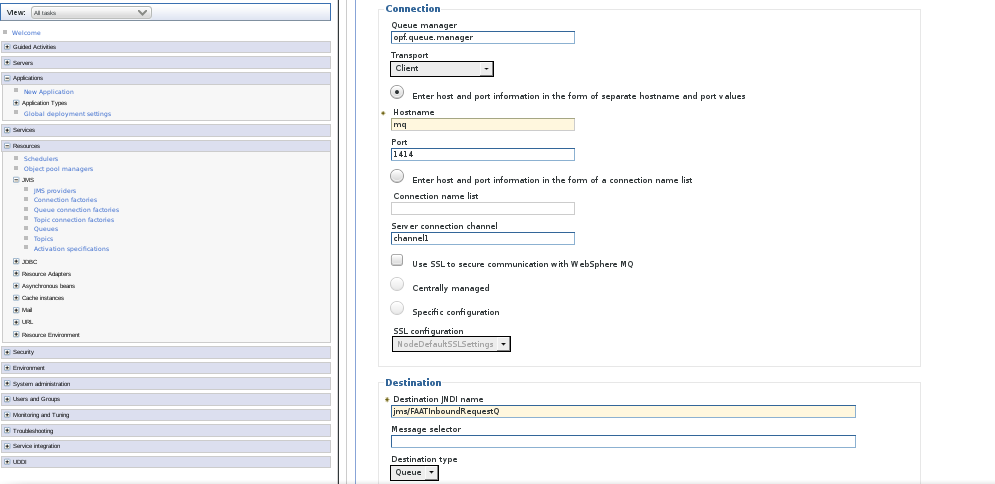
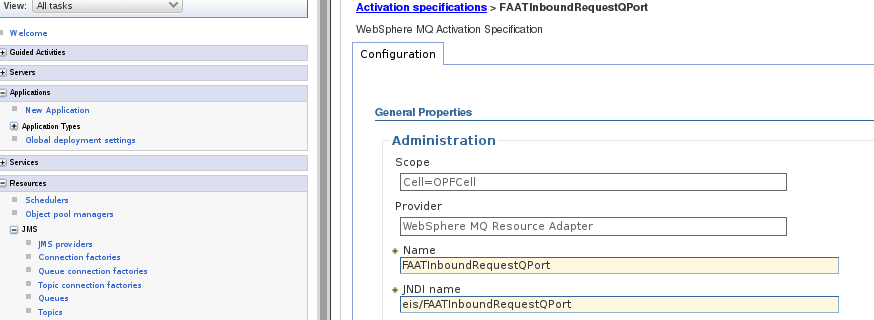
<res-type>javax.jms.Queue</res-type>

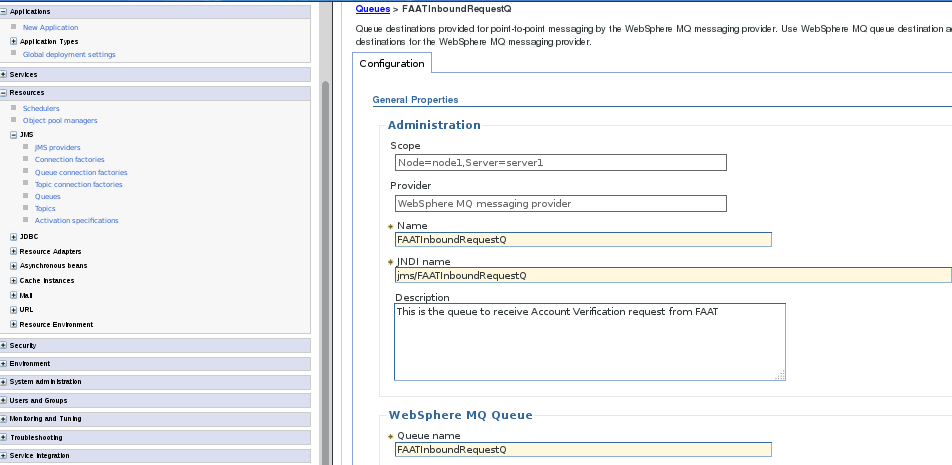
<res-auth>Container</res-auth>

<res-sharing-scope>Shareable</res-sharing-scope>

</resource-ref>

Step 3: Define Activation Specification



Step 4: Define queue name with JNDI name in WAS which is actually created physically in WebSphere Queue Manager

Step 5: Binding with MDB with Activation Specification(ibm-ejb-jar-bnd.xml)

When you are using the EJB binding file, the **activation-spec-binding-name** attribute in the ibm-ejb-jar-bnd.xml file must point to the activation specification property ID value that is specified in the server.xml file.

<ejb-jar-bnd>

<message-driven name="FAATInboundRequestMDB">

<jca-adapter activation-spec-binding-name="eis/FAATInboundRequestQPort" />

<ejb-ref name="ejb/LocatorService"

binding-name="ejblocal:ejb/com/clear2pay/bph/opf/locator/ejb/LocatorServiceHome" />

</message-driven>

<session name="LocatorService"

local-home-binding-name="ejblocal:ejb/com/clear2pay/bph/opf/locator/ejb/LocatorServiceHome">

<resource-ref name="jms/FAATInboundRequestQ" binding-name="jms/FAATInboundRequestQ" />

</session>

</ejb-jar-bnd>

## JMS Connection Factory, Connection pools and session pools in the WebSphere Application Server

**Q: WebSphere MQ Queue Connection Factories in WebSphere Application Server releases contain both a connection pool and a session pool for configuration. When configuring the session pool, is this configuration per each connection? For example, if my connection pool has a maximum size of 10, and my session pool also has a maximum size of 10, does that mean that I have a total of 10 sessions available per connection (which would make this 100 sessions for 10 connections) or does this mean that there are only 10 sessions available to be used among 10 connections?**

A: The session pool setting applies to each JMS connection as this is the factory for sessions, so you can have a maximum of 10 sessions for each connection and a maximum of 10 connections. So, in total, this mean you might have 100 channels (connections) open to the MQ Server.  
  
  
**Q: What happens if my session pool has a maximum of 10 session and connections and I attempt to create a new session from a connection that exceeds this maximum session and connection? Does the pool grow? Is an exception thrown? If so, what kind of exception?**  
  
A: When the maximum number of connections and sessions is reached, and a request for a new connection or session is received, the pool manager waits for a period of time defined in the Connection timeout property for an available physical connection. If a connection is not available in the time period defined by the Connection timeout property, the Pool manager throws a ConnectionWaitTimeoutException. This is documented in more detail in the Connection timeout section on the Session pool settings page of the WebSphere Application Server Information Center.  
  
  
**Q: When closing a QueueConnection JMS in WebSphere Application Server with Connection Pooling configured, what exactly happens?**  
  
A: When a connection is closed, it is returned to the connection pool, and the session to its session pool. Any QueueSender and QueueReceiver objects associated with the session are destroyed.

**Q: From a design perspective, is there a formula regarding the sizing of the WebSphere MQ JMS connection pool and JMS session pool with respect to the number of concurrent requests?**

A: No formula exists. However, we recommend that you set the Max Connections property for your connection pool to a value about 50% higher than your typical JMS connection concurrency, and the Max Connections property for your session pool to approximately 50% more than the average number of concurrent sessions requested on a JMS Connection. The requirements really depend on the design of your application; in particular, how it uses JMS connections and sessions. You should also remember that the Max Connections property for your session pool is the maximum number of sessions that can be made from a single JMS Connection, because a JMS connection is a factory for sessions.   
  
  
**Q: How can I utilize a WebSphere MQ JMS session pool without having to manage the JMS connections in the application? It appears that if I have a JMS connection pool and a JMS session pool, I need to keep track of my connections in some collector object to use the number of sessions I have configured for use with each connection. How do you make use of those sessions in the session pool without having to maintain your own programmatic list of connections that are configured in the connection pool? For example, having retrieved a connection from the connection pool, what is the approach for creating sessions for multiple users without having to maintain that connection reference and a collector object in my application?**

A: JMS connections are non-shareable. Therefore, unless your application does something like caching a connection in a static class variable (which is definitely not recommended in practice), an Enterprise JavaBeans™ method has only a single JMS connection/JMS session pair per EJB method thread. Currently, the JCA Connection Manager does not support sharing of non-transactional resources such as a JMS connection and so that must be handled in your application code. Note that, from a WebSphere MQ perspective, a JMS connection is very lightweight. It is the JMS session that is the heavyweight object. Also, be aware that in the Sun J2EE™ 1.4 specification, an Application Server must enforce an application so that it can only create a single JMS session from a JMS connection. A possible reason behind the Sun decision to do this is because of Connection Management considerations, and the fact that in J2EE 1.4, JMS providers are advised to be defined as JCA Resource Adapters.