JMS Frequently Asked Questions

Copyright

Icons in Body Text

Icon	Meaning
⚠	Caution
✿	Example
💡	Note
⬆	Recommendation
◇	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
`Example text`	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

SAP JMS 1.1 Provider

1. Is Java Message Service (JMS) a product?

No, the Java Message Service (JMS) is not a product itself, but a set of predefined interfaces provided in the `javax.jms` package. The applications use this API to send and receive messages to and from a destination located on the JMS Provider. Java programs that send and/or receive messages using JMS are called JMS clients.

Java applications can use JMS to send and receive messages using an enterprise messaging server in a standardized, vendor-independent way. The actual messaging product that implements the JMS interfaces is the JMS Provider.

2. What are the features of SAP JMS Provider?

SAP's JMS Provider has the following major features:

- JMS 1.1 Compliance – one of the main new features in JMS 1.1 is the domain unification: support of client code that works simultaneously with either the point-to-point or publish / subscribe domains. Queues and topics now can be accessed through the same session and thus in the same transaction.
- JCA 1.5 Adapter – the JMS Resource Adapter is an Inbound Resource Adapter according to the JCA 1.5 specification. It allows plugability of the JMS Providers with J2EE containers in a standardized fashion. The message-driven beans (MDBs) use the JMS Provider using the resource adaptor instead of working with the provider directly. This is preferable since it hides from the MDBs the JMS specifics as a messaging service provider.
- Software High Availability.
- JTA Transactions.
- XA Transactions.
- RDBMS Persistence – message persistency implemented using a relational database.
- Message Pre-fetching – the feature to pre-fetch JMS messages from the JMS server to a synchronous JMS client. This can improve performance in certain scenarios.
- Message Paging / Swapping – the feature to swap whenever necessary parts of the messages to a persistent store to free up main memory. This may cause persisting non-persistent messages.
- Multiple Queue Receivers – the JMS specification does not specify the behavior in case several consumers are registered on one and the same queue. The SAP's JMS Provider supports multiple queue receivers and the behavior is configurable – round-robin delivery or exception on second receiver.
- Dead Messages Support – dead messages are messages that cannot be delivered to the client after a configurable number of attempts. Such messages are moved to a dedicated dead messages queue where an administrator can review them.
- JMS Resources Deployment – the JMS resources needed are defined in a JMS resources descriptor and are automatically created during deployment. Deployment of applications that use JMS resources triggers a unified JMS resources deployment procedure that is driven by the JMS Connector and the JMS Provider in a sequence.

- Automatic MDB Scaling – scaling of MDBs on a queue can be achieved through the round-robin behavior of multiple queue receivers.
- Automatic Configuration Tuning – self-tunable configuration, containing parameters – formulas, whose value is dynamically calculated. Minimizes the administration overhead.
- HermesJMS Integration – integration with the popular open-source administration and monitoring tool HermesJMS.
- XML Message Export / Import – the ability to export and/or import JMS messages of a particular destination in XML format.
- JMX / JSR77 Support – remote administration and monitoring of the JMS Provider using JMX and/or JSR77.
- Template Configuration – configuration providing predefined templates for all JMS resources. Minimizes administration overhead. A special template for each JMS configuration entity type allows all configurations to derive from e certain template, inheriting all values and customizing part of them, thus facilitating the JMS administration.
- Security Administration – the set of features to define, modify and enforce security restrictions to JMS resources and operations. Note that the JMS specification does not define any security aspects of the JMS service.

3. Where can I find the JMS specification?

You can find the Java Message Service 1.0.2 and 1.1 specifications at http://java.sun.com/products/jms/docs.html.

At the same location there is a link to the online JMS API documentation.

4. Where can I learn more about JMS?

The following link provides more information about JMS:

https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/55e7d51e-0e01-0010-7b84-a95ee55eadeb - Java Messaging.

Configuration

5. Can I use JMS with a Microsoft Windows client?

The JMS is part of the Sun's J2EE specification, thus you may use it on any platform that supports Java. We do not support any other way of invoking it (C++, C#, etc).

The client API that we provide is according to the standard specification. Please, have a look at http://java.sun.com/products/jms/tutorial/ for a tutorial.

To use the JMS on windows you will need a Java virtual machine (here is one from sun: http://java.sun.com/j2se/1.4.2/download.html)

Then in order to use the SAP JMS, you will need the following jars in the classpath:

For SAP NetWeaver 2004 and SAP NetWeaver 7.0 (2004s):

- `sapj2eeclient.jar`
- `exception.jar`
- `logging.jar`

For SAP NetWeaver Application Server, Java™ EE 5 Edition:

- `sap.com~tc~je~clientlib~impl.jar`
- `sap.com~tc~exception~impl.jar`
- `sap.com~tc~logging~java~impl.jar`

You may find these jars from your SAP NetWeaver installation at:

`\usr\sap\<sid>\<instanceid>\j2ee\j2eeclient`

- `jms.jar`

This is Sun's JMS API jar. You can also find the jms.jar in your SAP NetWeaver installation at `\usr\sap\<sid>\<instanceid>\j2ee\cluster\bin\ext\jms`.

6. In the jms-factories.xml there are <user-name> and <password> for connection to the queue. When should I use them?

The `<user-name>` and `<password>` are for the call to `connectionFactory.createConnection(user, password)`, which is performed by the JMS Connector upon application startup. They will be provided to the security mechanism of the JMS Provider. If the user can not be authenticated then a JMSSecurityException will be thrown as per the JMS specification.

7. How do I use a temporary destination?

You can dynamically create destinations that last as long as the lifetime of the connection, which are known as temporary destinations. Consumers of temporary destinations can be created only from the same connection that created the destination.

The temporary destinations:

- Are intended to facilitate a simple request/reply mechanism.
- Represent a message queue or topic to which a producer can send and from which a consumer can receive messages.
- Exist only for the lifetime of the connection in which they are created.
- Are deleted when the connection is closed, and all messages inside the destination are lost.
- Are not intended to survive a server failure.
- Only consumers created in the same connection as the temporary destination can receive messages from it, although all producers can send messages to it. Although you can easily create temporary destinations dynamically without the help of an administrator, they require as many resources on the provider side as regular destinations, so avoid using temporary destinations extensively.

Use the following source code:

```
//create a new temporary queue
Queue queue = session.createTemporaryQueue();
// create a new temporary topic
Topic topic = session.createTemporaryTopic();
```

8. How can I configure some delay between message delivery

attempts?

Since SAP NetWeaver 2004 SP19, it is possible to configure message redelivery and the delay between two consecutive delivery attempts for queues. When a failure occurs delivering a message, several consecutive attempts are made to deliver the problematic message before it is considered undeliverable (dead). The idea of message redelivery is to allow the application to recover from some temporary error condition that has happened inside the `onMessage()` method. The full process is explained in SAP Note 777930. However, if there is no delay between delivery attempts, it is less likely that the application problem would have been fixed before the last delivery attempt. This behavior has been enhanced by introducing configurable redelivery delay. The new queue property is called `deliveryDelayInterval` and its value is the time in milliseconds between two consecutive message delivery attempts. By default the new property has the value of 2000 milliseconds (2 seconds).

Persistence Stores

9. Does the SAP JMS Provider support Store-and-Forward Quality-of-Service?

Yes, since SAP NetWeaver 7.1 SP3, SAP JMS Provider supports the Store-and-Forward Quality-of-Service. This is a high-availability feature that is used to implement reliable JMS communication across clusters. Telling by its name, Store-and-Forward hides temporary network problems between two clusters by storing locally the messages sent during the network outage and forwards them to the remote cluster once it is accessible. This is done in a reliable way – all messages are delivered exactly once and in the order they have been produced.

Taking benefit of SAP JMS Store-and-Forward requires no changes in the application's code – enabling it, is a matter of simple configuration.

More information

http://help.sap.com/saphelp_nwce10/helpdata/en/46/ab187cba0b4e68e10000000a155369/frameset.htm

10. Which types of JDBC Databases does SAP JMS support?

The SAP JMS Provider uses the Java EE Server's database. The particular databases supported are:

- Oracle
- Microsoft SQL Server
- MaxDB (SAPDB)
- DB2 for Unix and Windows
- DB2 for OS/400 (known as DB4)
- DB2 for OS/390 (known as DB6)

Transaction Support

11. If the messages are being consumed within a transaction,

what should be done when a failover happens? Are the messages lost?

The handling of a transaction is similar to the handling of persistent messages and the client acknowledgement. In case the server fails when you are in a transaction and you have not yet committed, then an `onException` event will be sent, you must reconnect your connection and recreate all sessions and consumers. The messages part of the last transaction will be delivered again; each one will have the flag `JMSRedelivered`. Then it is up to you how to deal with those messages.

12. What happens if acknowledge() is called within a transaction?

As per the JMS specification, if a session is transacted, message acknowledgement is handled automatically by commit, and recovery is handled automatically by rollback.

If `acknowledge()` is called within a transaction, it is ignored.

13. What happens to a message that is rolled back or recovered?

If a transaction rollback is done, its produced messages are destroyed and its consumed messages are automatically recovered.

The recovered messages are sent back to the queue. Then they are redelivered to the first available consumer. If there is no consumer, the message remains in the queue until there is one.

JMS Programming Practices

14. Does the SAP Application Server Java offer "connection pooling" for the connection between JMS Provider and consumer?

No. There is no pooling of the connections to the consumers.

There is however a JMS sessions pool maintained by the JMS Connector.

15. Why is the <res-auth> sometimes set and sometimes not?

```
<resource-ref>
<res-ref-name>ConfirmationQueueFactory</res-ref-name>
<res-type>javax.jms.QueueConnectionFactory</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

That has to do with the authentication in case you create manually JMS connection inside your bean.

If you specify `<res-auth>`Application`</res-auth>`, then your application must supply the password/user in the `getConnection()` call. If you specify `<res-auth>`Container`</res-auth>`, then the connection credentials will be supplied by the container (they will be taken from additional security tags).

16. Am I supposed to use XATopicConnectionFactory instead of TopicConnectionFactory in case I want to use J2EE container transactions from the SAP Application Server Java?

No. You need to use `XATopicConnectionFactory` only when there are more resource managers involved in the same transaction (e.g. `javax.sql.DataSource`). In order to manage the different resources in the transaction, the transaction manager needs to perform the two-phase commit and therefore the involved resources should be XA-compliant. However, we recommend using `XATopicConnectionFactory` always when possible.

17. Is there any possibility to increase the priority of a message? If yes, does the API call

```
Public void publish (Topic topic, Message message, int deliveryMode,
int priority, long timeToLive) throws JMSException
```

on the interface TopicPublisher? Does it make sense to set a special priority setting or should I leave it and use the internal default priority?

That is exactly the way to increase the priority of a JMS message; however the priority increase means that it will be served before any other messages from the same topic. Note that with such priority increase neither the JMS will run in higher priority thread nor your application will be served with more resources. Just the message you are sending with higher priority is likely to go in front of the normal priority ones. In case you are sending only one type of messages (for example, cache invalidations) it will be useless to raise the priority.

18. I have a single topic and three different listeners registered to this topic. Does every published message exist as three different instances in the topic? What does it mean that a copy of the message is created for each one of the listeners?

When you have topic then all 3 subscribers will receive each their own COPY of the message. It may happen that the subscribers are on different server node, JVM-s and even physically different PCs with a firewall in between that allows only access to the central JMS Provider. Thus there is no way to synchronize and provide one object to all the consumers.

19. My listeners contain selectors. How does the topic function in this case, is an instance of a message created for each listener despite the selector of the listener does not match this message?

If your selector doesn’t match the message, then you subscriber will not even receive the copy of the message. No instance will be created and transported.

20. In the jms-factories.xml there are <security-principal> and <security-credentials> for obtaining JNDI context when the Queue/Topic and receiver are on different servers. When should I use them?

Those are for obtaining the `InitialContext` of the remote server.

```
env.put(Context.SECURITY_PRINCIPAL, user);
env.put(Context.SECURITY_CREDENTIALS, password);
```

```
Context ctx = new InitialContext(env);
```

Only lookup operations will be performed from the `InitialContext`.

21. In the jms-destination.xml in SAP NetWeaver 2004 and SAP NetWeaver 7.0 (2004s) There is a user/password tag. When should I use them?

The username and password elements in `jms-destinations.xml` are optional. They have the same semantics as the respective elements in `jms-factories.xml`, but, if supplied, the destination ones override the factory ones.

When the application is started, the descriptors are parsed, and the JMS Connector creates the required destinations using the standard JMS API calls (`createConnection`, `createQueue/createTopic`). In case username and password have been supplied in the descriptors, the JMS Connector will pass these as parameters to the `createConnection` invocation; otherwise `createConnection` will be invoked without parameters.

Note that these username and password of the `jms-destinations.xml` are used only during application startup.

Then, it is up to the application's JMS logic how the destinations will be accessed – by creating connection with or without supplying username and password. Moreover, the username and password passed may be different than the ones provided in the application's descriptors.

Or, to summarize, the username and password in `jms-factories.xml` and/or `jms-destinations.xml` are relevant only during application startup (and are used by the JMS Connector); they are not used after that in the application's own JMS logic.

22. What will happen to messages that were already sent but not yet consumed?

It depends on the message mode you used when you are sending the messages. If you use message mode PERSISTENT the messages will stay and will be delivered successfully. That will happen even in case of failure and in case of cluster restart. If you use NON_PERSISTENT there is no guarantee that the messages will stay. As you might guess using persistent flag will produce some performance overhead. It's up to you to choose the mode that best suits you.

23. What will happen to messages that have been consumed but not acknowledged?

There will be no way to send the acknowledgment in a new connection, the acknowledgment is per session, and once your connection has become inactive there is no way to "attach" the session to the new connection. All attempts to work with the old Session will end in exceptions. You must create a new session and the subscribers. In case you are using ordinary topics, the messages will be lost since the topics were just created, however if you are using durable subscribers or queues the PERSISTENT messages that were delivered to the old session but not yet acknowledged will be delivered to the new one. They will have the flag JMSRedelivered. You must code your application to handle duplicates based on that flag.

24. Can I create a situation in which all the listeners will get the message simultaneously, without waiting for the other listeners' onMessage to finish execute?

It depends on how you create your message listeners. If you create them within one JMS session they will be served sequentially. This is required by the JMS specification:

A Session uses a single thread to run all its MessageListeners. While the thread is busy serving one listener, all other messages pending to be asynchronously delivered to the session must wait.

Since you want your consumers to run in parallel – all you need to do is register your topic consumers on different JMS Sessions. In that way the messages for the other listeners will not have to wait.

Using the API it is possible for you to register one listener inside a servlet running on the web container of node X, an MDB application running on node Y and a third listener running on a PC Z that is not even part of the cluster. Communication between all those 3 Java virtual machines would be very hard to achieve.

Thus, it will not be possible for the JMS to provide any synchronization / ordering of message delivery to your listeners, so you should not rely on any such functionality.

25. How do I deal with a listener that does not keep up with messages being sent?

Have in mind that sends are in general faster than receives. You can decrease receive overhead by not acknowledging every message, that is by delaying the acknowledge until several messages have been received, using client acknowledge. With synchronous listeners, the last few messages may not get acknowledged without extra code logic if they do not fall on an acknowledge “boundary” (received 9 out of 10, but do not receive the 10th).

26. How do I get a thread dump to help track down a problem?

Refer to SAP Note 710154.

27. Should clientIDs of JMS connections be unique?

Yes. Using durable subscribers, if subscribing from two different clients and using the same connection factory and the connection factory has a configured clientID, unique clientIDs must be set for each `TopicConnection` created from it. If you configure a connection factory with a clientID, you can only create one `TopicConnection` with that factory until that connection is closed.

28. How do I manage a Queue to view and delete specific messages?

To view and delete specific messages from a queue you have to perform the following tasks:

- Write an application that uses a `QueueBrowser`.
- Delete specific messages by consuming them using a selector with the message identifier as shown in the example:

```
 String selector = "JMSMessageID = " + message.getMessageID() +
"";
```

Keep in mind that the queue browser is a not a "live" view of the queue. It is a snap-shot.

29. How do I close a Queue so that the messages will not be reloaded at the next server startup?

You can perform the following tasks:

- Acknowledge the messages;

- Use non-persistent messages;
- Use temporary queues. They exist only for the life of the connection;

These are the ways to close a queue so that the messages will not be reloaded at the next server startup. Otherwise, there is no API to explicitly delete the messages of a particular destination.

30. In what order are messages delivered to a consumer?

SAP JMS Provider maintains order between any producer and consumer.

SAP JMS Provider does not guarantee the order when multiple producers send to a single consumer or multiple consumers receive from multiple producers.

There are special cases when non-persistent messages can get ahead of persistent messages with higher priority. A recover or rollback puts messages that were already received back into the queue/topic and that can affect the order. SAP JMS Provider maintains the order between a producer and a destination and after that the order between the destination and the consumer. That means that once messages arrive at the destination, the order does not change.

31. Is it possible to have multiple queue receivers listening on the same queue, using message selectors (typically filtering on a correlation ID) to determine which listener actually receives the message?

SAP JMS Provider supports this behavior although it is not defined in the JMS Specification. When a message arrives on the queue, JMS looks at all the consumers for this queue in the same order they have connected to that queue. The first consumer with selector that matches the message receives it.

If the consumers receive the messages asynchronously you first have to set them the listener. However, each time an asynchronous queue consumer receives a message, the consumer goes to the end of the list.

Unlike topics, if a message does not match a selector, it is left in the queue until there is a consumer with no selector or a consumer with a selector that matches.

32. Is there a way to make a queue such that if one application has one object as listener on that queue, no other application can listen to the messages on that queue?

Yes. Every queue has a configuration property – `loadBalanceBehaviour`. This property defines the load balance behavior of the queue in case the destination has more than one consumer. The possible values are 1 (Exclusive - the registering of a second consumer will fail) and 3 (Round-robin - messages will be distributed among all registered consumers in a round-robin fashion).

If you have this property set to 1, then an attempt to register a second consumer on this queue will result in an exception.

33. Why setting values does not work using javax.jms.Message.setJMSPriority, deliveryMode, destination, timeStamp or expiration setters?

The respective setter methods in `javax.jms.Message` are to be used only by the JMS Provider. These message values are overwritten by the system on each send/publish. You should use the respective setters in the `MessageProducer`, `QueueSender`, or `TopicPublisher` classes (i.e., `setJMSPriority`, `setDeliveryMode`, `setTimeToLive`).

34. What care must be taken in multi-threaded JMS clients?

Concerning the JMS specification JMS sessions are single-threaded. Thus, if multiple threads simultaneously access a session or one of its consumers or producers the resulting behavior is undefined. In addition, if multiple asynchronous consumers exist on a session, messages will be delivered to them sequentially and not in parallel.

To take advantage of multiple threads with JMS, use multiple sessions. For example, to allow parallel synchronous receive requests, design the application so that only one consumer may be active per session and use multiple sessions.

35. How should an application be set up to subscribe to multiple topics?

When listening to N topics with N subscribers and N sessions you use up to N simultaneous threads of execution.

If you have N subscribers and 1 session you serialize all subscribers through that one session. In case of heavy load these subscribers may not be able to keep up without the extra threads. Also, if you are using CLIENT_ACKNOWLEDGE, N sessions give you N separate message streams that can be individually recovered. Having 1 session crosses the streams giving you less control.

If you use transaction better use N sessions and N subscribers.

36. What is better - to use blocking receive() calls or asynchronous message receiving?

The synchronous `receive()` method blocks until a message comes, until the specified timeout (if any) elapses or until the application is stopped. It is recommended to use asynchronous message receiving whenever possible. A synchronous `receive()` invocation consumes resources for the entire duration that the call is blocked. With the asynchronous message receiving, consumers are notified only when a relevant message comes, so no resources are consumed waiting for a message.

If the application logic however requires strictly synchronous message receiving, you should consider using `receive(timeout)`, selecting some minimum possible timeout to minimize blocking, or using `receiveNoWait()`, which returns the next message or a null value if no message is currently available. In the latter case, the call does not block. The application should provide a way to repeat the request, without calling `wait().` Finally, if both `receive(timeout)` and `receiveNoWait()` are not an option, consider configuring a larger thread pool to have more threads available than the number of possible simultaneous blocking `receive()` calls.

37. How do I publish an XML message?

SAP does not provide a specific XML message. If you still need to publish such message, follow these steps:

1. Use a text message and pass the XML into `message.setText`.
2. Publish the message.

Here is an example:

```
{
   TextMessage xMsg =
   pubSession.createTextMessage();
   StringBuffer msg = new StringBuffer();
   msg.append ("<?xml version=\"1.0\"?>\n");
   msg.append ("<message>\n");
   msg.append (" <sender>" + username + "</sender>\n");
   msg.append (" <content>" + content +s + "</content>\n");
   msg.append ("</message>\n");
   xMsg.setText (msg.toString());
   publisher.send (xMsg);
   }
```

38. Is it possible to send or receive a message from within a message listener?

Yes. You can send to or receive from any queue or topic from within a message listener.

If the consumer is not an MDB, you can use the same Connection or Session that the `onMessage()` is part of. When you create your message listener, you pass in the constructor a session as a parameter. Then you have access to the session in your `onMessage` method and you would be able to make synchronous calls from within the `onMessage()` method.

Do not use another Session that is servicing another `onMessage()`. This will multi-thread that Session and Sessions do not support multi-threading.

If the session is not transactional, there can be duplicates or lost messages (assuming your `onMessage( )` code is attempting to forward messages):

3. If you call acknowledge after the `publish()` and the acknowledge fails for some reason (network/server failure), then you will see the message again and will end up publishing twice (possible duplicate semantics). You can try to keep track of sequence numbers to detect duplicates but this is not easy.

4. If you call acknowledge before the `publish()`, you get at-most-once semantics. If the `publish()` fails, you don't know if the failure occurred before or after the message reached the server.

In the following example you have an MDB which receives asynchronously a request from a servlet, checks if it remains in a database (here instead of database we use the `nextInt()` method) and then sends a message back to the servlet:

```
package test;

import java.util.Random;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.jms.MapMessage;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.MessageProducer;
import javax.jms.TemporaryTopic;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicPublisher;
import javax.jms.TopicSession;

@MessageDriven(activationConfig = {
      @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Topic"),

      @ActivationConfigProperty(propertyName = "destination",
propertyValue = "JMSTestDestination"),

      @ActivationConfigProperty(propertyName =
"connectionFactoryName", propertyValue = "TopicConnectionFactory"),
})
public class ProcessBean implements MessageDrivenBean,
MessageListener {
   @Resource(name = "StorageRequirementsTopic")
   Topic top1;

   @Resource(name = "TransportationRequirementsTopic")
   Topic top2;

   @Resource(name = "TopicConnectionFactory")
   TopicConnectionFactory tcf;

   private static final long serialVersionUID = 1L;

   private static TopicConnection topicConnection;

   private TopicSession topicSession;

   public void onMessage(Message message) {

      System.err.println("Received message");
      try {
         if ((message instanceof MapMessage)
               && (message.getJMSReplyTo() != null)) {

            MapMessage requestMessage = (MapMessage) message;

            System.out.println("Received request");
            System.out.println("\tTime:       "
                  + System.currentTimeMillis() + " ms");
            System.out.println("\tMessage ID: "
                  + requestMessage.getJMSMessageID());
            System.out.println("\tCorrel. ID: "
                  + requestMessage.getJMSCorrelationID());
            System.out.println("\tReply to:   "
                  + requestMessage.getJMSReplyTo());

            topicConnection = (TopicConnection)
tcf.createTopicConnection();
            topicSession = topicConnection.createTopicSession(false,
                  TopicSession.AUTO_ACKNOWLEDGE);

            TemporaryTopic temporaryTopic = (TemporaryTopic) message
                  .getJMSReplyTo();
            MessageProducer replyProducer = topicSession
                  .createProducer(temporaryTopic);

            MapMessage replyMessage =
topicSession.createMapMessage();
            replyMessage.setJMSCorrelationID(requestMessage
                  .getJMSMessageID());

            Random r = new Random();
            String s = "";

            try {
               String reqID =
message.getStringProperty("RequirementID");
               switch (r.nextInt(2)) {

               case 0:
                  s = "Requested item is sent: " + reqID;
                  TopicPublisher topicPublisher1 = topicSession
                        .createPublisher(top1);
                  topicPublisher1.publish(message);

                  break;
               case 1:
                  s = "Requested item is not available";

                  TopicPublisher topicPublisher2 = topicSession
                        .createPublisher(top2);
                  topicPublisher2.publish(message);

                  break;
               }
            } catch (Exception e) {
               e.printStackTrace(System.err);
               replyMessage.setString("result", e.getMessage());
            }

            replyMessage.setString("result", s);

            replyProducer.send(replyMessage);

            System.err.println("message has been sent");

            System.out.println("Sent reply");
            System.out.println("\tTime:       "
                  + System.currentTimeMillis() + " ms");
            System.out.println("\tMessage ID: "
                  + replyMessage.getJMSMessageID());
            System.out.println("\tCorrel. ID: "
                  + replyMessage.getJMSCorrelationID());
            System.out.println("\tReply to:   "
                  + replyMessage.getJMSReplyTo());

            topicConnection.close();

         } else {
            System.out.println("Invalid message detected");
            System.out.println("\tType:       "
                  + message.getClass().getName());
            System.out.println("\tTime:       "
                  + System.currentTimeMillis() + " ms");
            System.out
                  .println("\tMessage ID: " +
message.getJMSMessageID());
            System.out.println("\tCorrel. ID: "
                  + message.getJMSCorrelationID());
            System.out.println("\tReply to:   " +
message.getJMSReplyTo());
```

```
            message.setJMSCorrelationID(message.getJMSMessageID());

            System.out.println("Sent to invalid message queue");
            System.out.println("\tType:       "
                  + message.getClass().getName());
            System.out.println("\tTime:       "
                  + System.currentTimeMillis() + " ms");
            System.out
                  .println("\tMessage ID: " +
message.getJMSMessageID());
            System.out.println("\tCorrel. ID: "
                  + message.getJMSCorrelationID());
            System.out.println("\tReply to:   " +
message.getJMSReplyTo());
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }

   public void ejbRemove() {

      System.out.println("In MDBEjbBean.remove()");

   }

   public void setMessageDrivenContext(MessageDrivenContext context)
{

      System.out.println("In" +
"MDBEjbBean.setMessageDrivenContext()");
   }

   public void ejbCreate() {

      System.out.println("In MDBEjbBean.ejbCreate()");

   }
}
```

39. Is it better to have more or fewer sessions for a given number of subscribers?

It is better to have more sessions. When you use N sessions with N subscribers you have N simultaneous threads. Each session gets its own thread as long as there are enough threads available. Otherwise, the sessions sequentially reuse the threads.

If you have one session with N subscribers all subscribers are serialized through that session. In case of heavy load these subscribers may not be able to keep up without extra threads.

If you are using CLIENT_ACKNOWLEDGE, N sessions gives you N separate message streams that can be individually recovered. Having one session crosses the streams giving you less control.

40. How do I handle request/response behavior using JMS?

There are several approaches to handling request/response processing with JMS.

Use a temporary queue for each requestor and have the response go back to that queue.

In the following example the Requestor sends a request on a Queue to the Replier. The Replier receives the request asynchronously using `onMessage()` method and then sends a response back to the Requestor using Temporary Queue.

Here is the example. The following source code is JMS 1.1 compliant API.

Requestor:

```
    temporaryQueue = session.createTemporaryQueue();

    requestProducer = session.createProducer(sapDemoQueue);

    replyConsumer = session.createConsumer(temporaryQueue);
```

Replier:

```
    TemporaryQueue temporaryQueue = (TemporaryQueue)
    message.getJMSReplyTo();

    MessageProducer replyProducer =
    queueSession.createProducer(temporaryQueue);
```

In the Replier you get the Temporary Queue in the `onMessage()` method.

41. How do I put a message back on the queue for processing?

You can do one of the following:

- Use a transacted session, then rollback the session so the message will go back to the queue.
- Use Session.CLIENT_ACKNOWLEDGE when creating a session, and then recover the session so the message will go back to the queue.
- Use a JTA transaction, then rollback the transaction so the message will go back to the queue.

42. Is it right to add new sessions and subscribers to a queue or topic connection once it has been started?

Yes, under one condition. You may not add new subscribers/consumers to a session if it already has active asynchronous consumers. According to the JMS Specification sessions must only be accessed by a single thread. If you still need to add new subscribers/consumers to a session, create a new Session and add them.

You can add receivers to a started connection. Receivers are not asynchronous. That is why you need a listener to make them asynchronous. In a synchronous receiver you receive the messages with `receiver.receive()`. When you add a listener the receiver becomes asynchronous and then you receive the messages with `onMessage`() method. If in one session you use `receive`() and `onMessage`() as well then the `receive`() method is ignored and will not receive its messages. If you want to have both synchronous and asynchronous receivers use separate sessions.

43. Why have different connection factories?

Each connection factory has a set of connection attributes. With different connection factories you get different sets of connection attributes. This is very convenient for clients that need different behaviors. If all your clients need the same behavior, then one connection factory is sufficient.

44. How should connections and sessions be allocated?

Connection is an object representing a physical connection to the JMS Provider. Due to the authentication and communication done when a connection is created, the connection is relatively heavyweight. Most clients need to use a single connection to do their messaging. A session is a single-threaded context for sending and receiving messages. JMS clients typically create one or more sessions so that they can use more threads. If you have thread groups and need to start/stop/close the resources for a given group, one connection per group is a good idea. Each group can have exactly one thread.

45. Is there a way to dynamically change the selector of a topic consumer?

No. The JMS specification does not define any methods for changing the selector of an existing subscriber. A selector can be provided only in the `TopicSubscriber` constructor. Changing the message selector can be simulated by first removing the existing consumer and then creating it with a different selector.

Message-Driven Beans

46. What are the advantages of MDBs compared to standard message listeners?

The message-driven bean is a stateless component that is invoked by the EJB container as a result of receiving JMS messages. It then performs business logic based on the message content, freeing the developers from any JMS messaging and failover specifics. One of the main advantages of using message-driven beans instead of a standard JMS `MessageListener` is that a JTA transaction can be started automatically and the received message will be part of that transaction. In this case, other operations can be enlisted in the same JTA transaction such as database operations. This is the only way to associate a message from an asynchronous consumer and another JTA operation with the same transaction.

47. According to help.sap.com, "*A durable subscription can have only one active subscriber at a time.*" But there is more than one ejbCreate() called even though the "durable" attribute is set. should we expect only one bean instance with durable property?

The requirement that the durable subscriber is unique for a name is from the JMS specification.

The messages to MDBs are delivered via a ConnectionConsumer that is only one for the whole pool. There is only one such consumer and that is why it does not violate the specification. If you attempt to create a pure JMS durable topic subscriber with the same name, while the MDB pool is still active then you end in exception. Thus there could be many MDB instances even though the subscription is durable. If you expect only one instance, then you may put the property MaxSize to 1 and use a blocking pool. This property indicates the initial number of instances that can exist in the pool. The MaxSize property is fully functional if you replace the standard EJB pool with a blocking one – one that is not resized in case the maximum value is reached.

For more detailed description of the properties of the pool, refer to:

http://help.sap.com/saphelp_nw04/helpdata/de/37/30c557fad05341a951cfd051bf0b44/content.htm

48. According to the documentation I must specify resource reference to the used queue in the deployment descriptor. I have not done this but it works anyway. What should I do?

If your application is deployed and works properly, than it seems that you or perhaps the SAP NetWeaver Developer Studio automatically has supplied correctly the reference to the queue. Otherwise the server will have no idea from where to take the messages for your MDB. The reference to the resources should be in that deployment descriptor – `ejb-j2ee-engine.xml` or `ejb-jar.xml`. In Java EE 5 annotations are introduced which are an alternative to the deployment descriptor.

49. What happens to the message delivery if the MDB throws an exception causing a transaction rollback or a different EJB in the same transaction throwing such an exception? Is the message redelivered then?

The message will be redelivered again 10 times and then it will be discarded if it is again causing exception and rollbacks. The assumption is that in case a message causes 10 times exception/rollbacks than there will be problem on the 11th try and there is a need to stop it in order to avoid endless cycle and 100% CPU usage.

This functionality is currently available only for queues! For topics and durable subscription the message will be delivered forever until the processing succeeds.

50. Can an MDB be a message producer or both a producer and consumer?

Yes. There is no JMS context inside the MDB. All you have to do is to create a connection, session and a producer yourself.

- You can create a connection, session and producer every time you come into the `onMessage()` method of the MDB. This is sufficient if the message rate is relatively low.
- You can create the necessary objects in `ejbActivate`().

  The objects are not serializable. They cannot be passivated for a stateful session bean. When the EJB is deactivated, you need to close the associated objects.

51. If an MDB uses a durable subscription, will messages be accumulated If the MDB is not deployed?

No. Before you deploy the MDB for the first time there are no messages accumulated. Like stateless session beans, a message-driven bean is never passivated, and it has only two states: nonexistent and ready to receive messages. That means that you need to deploy the MDB to make it active.

52. How do I configure the JMS resources of an MDB?

You need to configure the XML files in order to make the application running.

You have three XML files:

5. `jms-resources.xml` - defines the JMS resources.
6. `ejb-j2ee-engine.xml` - declares the destination and factories which will be needed for the MDB to receive messages.

7. `ejb-jar.xml` - specifies the type of the destination to which the MDB will be registered and will receive messages.

If your destination was named in `jms-resources.xml` file as MyTestQueue as below (step 1)

```
<destination>
   <name>MyTestQueue</name>
   <type>javax.jms.Queue</type>
   <sap-local-destination-type>
      <virtual-provider>default</virtual-provider>
   </sap-local-destination-type>
</destination>
```

and this is the destination to which your MDB is listening to, you should specify this name under `<message-props>` tag from `ejb-j2ee-engine.xml` (step 2)

```
<message-props>
   <destination-name>MyTestQueue</destination-name>
   <connection-factory-name>
      MyTestQConnFactory
   </connection-factory-name>
</message-props>
```

As you see the same is valid for factories as well.

In addition you have to declare in the `ejb-jar.xml` that your MDB will listen to a destination of type queue (in this example), and it is done in the following way (step 3)

```
<message-driven-destination>
   <destination-type>javax.jms.Queue</destination-type>
</message-driven-destination>
```

Tag `<message-driven-destination>` is child of `<message-driven>`, which is a child of `<enterprise-beans>` tag of `ejb-jar.xml`.

The names which are defined are the “short” names (not lookup names). The lookup name of every of this resources will be correspondingly:

jmsqueues/default/MyTestQueue

jmsfactories/default/MyTestQConnFactory

If you want to send a message to a destination from your EJB or Web Component, then the declaration of the JMS resources is different.

`Jms-resources.xml` is not changing but `ejb-jar.xml` has to declare them as resource references for example:

```
<resource-ref>
   <res-ref-name>MyTestQConnFactory</res-ref-name>
   <res-type>javax.jms.QueueConnectionFactory</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
<resource-env-ref>
   <resource-env-ref-name>MyTestQueue</resource-env-ref-name>
   <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

53. Is it possible to somehow increase the performance of MDBs?

Yes, for some scenarios. Since SAP NetWeaver 2004 SP19 (and also in SAP NetWeaver Application Server, Java^{TM} EE 5 Edition), a new property, `parallel-consumers`, has been introduced in the deployment descriptor `ejb-j2ee-engine.xml`. That can be used to tune the performance for transacted MDBs on queues that are doing long blocking operations inside their `onMessage` method. If the property is not specified, or has value 1, the server functionality will not change in any way.

For example, if you define

```
<property>
   <property-name>parallel-consumers</property-name>
   <property-value>10</property-value>
</property>
```

When you send messages to a queue, the JMS scales these messages into 10 “mini-queues” and each such queue “feeds” one bean instance. So if you send a great number of messages, they are scaled among these 10 MDB instances. However, if you publish only 2 messages from different connections, they potentially could fall into one and the same mini-queue and be delivered sequentially.

The new property is recommended only in case the application is performing some long blocking operations inside the `onMessage()` method, like for example network I/O calls.