Characteristics Concerning Particular JMS Providers

Find here some characteristics concerning the following JMS providers:

- Active MQ
- Oracle WebLogic
- GlassFish MQ 4

The difference between LDAP via JNDI and via a file binding is explained by reference to GlassFish MQ.

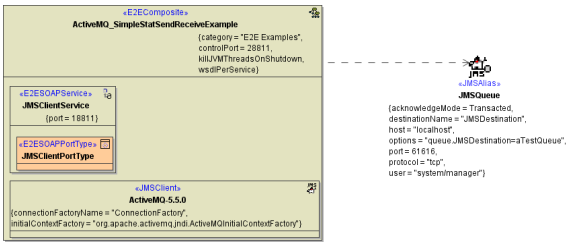
Example Files (Builder project Add-ons/JMS):

<your example path>\Add-ons\JMS\uml\simpleStatSendReceive.xml
<your example path>\Add-ons\JMS\uml\simpleStatSendReceive_Depl_ActivMQ.xml
<your example path>\Add-ons\JMS\uml\simpleStatSendReceive_Depl_WebLogic.xml
<your example path>\Add-ons\JMS\uml\simpleStatSendReceive_Depl_GlassFish.xml
<your example path>\Add-ons\JMS\uml\simpleStatSendReceive_Depl_GlassFish_FileJNDI.xml

On this Page:

Active MQ

The following picture shows a component diagram of an xUML service using Active MQ as JMS provider.



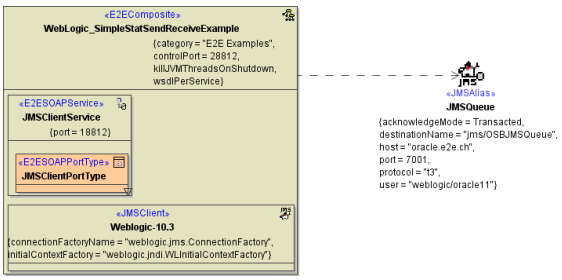
Setting user and password is mandatory. This applies to all JMS providers.

Setting the queue option on the JMS alias, the connection to Active MQ works out of the box with an default Active MQ installation. Principally, this contradicts the idea of using JNDI as an abstraction layer between the service and the JMS provider. To avoid this conflict, additionally an LDAP server can be installed and configured (also see GlassFish MQ).

The receipt of messages with Active MQ is only reliable using `millisecondsToWait >= 10` . This receive parameter can only be specified using dynamic JMS.

Oracle WebLogic

The following picture shows a component diagram of an XUML service using WebLogic as JMS provider.



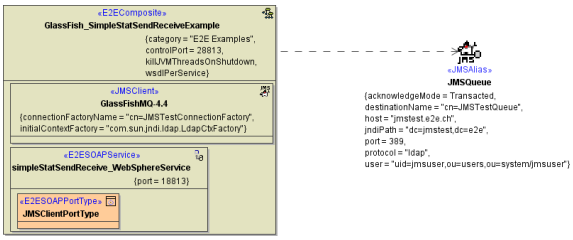
Setting user and password is mandatory. This applies to all JMS providers.

The tagged value **destinationName** has to contain the Oracle queue name.

GlassFish MQ 4

JNDI by taking the example of GlassFish

The following picture shows a component diagram of an xUML service using IBM WebSphereMQ 7 as JMS Client in connection with an LDAP JNDI Provider.



The LDAP directory server can act as a repository for connection details. Together with JNDI, this provides an abstraction layer between the service and the JMS provider, which means that connection details can be retrieved from the directory service without the client needing to know which provider is actually administrating the message queues.

The following information has to be specified to access a message queue that way:

- On client side: In the tagged value **connectionFactoryName** , define the common name (`cn`) of the key used to retrieve the connectionFactoryName from the LDAP service. In the tagged value **initialContextFactory** , the Java class used to access the LDAP service is specified. This value is set automatically by the Components Wizard.
- On the <<JMSAlias>> specify the following:
    - port and protocol of the LDAP service
    - user (`uid`) and organizational unit (`ou`) used to access the LDAP service
    - common name (`cn`) of the key used to retrieve the name of the queue from the LDAP service. The name is to be specified the way `cn=<name of the key>` .
    - user (`uid` and `ou`) and password of the LDAP service.

JNDI with file binding by taking the example of GlassFish

An LDAP server not being available, the directory service can be substituted by a file binding. In that case, all necessary information on how to access the message queue is stored in a file .bindings, which is to be created by use of the WebSphere MQ Explorer.

The following information has to be specified to access a message queue that way:

- On client side:
    - In the tagged value **connectionFactoryName** , define the common name (`cn`) of the key used to retrieve the connectionFactoryName as defined in the file binding.
    - In the tagged value **initialContextFactory** , the Java class used to access the file binding is specified. This value is set automatically by the Components Wizard.
- On the <<JMSAlias>>:
    - Specify the protocol **(file)** of the JNDI naming service.
    - In the tagged value **path** specify the path to the directory containing the .bindings file.
    - In the tagged value **destinationName** define the common name of the key used to retrieve the name of the queue from the .bindings file.