Oracle Banking APIs

JMS Configuration Multi Entity Guide

Part No. E97825-01

June 2018

ORACLE®

JMS Configuration Multi Entity Guide

June 2018

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001
www.oracle.com/financialservices/

Table of Contents

JMS Configuration for external System

	Name
Author	OBAPI Development team
Current Version	1
Date	

Revision History

Version	Updates	Author	Date
Draft	Initial version		

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc..

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

Preface gives information on the intended audience. It also describes the overall structure of the User Manual.

The subsequent chapters describes following details

- Objective and Scope
- Steps for JMS Configuration

1.5 Related Information Sources

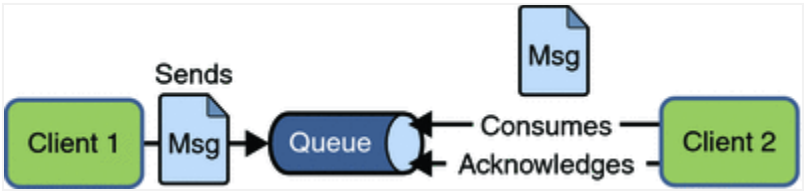
For more information on Oracle Banking APIs Release 18.2.0.0.0, refer to the following documents:

- Oracle Banking APIs Installation Manuals

2. Objective and Scope

2.1 Background

JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable communication. Messaging enables distributed communication that is loosely coupled. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only which message format and which destination to use.JMS configuration is required to send message (request) to external system and receive processed message (response) from external system.



2.2 Objective and Scope

Define a common set of messaging concepts and facilities. The scope of this document is to provide steps to configure foreign server for connecting external system using JNDI provider and configure JMS queue to receive data from external system. Foreign server is used to send message to external system with help of JNDI Initial, JNDI connection url, JNDI connection factory and JNDI destination. To configure JMS receiver queue in web logic we have to create JMS server and JMS module. Where JMS module include creation of JMS connection factory, JMS queue and SubDeployment.

Home

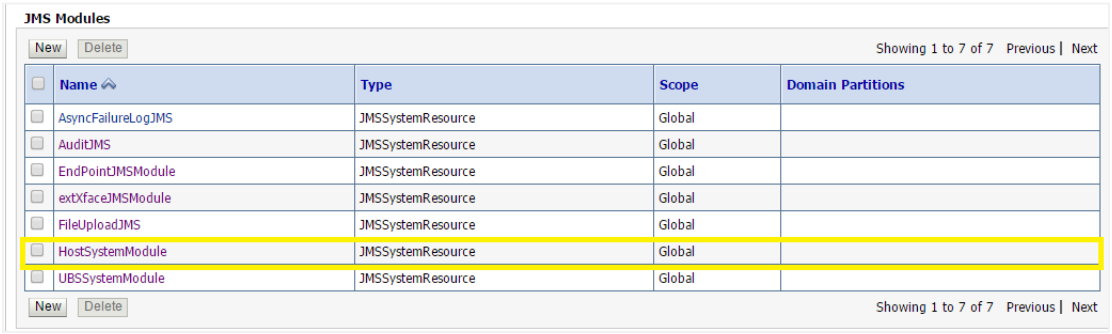
3. JMS Step 1: Create foreign server in a weblogic server

3.1 Introduction and Definitions

A Foreign Server represents a JNDI provider that is outside WebLogic server. It contains information that allows a local WebLogic Server instance to reach a remote JNDI provider, thereby allowing for a number of foreign connection factory and destination objects to be defined on one JNDI directory.

3.1.1 Create a JMS Module

- Services > Messaging > JMS Modules
- Select New
- Name: HostSystemModule
- Leave the other options empty
- Targets: obdx_server
- Press Next
- Leave "Would you like to add resources to this JMS system module" unchecked and press Finish .

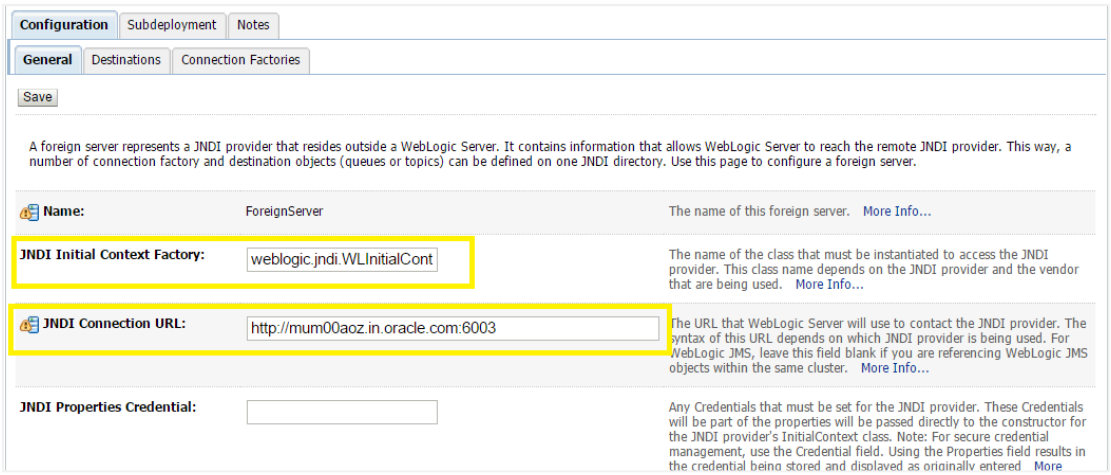


3.1.2 Create a foreign Server

- Services > Messaging > JMS Modules
- Select HostSystemModule and press New
- Select Foreign Server and Next
- Name: ForeignServer (Once you create a foreign server, you cannot rename it. Instead, you must delete it and create another one that uses the new name) and Click Next to proceed to the targeting page or click **Finish** to create the foreign server.

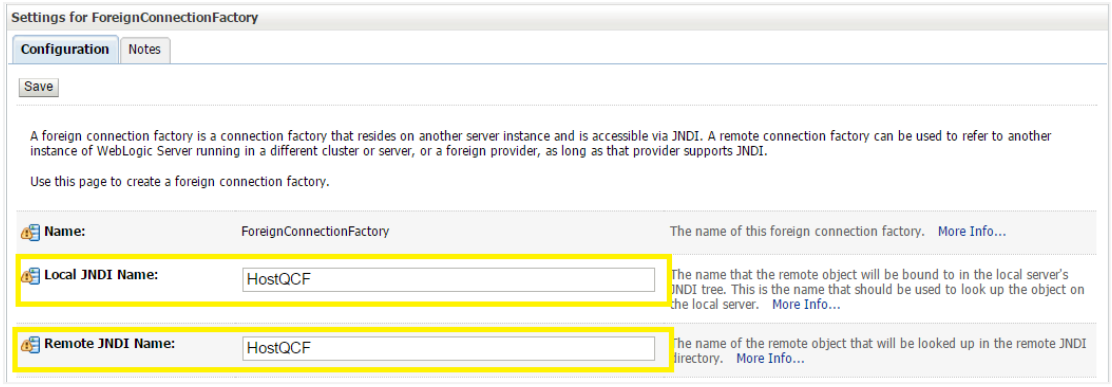
3.1.3 To configure additional properties for the new foreign server

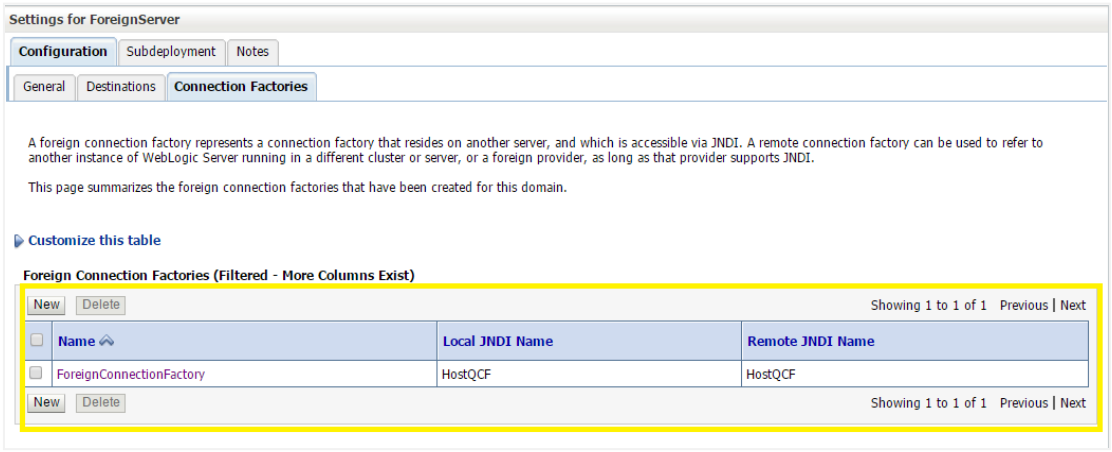
- Services > Messaging > JMS Modules
- Select HostSystemModule
- Click on ForeignServer
- On the Configuration> General tab
- Enter Following details.
  - JNDI Initial: enter the name of the class that must be instantiated to access the JNDI provider. For example (weblogic.jndi.WLInitialContextFactory)
  - JNDI Connection URL: enter the URL that WebLogic Server uses to contact the JNDI provider. (http://IP:port)
- Click **Save**.



3.1.4 Create foreign connection factories

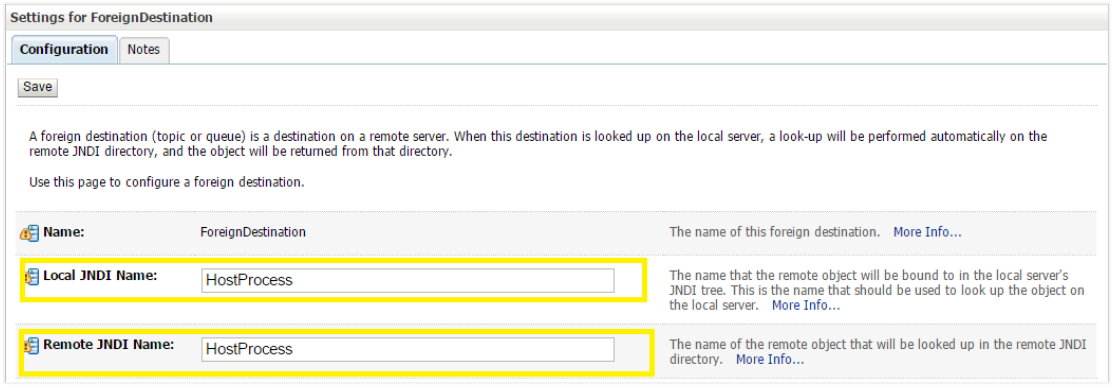
- Services > Messaging > JMS Modules
- Select HostSystemModule
- Click on ForeignServer
- On the Configuration**> Connection** Factories tab press **New**
- Enter Following details
  - Name: enter a name for the foreign connection factory.
  - Local JNDI Name: specify the name that the remote object will be bound to in the local server's JNDI tree and is used to look up the object on the local server.
  - Remote JNDI Name: specify the name of the remote object that will be looked up in the remote JNDI directory.
- Click **Ok**.

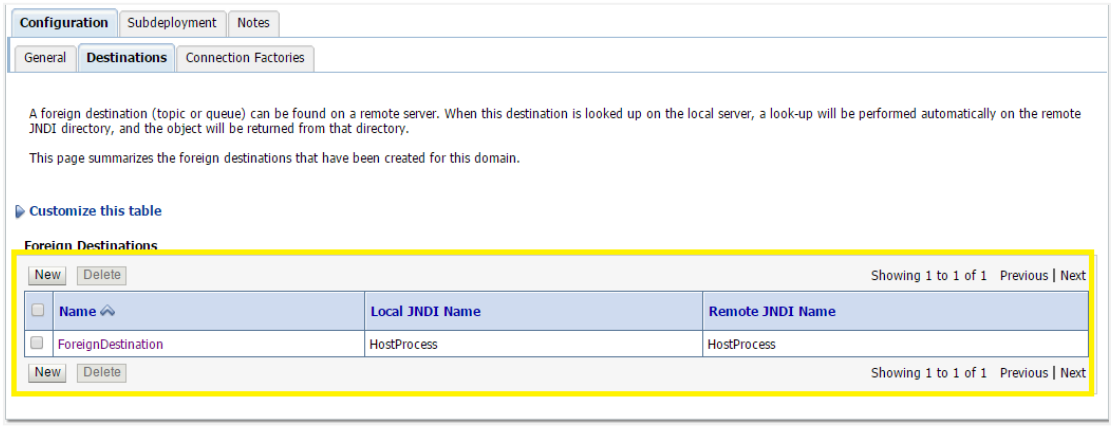




3.1.5 Create foreign destinations

- Services > Messaging > JMS Modules
- Select HostSystemModule
- Click on ForeignServer
- On the Configuration>Destination tab press New
- Enter Following details
    - Name: enter a name for the foreign destination.
    - Local JNDI Name: specify the name that the remote object will be bound to in the local server's JNDI tree and is used to look up the object on the local server.
    - Remote JNDI Name: specify the name of the remote object that will be looked up in the remote JNDI directory.
- Click Ok.


Settings for ForeignDestination
Configuration
Notes
Save
A foreign destination (topic or queue) is a destination on a remote server. When this destination is looked up on the local server, a look-up will be performed automatically on the remote JNDI directory, and the object will be returned from that directory.
Use this page to configure a foreign destination.
Name:
ForeignDestination
The name of this foreign destination. More Info...
Local JNDI Name:
HostProcess
The name that the remote object will be bound to in the local server's JNDI tree. This is the name that should be used to look up the object on the local server. More Info...
Remote JNDI Name:
HostProcess
The name of the remote object that will be looked up in the remote JNDI directory. More Info...


Configuration
Subdeployment
Notes
General
Destinations
Connection Factories
A foreign destination (topic or queue) can be found on a remote server. When this destination is looked up on the local server, a look-up will be performed automatically on the remote JNDI directory, and the object will be returned from that directory.
This page summarizes the foreign destinations that have been created for this domain.
Customize this table
Foreign Destinations
New
Delete
Showing 1 to 1 of 1 Previous | Next
Name
Local JNDI Name
Remote JNDI Name
ForeignDestination
HostProcess
HostProcess
New
Delete
Showing 1 to 1 of 1 Previous | Next

Home

4. JMS Step 2 - How to Create a Simple JMS Queue in Weblogic Server

4.1 Introduction and Definitions

A JMS queue in Weblogic Server is associated with a number of additional resources:

JMS Server

A JMS server acts as a management container for resources within JMS modules. Some of its responsibilities include the maintenance of persistence and state of messages and subscribers. A JMS server is required in order to create a JMS module.

JMS Module

A JMS module is a definition which contains JMS resources such as queues and topics. A JMS module is required in order to create a JMS queue.

Subdeployment

JMS modules are targeted to one or more WLS instances or a cluster. Resources within a JMS module, such as queues and topics are also targeted to a JMS server or WLS server instances. A subdeployment is a grouping of targets. It is also known as advanced targeting.

Connection Factory

A connection factory is a resource that enables JMS clients to create connections to JMS destinations.

JMS Queue

A JMS queue (as opposed to a JMS topic) is a point-to-point destination type. A message is written to a specific queue or received from a specific queue.

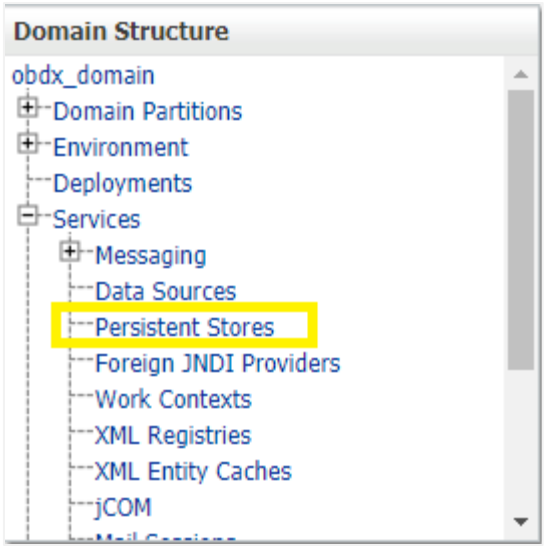
The objects used in this example are:

Object Name	**Type**	**JNDI Name**
ExtXfaceJMSServer	JMS Server	
extXfaceJMSModule	JMS Module	
extXfaceSubdeployment	Subdeployment	
ReceiverQCF	Connection Factory	
ReceiverQueue	JMS Queue	

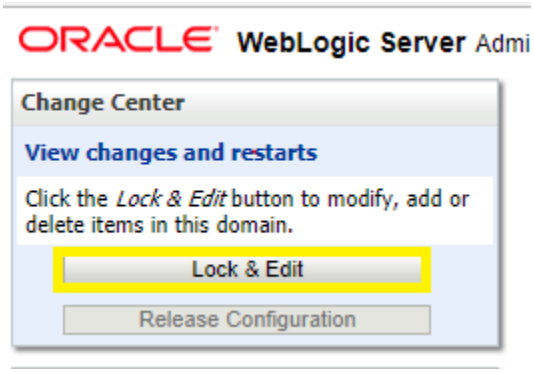
1. Configuration Steps-The following steps are done in the WebLogic Server Console, beginning with the left-hand navigation menu.

Create Persistent store-

- Here you have to Create a new persistent store (Once the persistent store is created that can be used for both sender and receiver serever. Hence there is no nedd to create a different persistent store for two different servers.) Hence Before creating a JMS server you need to create the Persistent store if its not already created. Follow the steps shown below for creating a persistent store.
- Select **Services > Persistent Stores.**

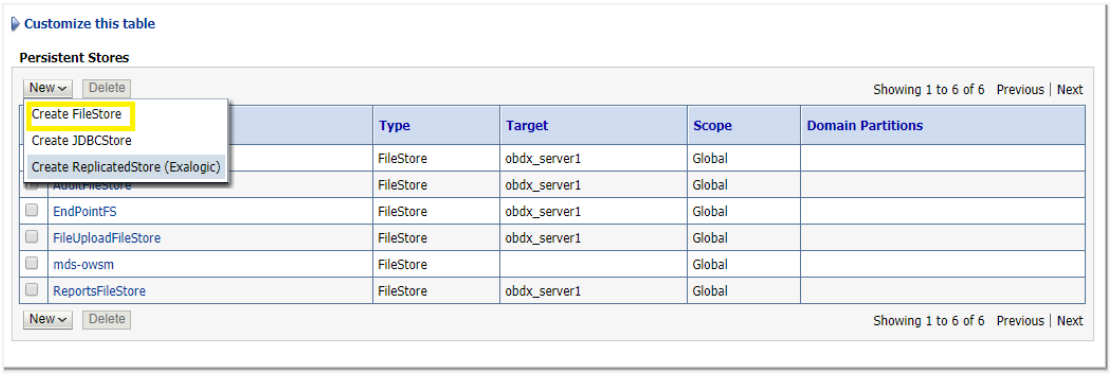


First Select Lock & Edit as shown-

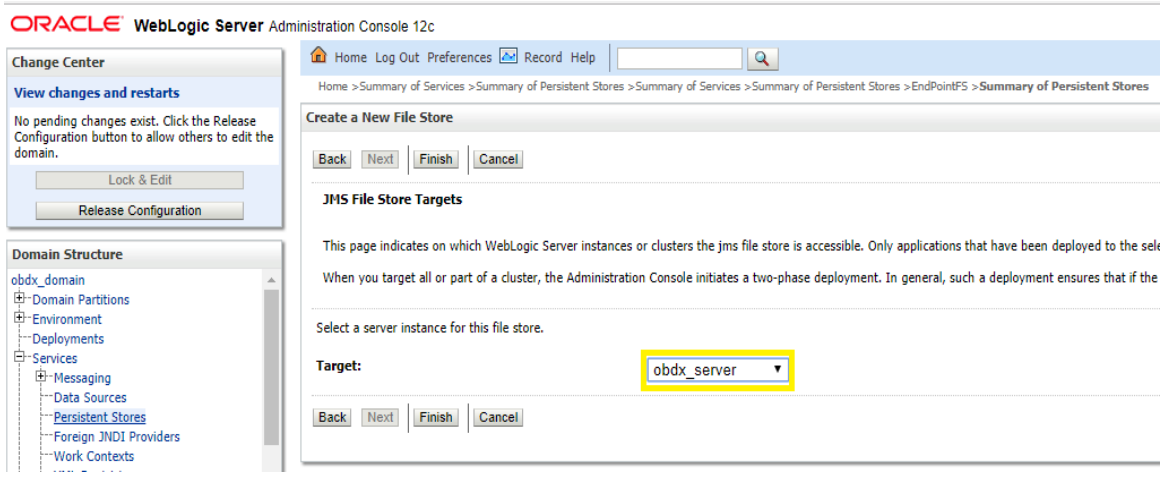


-

- Select new and the select create FileStore from the list as shown below-



- Give the name of the filestore. Example- **EndPointFS** and the Directory location, example **/scratch/obdx/wls. Directory location field is optional and the path given above is just an example , it may vary according to the server.**
- Click **Next.**
- Select the target server as shown in following snapshot-

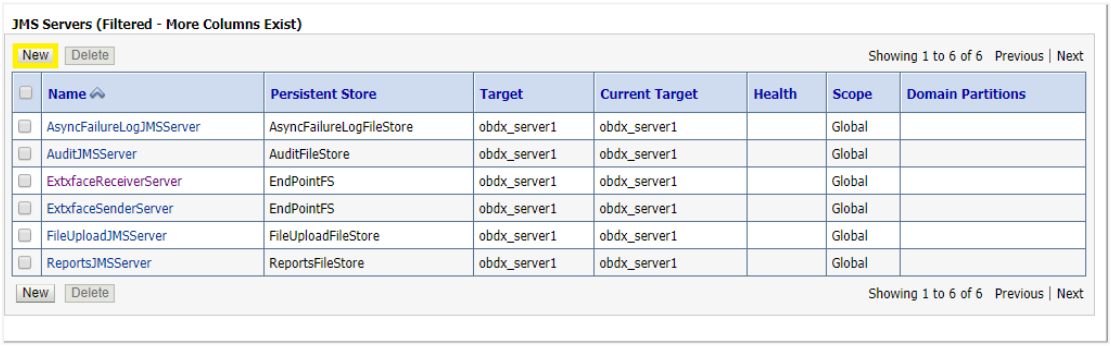


- Click **Finish.**

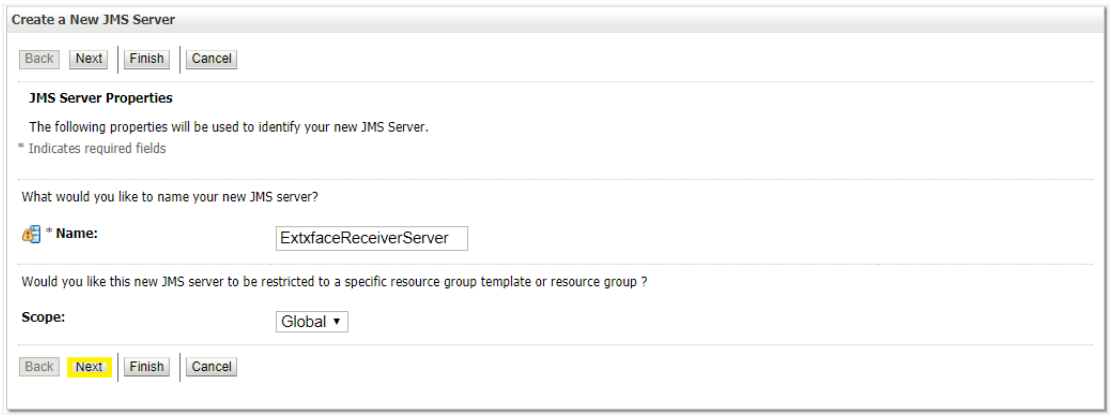
4.1.1 Create a JMS Server-

Services > Messaging > JMS Servers

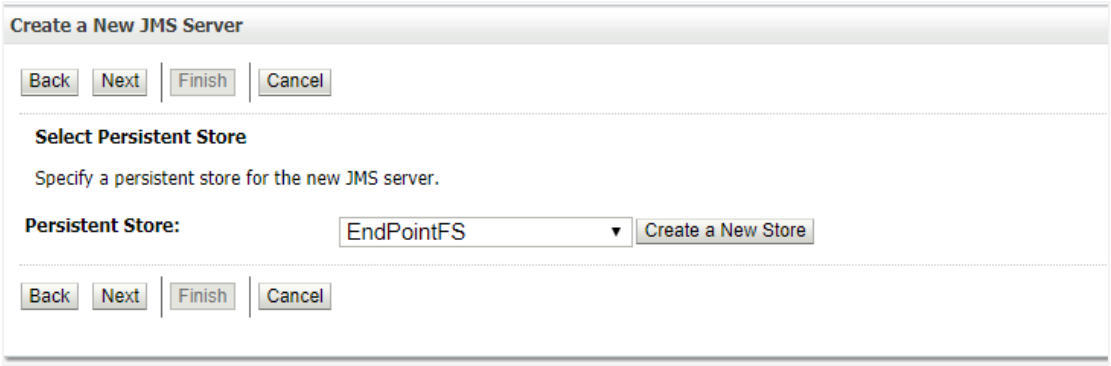
- Select **New.**



- Name: Give name as for example-**ExtxfaceReceiverServer .**
- After naming the server **Click next** as shown in following example screenshot.

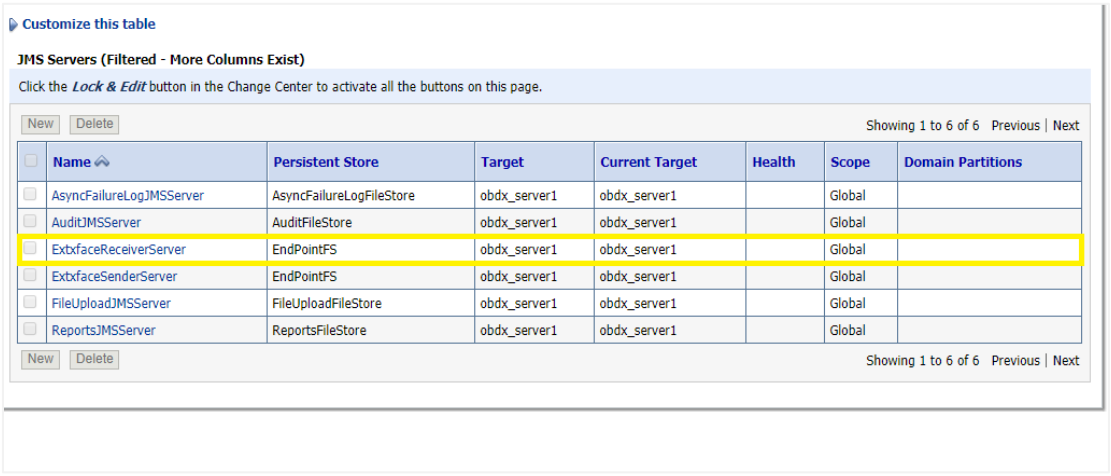


- **Persistent Store**: Select the name Persistent store from the dropdown list which was created in the previous step. Example-**EndPointFS.**
- Click **Next.**



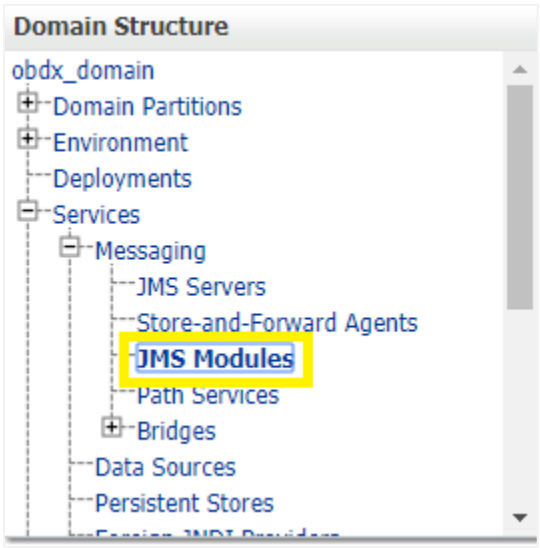
- **Target**: Target should Point to the **Weblogic server cluster** as in this case target is set to **obdx_server1** cluster. (Or any other available cluster).
- Click **Finish.**

The JMS server should now be visible in the list.

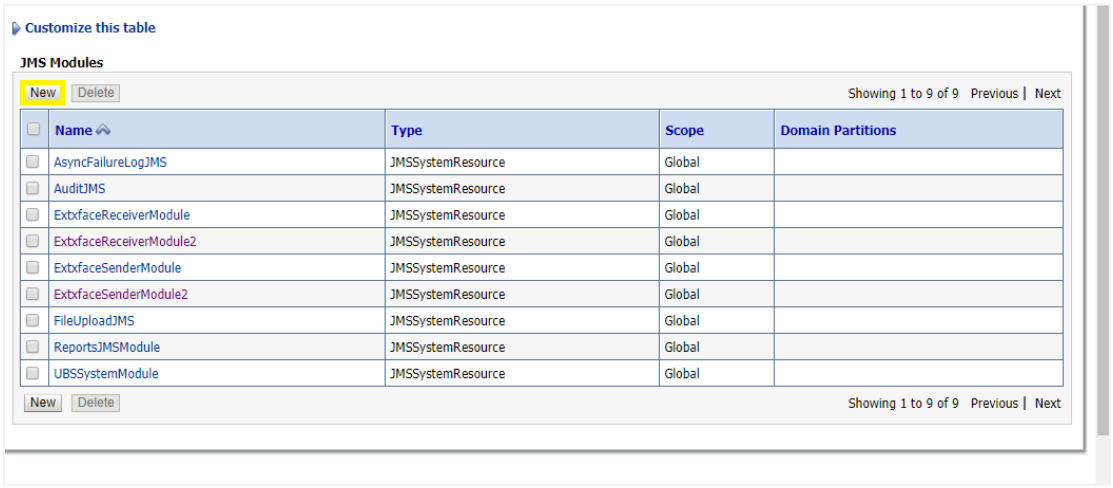


4.1.2 Create a JMS Module

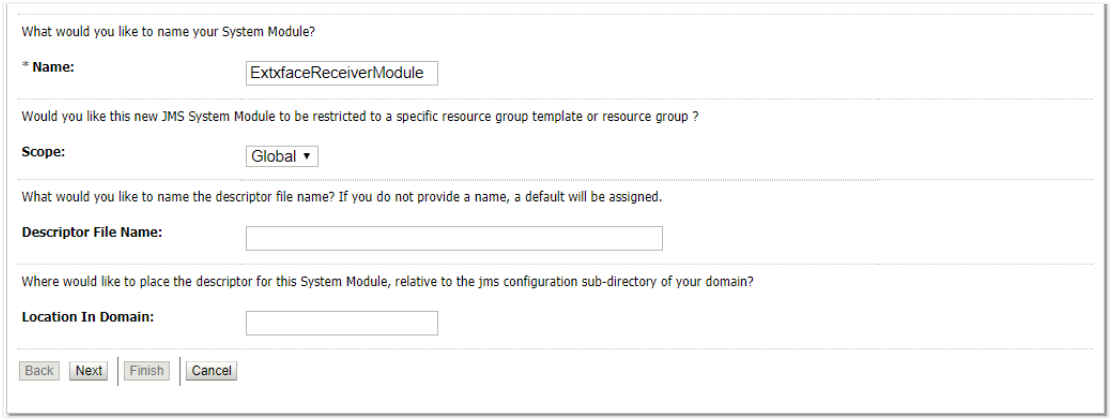
- Services > Messaging > JMS Modules.



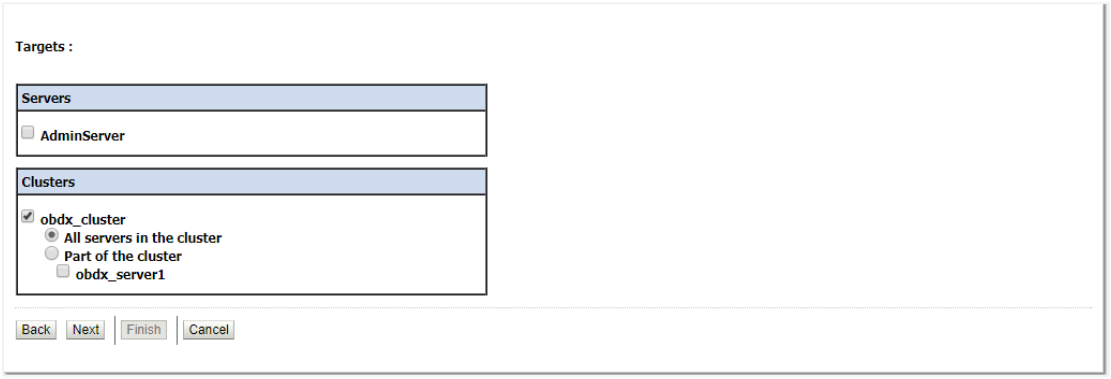
- Select **New.**



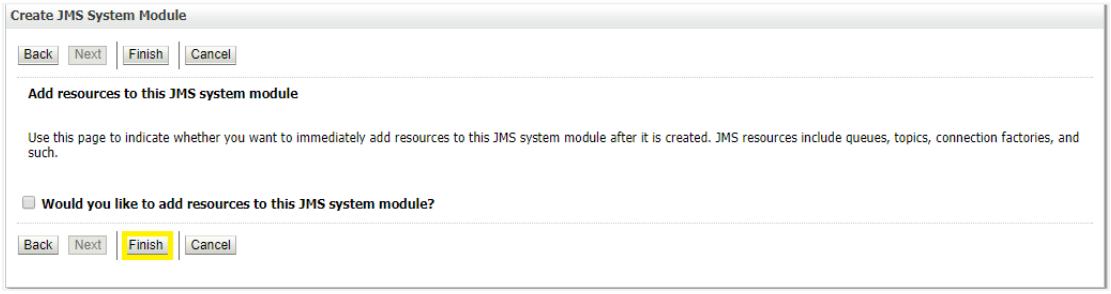
- Name: Provide name for JMS Module.
- Leave **the other options empty.**
- Click **Next.**

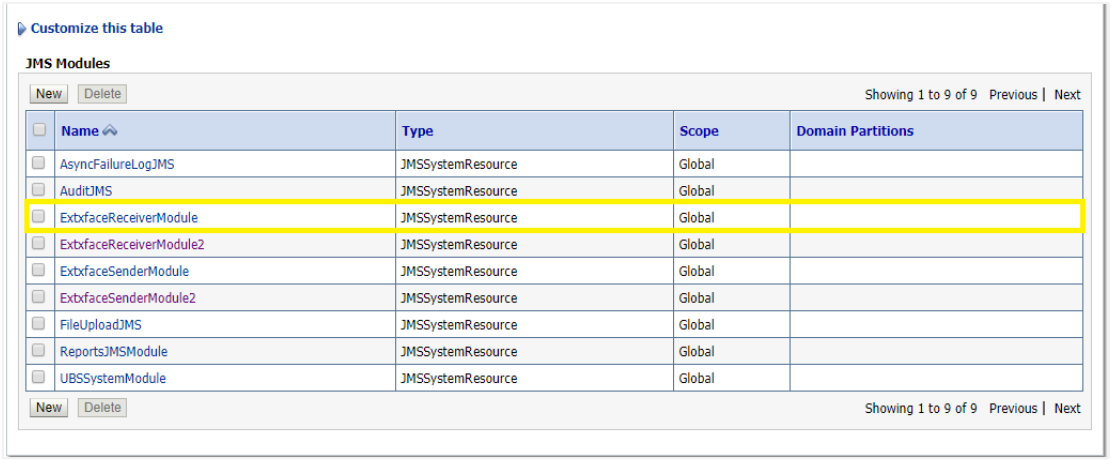


- Targets: **Obdx_Cluster** (or choose any other clusters available).
- Press **Next.**



Leave "**Would you like to add resources to this JMS system module**" unchecked and press **Finish** .

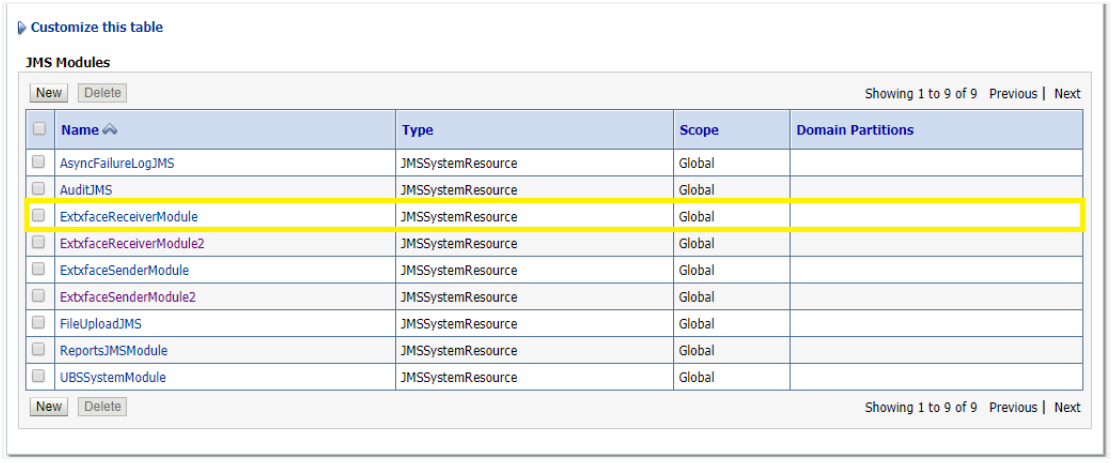




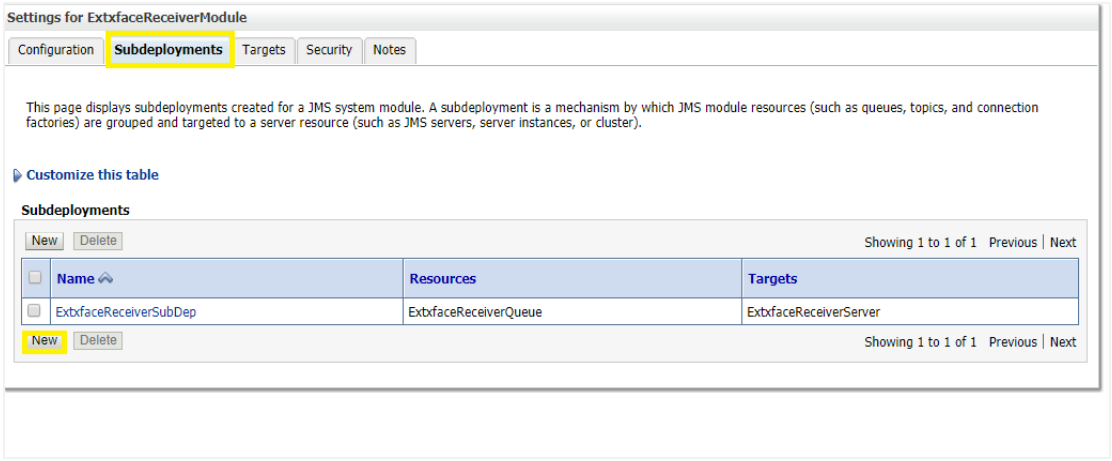
4.1.3 Create a SubDeployment

A subdeployment is not necessary for the JMS queue to work, but it allows you to easily target subcomponents of the JMS module to a single target or group of targets. We will use the subdeployment in this example to target the following connection factory and JMS queue to the JMS server we created earlier.

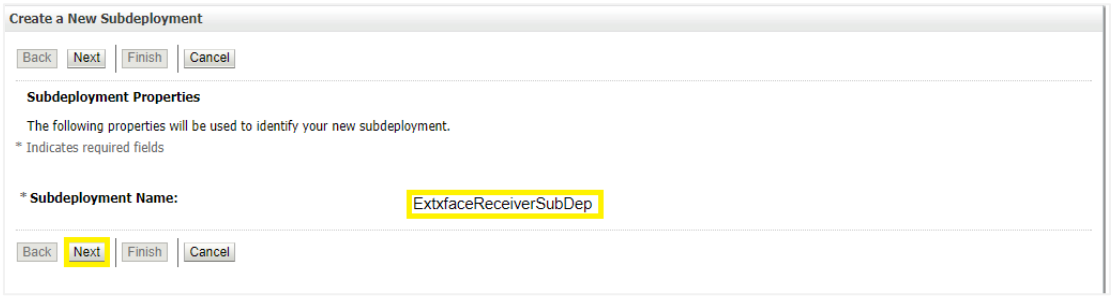
- Services > Messaging > JMS Modules.
- Select **ExtxfaceReceiverModule.**



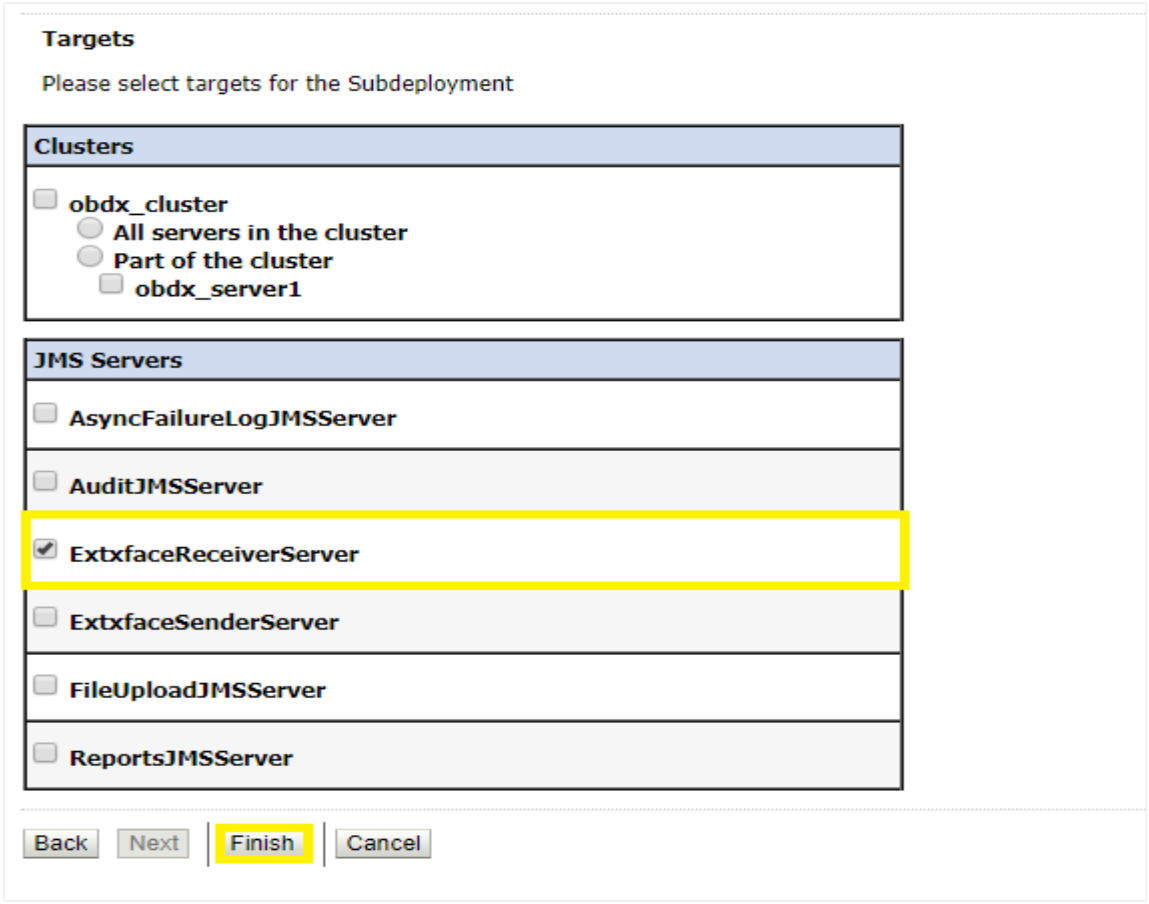
- Select the **Subdeployments** tab and click **New.**



- Subdeployment Name: give subdeployment name. example- **ExtxfaceReceiverSubDep**
- Press **Next.**

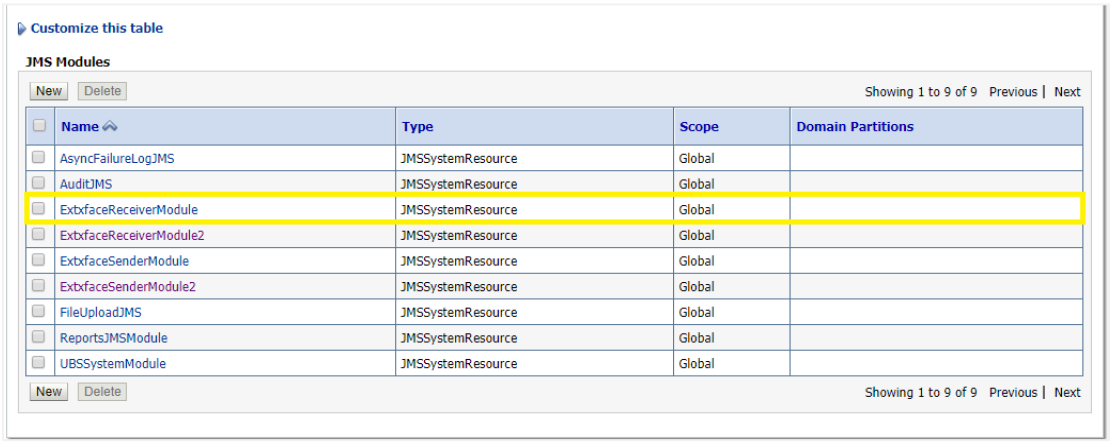


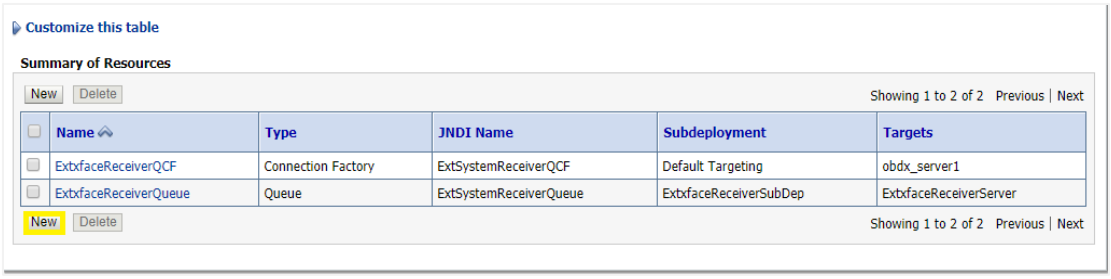
- Here you can select the target(s) for the subdeployment. You can choose either Servers (i.e. WebLogic managed servers, such as the **obdx_server**) or JMS Servers such as the JMS Server created earlier. As the purpose of our subdeployment in this example is to target a specific JMS server, we will choose the JMS Server option.
  Select the **ExtxfaceReceiverServer** created earlier.
- Press **Finish.**


Targets
Please select targets for the Subdeployment
Clusters
obdx_cluster
All servers in the cluster
Part of the cluster
obdx_server1
JMS Servers
AsyncFailureLogJMSServer
AuditJMSServer
ExtxfaceReceiverServer
ExtxfaceSenderServer
FileUploadJMSServer
ReportsJMSServer
Back
Next
Finish
Cancel

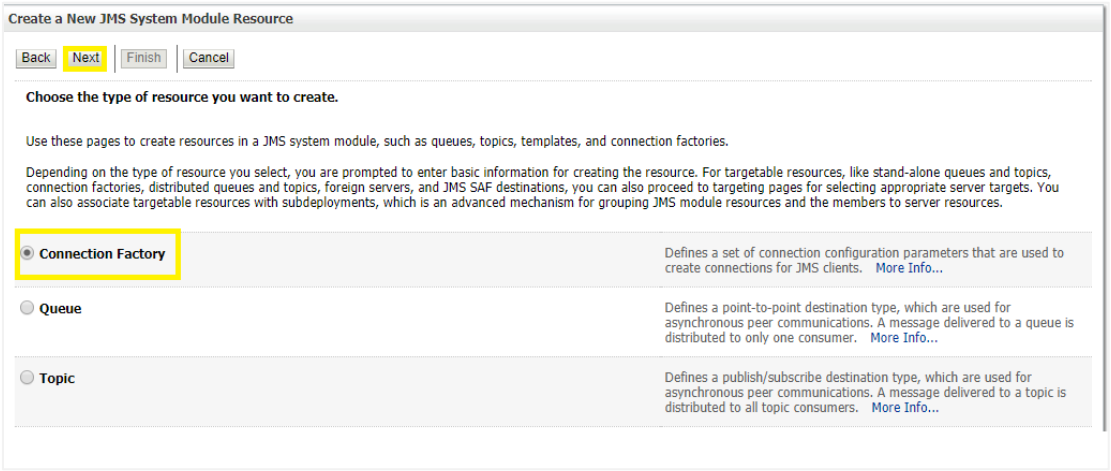
4.1.4 Create a Connection Factory

- Services > Messaging > JMS Modules
- Select **ExtxfaceReceiverModule** and press **New.**

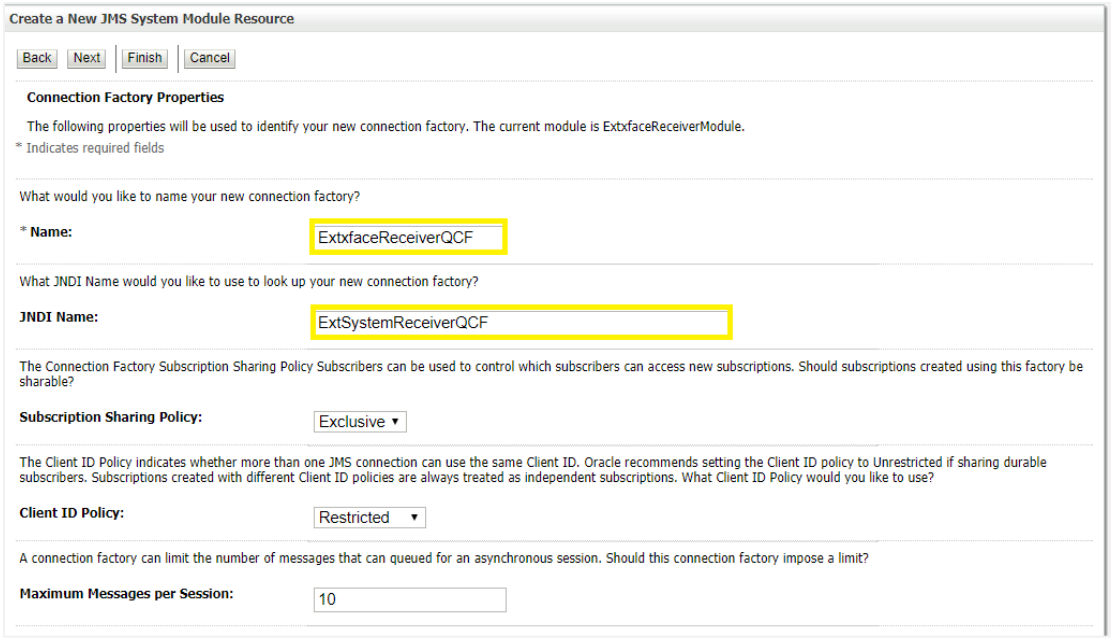




- Select **Connection Factory** and click **Next.**



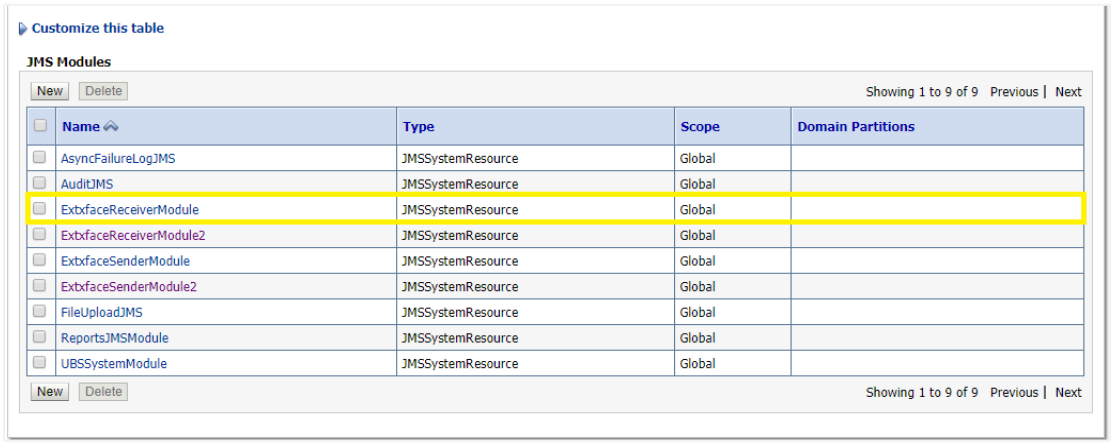
- Name: Give name of the connection factory example- **ExtxfaceReceiverQCF.**
  JNDI Name: **ExtSystemReceiverQCF.**
- Click **Next.**

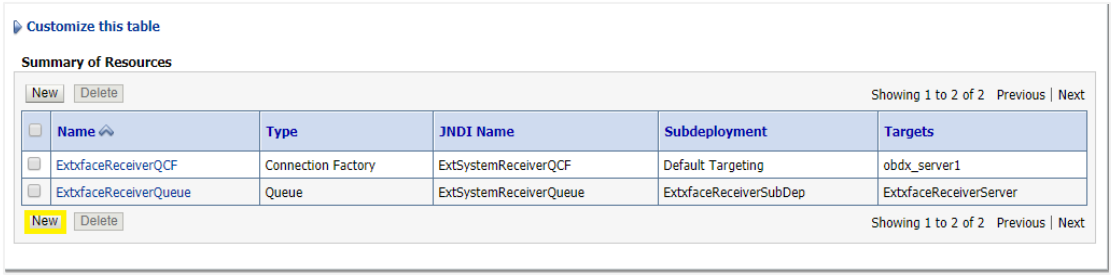


- Select Default Targeting Enabled and Press **Finish**
- The connection factory should be listed on the following page with **Default Targeting** as Subdeployment and WebLogic cluster as the target.

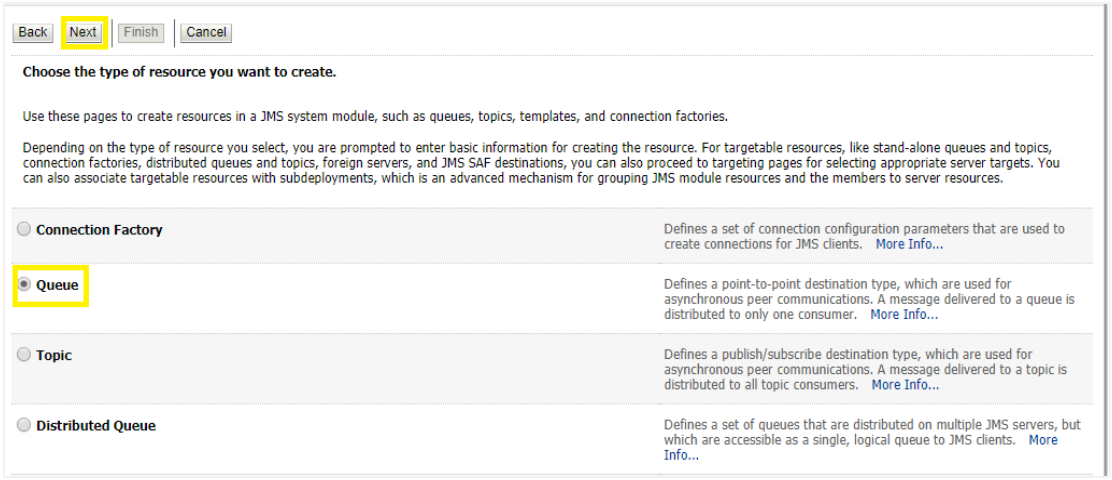
4.1.5 Create a JMS Queue

- Services > Messaging > JMS Modules
- Select **ExtxfaceReceiverModule** and Click **New.**

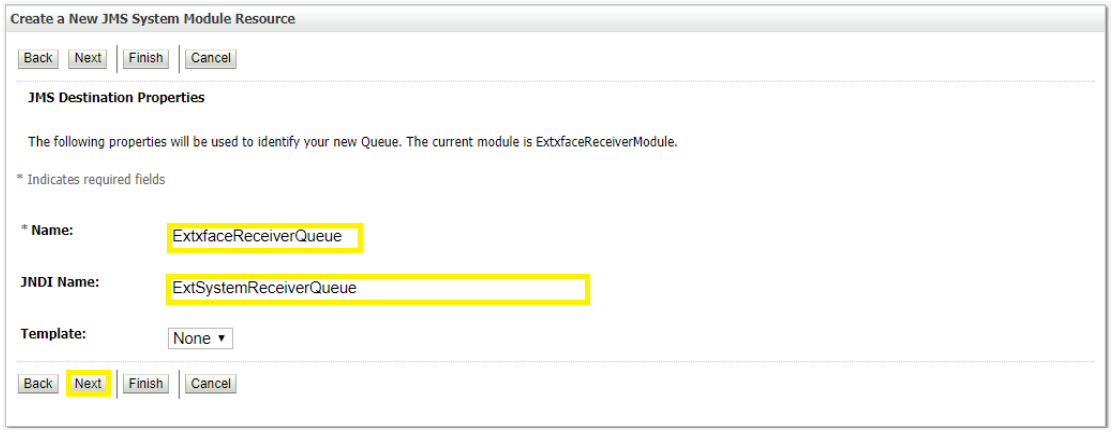




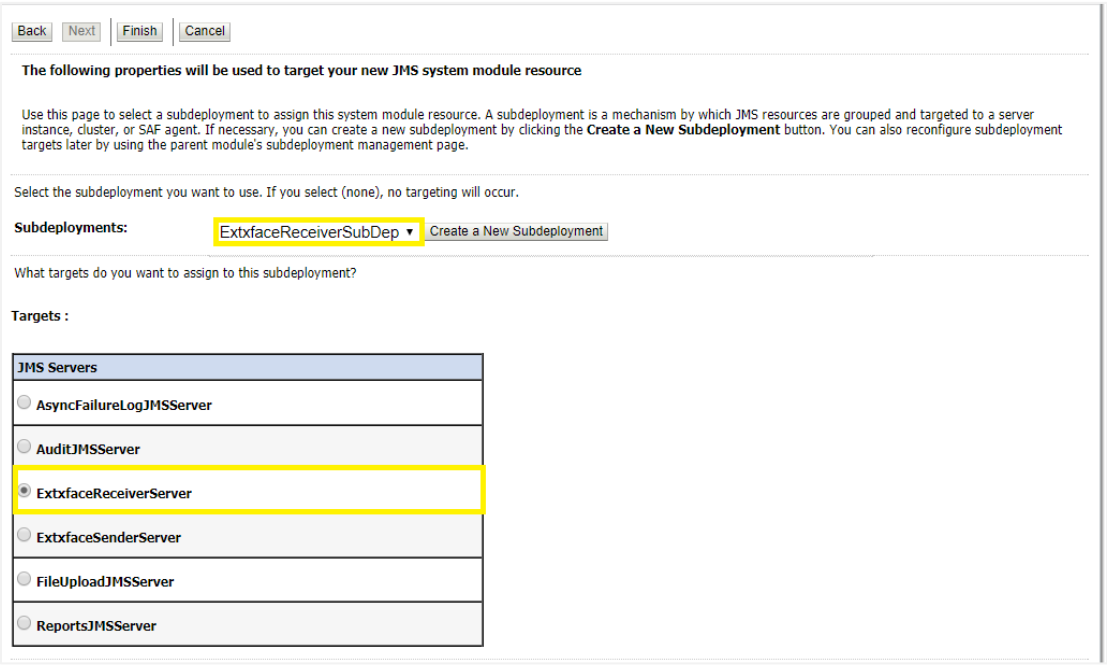
- Select **Queue** and Click **Next.**



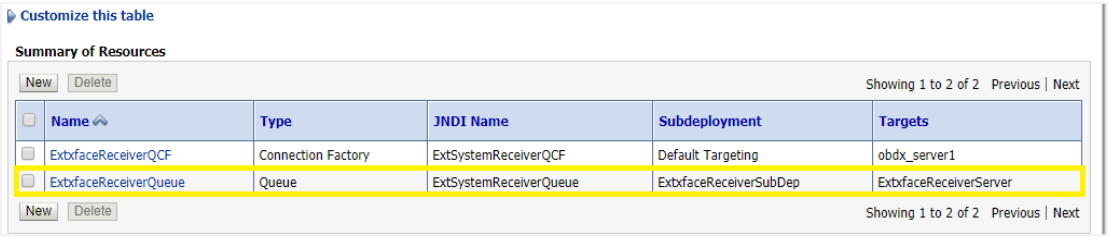
- **Name**: Provide name of the message queue. example- **ExtxfaceReceiverQueue.**
  JNDI Name: Provide JNDI name. example- **ExtSystemReceiverQueue.**
  Template: **None.**
- Press **Next.**



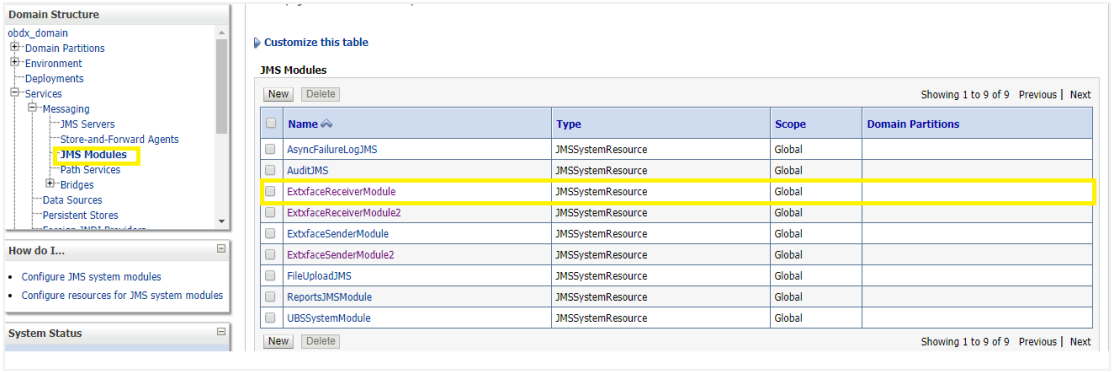
- **Subdeployments:** Give the name of the sub-deployment name in which Queue is supposed to be added**. Example-** ExtxfaceReceiverSubDep.
- Select the Target as **ExtxfaceReceiverServer**
  Click **Finish.**



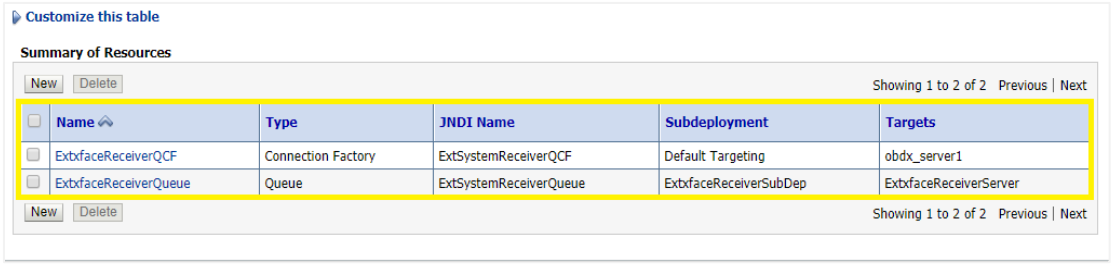
The **ReceiverQueue** should be listed on the following page with Sub-deployment as **ExtxfaceReceiverSubDep** and target as **ExtxfaceReceiverServer.**



Confirm the resources for the **ExtxfaceReceiverModule**. Using the Domain Structure tree, navigate to Services > Messaging > JMS Modules then select **ExtxfaceReceiverModule**



You should see the following resources-



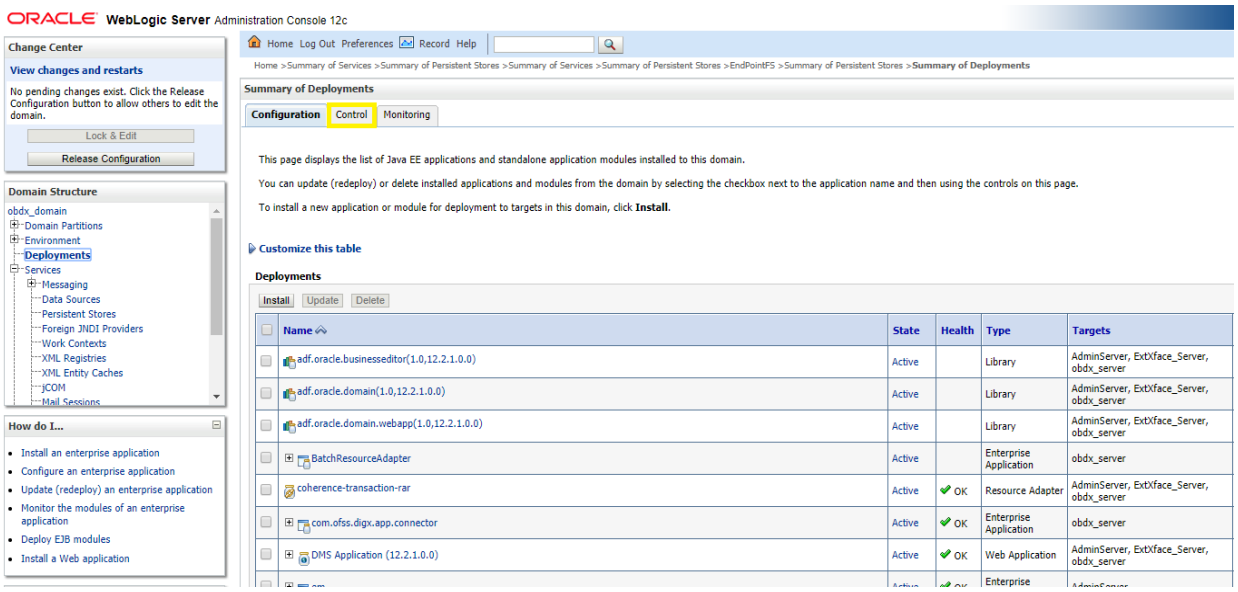
The JMS queue is now complete and can be accessed using the JNDI names

ExtSystemReceiverQCF And **ExtSystemReceiverQueue**..

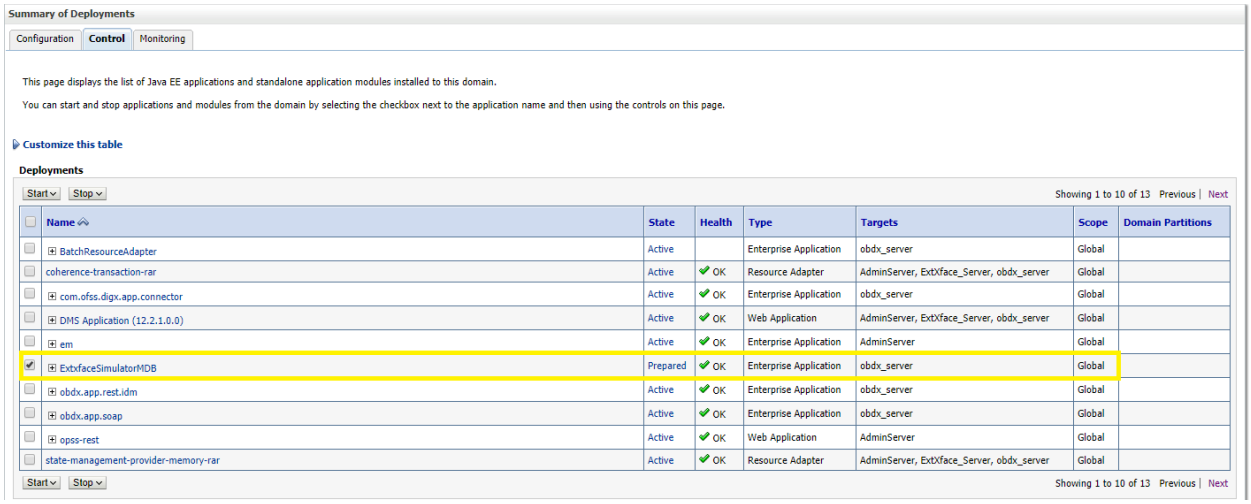
Note: Repeat the above process from the step 4.1 i.e Create File-Store to create the JMS Configuration for Sender module. Separate JMS Server , Module and Queues would get created for Sender.

In case of a multi-entity setup where Third-party entity is not a base entity after Creating the JMS configuration for both the Receiver and sender you have to manually deploy the ExtxfaceSimulatorMDB.ear on weblogic server present in the installebes. After deploying the ExtxfaceSimulatorMDB and restarting the server, check the **state** of the application by going in **Deployments** wizard on the weblogic server console. If it is not in “Active” state, it needs to be started manually, to do so, follow the steps mentioned below by keeping server in ‘Running’ state-

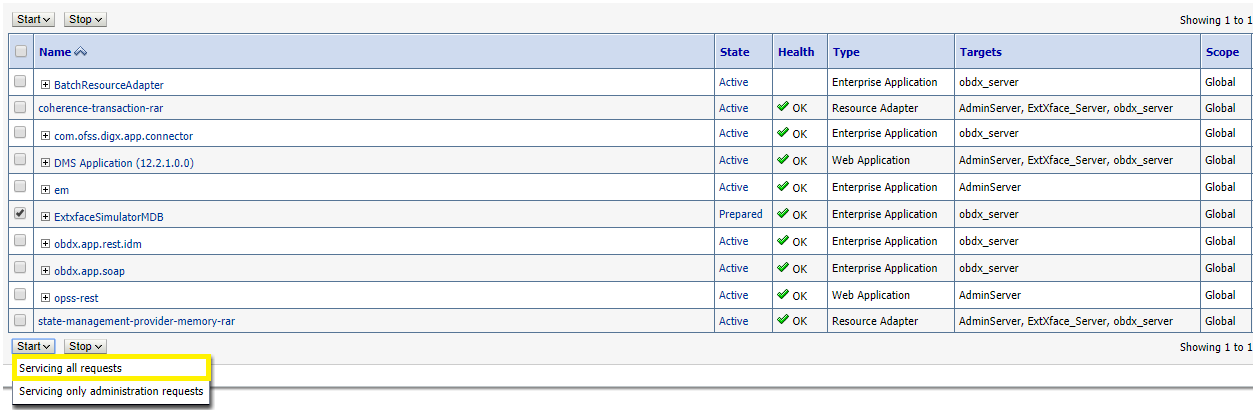
- Go into the **Control** tab.



- From the List of applications select the checkbox before **ExtxfaceSimulatorMDB ear**.



- Select the '**Start'** dropdown list and from that select option-"**Servicing all requests**".



Note: Whenever a new Entity is created within a setup(i.e multiple third party entities) the following mentioned steps should be followed in order to enable support for MultiEntity.

In order to enable the support for newly created Entity, Sender/Receiver Connection Factories and Queues are needed to be created within a new Sender/Receiver JMS Modules. These modules can be hosted on the same Sender/Receiver JMS server created as per the steps defined in section 4.1.1(Create JMS Server) for the earlier Entity.

- Create a new JMS Module by repeating steps given in section 4.1.2(Creating JMS Module) , on the same JMS server with new names as follows-

	JMS Module Name
Sender JMS Module	ExtxfaceSenderModule2
Receiver JMS Module	ExtxfaceReceiverModule2

- Create a new SubDeployment within both Sender/Receive module created with above step by repeating the procedure given in section 4.1.3 (Create JMS Subdeployment) with the new name as follows-

	JMS Module Name	**SubDeployment Name**
Sender JMS Module	ExtxfaceSenderModule2	ExtxfaceReceiverSubDep2
Receiver JMS Module	ExtxfaceReceiverModule2	ExtxfaceSenderSubDep2

- Create Sender/Receiver connection factories within newly created module by following the steps defined in the section 4.1.4(Creating Connection Factories), with different names as follows-

	Connection Factory Name	**Connection Factory JNDI Name**
Sender Connection Factory	ExtxfaceSenderQCF2	ExtSystemSenderQCF2
Receiver Connection Factory	ExtxfaceReceiverQCF2	ExtSystemReceiverQCF2

- Create Sender/Receiver JMS queues within newly created JMS module by repeating the steps given in section 4.1.5 (Creating JMS Queues), with the new names to the sender/receiver queues as follows-

	JMS Queue Name	**JMS Queue JNDI Name**
Sender JMS Queue	ExtxfaceSenderQueue2	ExtSystemSenderQueue2

	JMS Queue Name	JMS Queue JNDI Name
Receiver JMS Queue	ExtxfaceReceiverQueue2	ExtSystemReceiverQueue2

After creating the new JMS sender/receiver modules, connection factories and queues by following the above defined steps. Further Redeploy the ExtxfaceSimulatorMDB.ear with the following changes -

- Add new <message-driven> tag in **ejb-jar.xml** (**Path**-ExtxfaceSimulatorMDB.ear\com.ofss.digx.extxface.mdb.jar\META-INF\ **ejb-jar.xml**) as shown below –

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved. -->
- <ejb-jar version="3.0" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
  xmlns:ejb="http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <display-name>com.ofss.digx.extxface.mdb</display-name>
   - <enterprise-beans>
      - <message-driven>
          <display-name>ExtxfaceSimulatorMDB</display-name>
          <ejb-name>ExtxfaceSimulatorMDB</ejb-name>
          <ejb-class>com.ofss.digx.extxface.mdb.ExtxfaceSimulatorMDB</ejb-class>
          <transaction-type>Bean</transaction-type>
          <message-destination-type>javax.jms.Queue</message-destination-type>
        </message-driven>
      - <message-driven>
          <display-name>ExtxfaceSimulatorMDB2</display-name>
          <ejb-name>ExtxfaceSimulatorMDB2</ejb-name>
          <ejb-class>com.ofss.digx.extxface.mdb.ExtxfaceSimulatorMDB</ejb-class>
          <transaction-type>Bean</transaction-type>
          <message-destination-type>javax.jms.Queue</message-destination-type>
        </message-driven>
    </enterprise-beans>
</ejb-jar>
```

Fig.1 ExtxfaceSimulatorMDB.ear\com.ofss.digx.extxface.mdb.jar\META-INF\ejb-jar.xml

```
<message-driven>
<display-name>ExtxfaceSimulatorMDB2</display-name>
<ejb-name>ExtxfaceSimulatorMDB2</ejb-name>
<ejb-class>com.ofss.digx.extxface.mdb.ExtxfaceSimulatorMDB</ejb-class>
<transaction-type>Bean</transaction-type>
<message-destination-type>javax.jms.Queue</message-destination-type>
</message-driven>
```

Note: As Shown in above example the value of the <ejb-class> sub-tag in <message-driven> tag should be same for all the Entities.

- Add new <weblogic-enterprise-bean> configuration tag in **weblogic-ejb-jar (Path-** ExtxfaceSimulatorMDB.ear\com.ofss.digx.extxface.mdb.jar\META-INF\ **weblogic-ejb-jar)** as shown below**-**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved. -->
<weblogic-ejb-jar xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.1/weblogic-ejb-jar.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar">
  - <weblogic-enterprise-bean>
      <ejb-name>ExtxfaceSimulatorMDB</ejb-name>
      <dispatch-policy>ExtxfaceWorkManager</dispatch-policy>
    - <message-driven-descriptor>
        - <pool>
            <initial-beans-in-free-pool>10</initial-beans-in-free-pool>
            <max-beans-in-free-pool>100</max-beans-in-free-pool>
          </pool>
          <destination-jndi-name>ExtSystemSenderQueue</destination-jndi-name>
          <connection-factory-jndi-name>ExtSystemSenderQCF</connection-factory-jndi-name>
          <jms-polling-interval-seconds>1</jms-polling-interval-seconds>
      </message-driven-descriptor>
      <jndi-name>ExtSystemSenderQueue</jndi-name>
    - <transaction-descriptor>
        <trans-timeout-seconds>60</trans-timeout-seconds>
      </transaction-descriptor>
  </weblogic-enterprise-bean>
  - <weblogic-enterprise-bean>
      <ejb-name>ExtxfaceSimulatorMDB2</ejb-name>
      <dispatch-policy>ExtxfaceWorkManager</dispatch-policy>
    - <message-driven-descriptor>
        - <pool>
            <initial-beans-in-free-pool>10</initial-beans-in-free-pool>
            <max-beans-in-free-pool>100</max-beans-in-free-pool>
          </pool>
          <destination-jndi-name>ExtSystemSenderQueue2</destination-jndi-name>
          <connection-factory-jndi-name>ExtSystemSenderQCF2</connection-factory-jndi-name>
          <jms-polling-interval-seconds>1</jms-polling-interval-seconds>
      </message-driven-descriptor>
      <jndi-name>ExtSystemSenderQueue2</jndi-name>
    - <transaction-descriptor>
        <trans-timeout-seconds>60</trans-timeout-seconds>
      </transaction-descriptor>
  </weblogic-enterprise-bean>
  - <run-as-role-assignment>
      <role-name>LookupRole</role-name>
      <run-as-principal-name>weblogic</run-as-principal-name>
    </run-as-role-assignment>
  - <work-manager>
      <name>ExtxfaceWorkManager</name>
```

Fig.2 ExtxfaceSimulatorMDB.ear\com.ofss.digx.extxface.mdb.jar\META-INF\weblogic-ejb-jar.xml

<weblogic-enterprise-bean>

<ejb-name>ExtxfaceSimulatorMDB2</ejb-name>

<dispatch-policy>ExtxfaceWorkManager</dispatch-policy>

```
        <message-driven-descriptor>
                <pool>
                        <initial-beans-in-free-pool>10</initial-beans-in-free-pool>
                        <max-beans-in-free-pool>100</max-beans-in-free-pool>
                </pool>
                <destination-jndi-name>ExtSystemSenderQueue2</destination-jndi-name>
                <connection-factory-jndi-name>ExtSystemSenderQCF2</connection-factory-jndi-
name>
                <jms-polling-interval-seconds>1</jms-polling-interval-seconds>
        </message-driven-descriptor>
        <jndi-name>ExtSystemSenderQueue2</jndi-name>
        <transaction-descriptor>
                <trans-timeout-seconds>60</trans-timeout-seconds>
        </transaction-descriptor>
</weblogic-enterprise-bean>
```

Note: <destination-jndi-name> i.e. JNDI name of the JMS sender queue should be same as given while creating the queue. In above example it is- ExtSystemSenderQueue2.

<connection-facory-jndi-name> i.e. connection factory JNDI name should be same as given while creating the new connection factory. In the above example it is- ExtSystemSenderQCF2.

<dispatch-policy> value should be same for all the Entities. i.e., ExtxfaceWorkManager

---

After Redeploying the ExtxfaceSimulatorMDB and restarting the server, check the **state** of the application by going in **Deployments** wizard on the weblogic server console. If it is not in “Active” state, it needs to be started manually, to do so, follow the steps mentioned below by keeping server in ‘Running’ state-

- Go into the **Control** tab.
- From the List of applications select the checkbox before **ExtxfaceSimulatorMDB ear**.
- Select the ‘**Start’** dropdown list and from that select option-“**Servicing all requests**”.

Home