A Message-Driven Bean Example

Message-driven beans can implement any messaging type.

Most commonly, they implement the Java Message Service (JMS) technology.

The example in this chapter uses JMS technology, so you should be familiar with basic JMS concepts such as queues and messages.

To learn about these concepts, see Chapter 45, Java Message Service Concepts.

This chapter describes the source code of a simple message-driven bean example.

Before proceeding, you should read the basic conceptual information in the section What Is a Message-Driven Bean? as well as Using Message-Driven Beans to Receive Messages Asynchronously.

The following topics are addressed here:

- `simplemessage` Example Application Overview
- The `simplemessage` Application Client
- The Message-Driven Bean Class
- Packaging, Deploying, and Running the `simplemessage` Example

`simplemessage` Example Application Overview

The `simplemessage` application has the following components:

- `SimpleMessageClient`: An application client that sends several messages to a queue

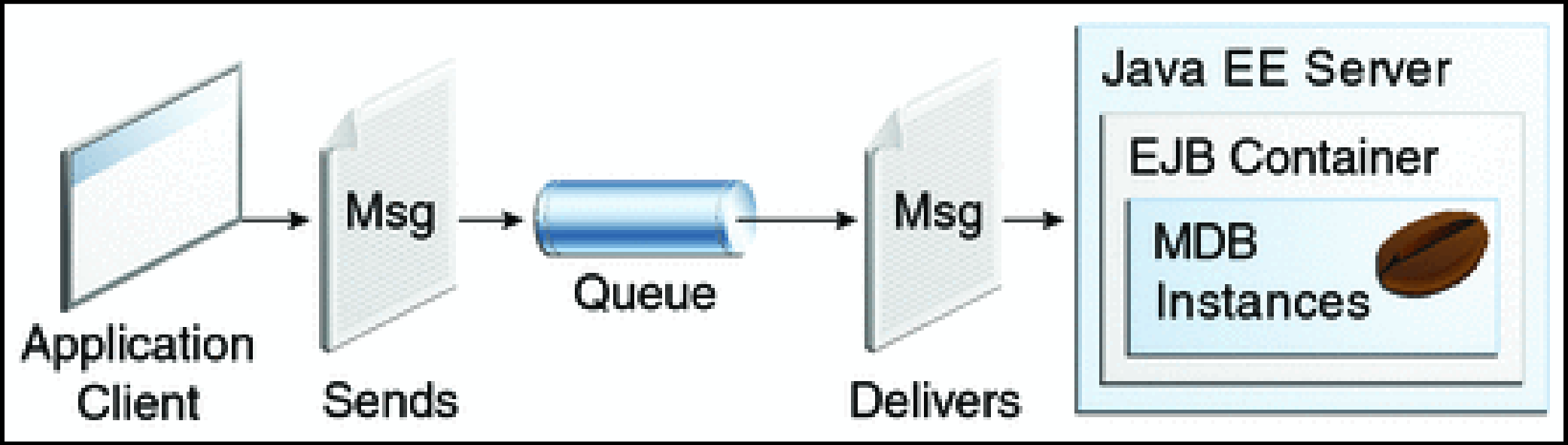
- `SimpleMessageBean`: A message-driven bean that asynchronously receives and processes the messages that are sent to the queue

Figure 25-1 illustrates the structure of this application.

The application client sends messages to the queue, which was created administratively using the Administration Console.

The JMS provider (in this case, the GlassFish Server) delivers the messages to the instances of the message-driven bean, which then processes the messages.

Figure 25-1 The `simplemessage` Application



The source code for this application is in the *tut-install*`/examples/ejb/simplemessage/` directory.

The `simplemessage` Application Client

The `SimpleMessageClient` sends messages to the queue that the `SimpleMessageBean` listens to.

The client starts by injecting the connection factory and queue resources:

```
@Resource
(mappedName="jms/ConnectionFactory")
private static ConnectionFactory
connectionFactory;
@Resource(mappedName="jms/Queue")
private static Queue queue;
```

Next, the client creates the connection, session, and message producer:

```
connection =
connectionFactory.createConnection();
session = connection.createSession
(false, Session.AUTO_ACKNOWLEDGE);
messageProducer =
session.createProducer(queue);
```

Finally, the client sends several messages to the queue:

```
message =
session.createTextMessage();
for (int i = 0; i < NUM_MSGS; i++){
message.setText
("This is message " + (i + 1));
System.out.println
("Sending message: " +
message.getText());
messageProducer.send(message);
}
```

The Message-Driven Bean Class

The code for the `SimpleMessageBean` class illustrates the requirements of a message-driven bean class:

- It must be annotated with the `@MessageDriven` annotation if it does not use a deployment descriptor.

- The class must be defined as `public`.
- The class cannot be defined as `abstract` or `final`.
- It must contain a public constructor with no arguments.
- It must not define the `finalize` method.

It is recommended, but not required, that a message-driven bean class implement the message listener interface for the message type it supports.

A bean that supports the JMS API implements the `javax.jms.MessageListener` interface.

Unlike session beans and entities, message-driven beans do not have the remote or local interfaces that define client access.

Client components do not locate message-driven beans and invoke methods on them.

Although message-driven beans do not have business methods, they may contain helper methods that are invoked internally by the `onMessage` method.

For the GlassFish Server, the `@MessageDriven` annotation typically contains a `mappedName` element that specifies the JNDI name of the destination from which the bean will consume messages.

For complex message-driven beans, there can also be an `activationconfig` element containing `@ActivationConfigProperty` annotations used by the bean.

A message-driven bean can also inject a `MessageDrivenContext` resource.

Commonly you use this resource to call the `setRollbackOnly` method to handle exceptions for a bean that uses container-managed transactions.

Therefore, the first few lines of the `SimpleMessageBean` class look like this:

```
@MessageDriven(
mappedName="jms/Queue",
activationConfig = {
@ActivationConfigProperty(
propertyName = "acknowledgeMode",
propertyValue =
"Auto-acknowledge"),
@ActivationConfigProperty(
propertyName = "destinationType",
propertyValue = "javax.jms.Queue"
) })
```

```
public class SimpleMessageBean
implements MessageListener {
@Resource
private MessageDrivenContext mdc;
...
```

NetBeans IDE typically creates a message-driven bean with a default set of `@ActivationConfigProperty` settings.

You can delete those you do not need, or add others.

Table 25-1 lists commonly used properties.

Table 25-1 `@ActivationConfigProperty` Settings for Message-Driven Beans

Property Name	Description
`acknowledgeMode`	Acknowledgment mode; see Controlling Message Acknowledgment for information
`destinationType`	Either `javax.jms.Queue` or `javax.jms.Topic`

`subscriptionDurability`	For durable subscribers, set to `Durable`; see Creating Durable Subscriptions for information
`clientId`	For durable subscribers, the client ID for the connection
`subscriptionName`	For durable subscribers, the name of the subscription
`messageSelector`	A string that filters messages; see JMS Message Selectors for information, and see An Application That Uses the JMS API with a Session Bean for an example
`addressList`	Remote system or systems to communicate with; see An Application Example That Consumes Messages from a Remote Server for an example

The `onMessage` Method

When the queue receives a message, the EJB container invokes the message listener method or methods.

For a bean that uses JMS, this is the `onMessage` method of the `MessageListener` interface.

A message listener method must follow these rules:

- The method must be declared as `public`.
- The method must not be declared as `final` or `static`.

The `onMessage` method is called by the bean's container when a message has arrived for the bean to service.

This method contains the business logic that handles the processing of the message.

It is the message-driven bean's responsibility to parse the message and perform the necessary business logic.

The `onMessage` method has a single argument: the incoming message.

The signature of the `onMessage` method must follow these rules:

- The return type must be `void`.
- The method must have a single argument of type `javax.jms.Message`.

In the `SimpleMessageBean` class, the `onMessage` method casts the incoming message to a `TextMessage` and displays the text:

```
public void onMessage
(Message inMessage) {
TextMessage msg = null;
try {
if
(inMessage instanceof TextMessage){
msg = (TextMessage) inMessage;
logger.info(
"MESSAGE BEAN: Message received: "
+ msg.getText());
} else {
```

```
logger.warning(
"Message of wrong type: " +
inMessage.getClass().getName());
}
} catch (JMSException e) {
e.printStackTrace();
mdc.setRollbackOnly();
} catch (Throwable te){
te.printStackTrace();
}
}
```

Packaging, Deploying, and Running the `simplemessage` Example

To package, deploy and run this example, go to the *tut-install*`/examples/ejb/simplemessage/` directory.

Creating the Administered Objects for the `simplemessage` Example

This example requires the following:

- A JMS connection factory resource
- A JMS destination resource

If you have run the simple JMS examples in Chapter 45, Java Message Service Concepts and have not deleted the resources, you already have these resources and do not need to perform these steps.

You can use Ant targets to create the resources.

The Ant targets, which are defined in the `build.xml` file for this example, use the `asadmin` command.

To create the resources needed for this example, use the following commands:

```
ant create-cf ant create-queue
```

These commands do the following:

- Create a connection factory resource named `jms/ConnectionFactory`
- Create a destination resource named `jms/Queue`

The Ant targets for these commands refer to other targets that are defined in the *tut-install*`/examples/bp-project/app-server-ant.xml` file.

To Build, Deploy, and Run the `simplemessage` Application Using NetBeans IDE

1. From the File menu, choose Open Project.

2. In the Open Project dialog, navigate to:
   `tut-install``/examples/ejb/`

3. Select the `simplemessage` folder.

4. Select the Open as Main Project check box and the Open Required Projects check box.

5. Click Open Project.

6. In the Projects tab, right-click the `simplemessage` project and choose Build.

This task packages the application client and the message-driven bean, then creates a file named `simplemessage.ear` in the `dist` directory.

7. Right-click the project and choose Run.

This command deploys the project, returns a JAR file named `simplemessageClient.jar`, and then executes it.

The output of the application client in the Output pane looks like this (preceded by application client container output):

```
Sending message: This is message 1
Sending message: This is message 2
Sending message: This is message 3
To see if the bean received the
messages, check
<install_dir>/domains/domain1/logs/
server.log.
```

The output from the message-driven bean appears in the server log (*domain-dir*`/logs/server.log`), wrapped in logging information.

```
MESSAGE BEAN: Message received:
This is message 1
MESSAGE BEAN: Message received:
This is message 2
MESSAGE BEAN: Message received:
This is message 3
```

The received messages may appear in a different order from the order in which they were sent.

To Build, Deploy, and Run the `simplemessage` Application Using Ant

1. In a terminal window, go to:

`tut-install``/examples/ejb/simplemessage/`

2. To compile the source files and package the application, use the following command:

```
ant
```

This target packages the application client and the message-driven bean, then creates a file named `simplemessage.ear` in the `dist` directory.

By using resource injection and annotations, you avoid having to create deployment descriptor files for the message-driven bean and application client.

You need to use deployment descriptors only if you want to override the values specified in the annotated source files.

3. To deploy the application and run the client using Ant, use the following command:

```
ant run
```

Ignore the message that states that the application is deployed at a URL.

The output in the terminal window looks like this (preceded by application client container output):

```
Sending message: This is message 1
Sending message: This is message 2
Sending message: This is message 3
To see if the bean received the
messages, check
<install_dir>/domains/domain1/logs/
server.log.
```

In the server log file, the following lines appear, wrapped in logging information:

```
MESSAGE BEAN: Message received:
This is message 1
MESSAGE BEAN: Message received:
This is message 2
MESSAGE BEAN: Message received:
This is message 3
```

The received messages may appear in a different order from the order in which they were sent.

Removing the Administered Objects for the `simplemessage` Example

After you run the example, you can use the following Ant targets to delete the connection factory and queue:

```
ant delete-cf ant delete-queue
```