A message driven bean is a type of enterprise bean which is invoked by EJB container when it receives a message from queue or topic. Message driven bean is a stateless bean and is used to do task asynchronously.

To demonstrate use of message driven bean, we'll make use of ejb-persistence chapter and we're going to do the following tasks.

- **Step 1.** Create table in database (Refer to *EJB-Persistence* chapter).
- **Step 2.** Create Entity class corresponding to table (Refer to *EJB-Persistence* chapter).
- **Step 3.** Create DataSource and Persistence Unit (Refer to *EJB-Persistence* chapter).
- **Step 4.** Create a stateless ejb having EntityManager instance (Refer to *EJB-Persistence* chapter).
- **Step 5.** Update stateless ejb.Add methods to add records and get records from database via entity manager (Refer to *EJB-Persistence* chapter).
- **Step 6.** Create a Queue named **BookQueue** in JBoss default application directory.
- **Step 7.** A console based application client will send message to this queue.
- **Step 8.** Create a Message driven bean which will use the stateless bean to persist the client data.
- **Step 9.** EJB Container of jboss will call the above message driven bean and pass it the message that client will be sending to.

### Create Queue

Create a file named jbossmq-destinations-service.xml if not exists in **<JBoss Installation Folder> > server > default > deploy** folder.

Here we're creating a queue named BookQueue

```xml
<mbean code="org.jboss.mq.server.jmx.Queue"
   name="jbossmq:destination:service=Queue,name=BookQueue">
   <depends optional-attribute-name="DestinationManager">
       jbossmq:service=DestinationManager
   </depends>
</mbean>
```

When you start the JBoss, you'll see the a similar entry in jboss log

```
```

### Create Message Driven Bean

```java
@MessageDriven(name = "BookMessageHandler",
   activationConfig = {
       @ActivationConfigProperty(propertyName = "destinationType",
           propertyValue = "javax.jms.Queue"),
       @ActivationConfigProperty(propertyName = "destination", propertyValue = "/queue/BookQueue")
   })
```
```java
public class LibraryMessageBean implements MessageListener {

@Resource
private MessageDrivenContext mdctx;

@EJB
LibraryPersistentBeanRemote libraryBean;

public LibraryMessageBean()
{
}

public void onMessage(Message message) {
}
}
```

- LibraryMessageBean is annotated with @MessageDriven annotation to mark it as message driven bean.
- Its properties are defined as destinationType - Queue and destination - /queue/BookQueue.
- It implements MessageListener interface which exposes onMessage method.
- It has MessageDrivenContext as resource.
- LibraryPersistentBeanRemote stateless bean is injected in this bean for persistence purpose.

Build the EjbComponent project and deploy it on JBoss. After building and deploying the ejb module, we need a client to send a message to jboss queue.

**Example Application**

Let us create a test EJB application to test Message Driven Bean.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name EjbComponent under a package com.tutorialspoint.entity as explained in the EJB - Create Application chapter. You can also use the project created in EJB - Create Application chapter as such for this chapter to understand ejb persistence concepts.</td>
</tr>
<tr>
<td>2</td>
<td>Create Book.java under package com.tutorialspoint.entity as created in EJB-Persistence chapter</td>
</tr>
<tr>
<td>3</td>
<td>Create LibraryPersistentBean.java and LibraryPersistentBeanRemote as created in EJB-Persistence chapter</td>
</tr>
<tr>
<td>4</td>
<td>Create jboss-ds.xml in EjbComponent &gt; setup folder and persistence.xml in EjbComponent &gt; src &gt; conf folder. These folder can be seen in files tab in Netbeans as created in EJB-Persistence chapter</td>
</tr>
<tr>
<td>5</td>
<td>Create LibraryMessageBean.java under a package com.tutorialspoint.messagebean and modify it as shown below.</td>
</tr>
<tr>
<td>6</td>
<td>Create BookQueue queue in Jboss as described above.</td>
</tr>
<tr>
<td>7</td>
<td>Clean and Build the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>8</td>
<td>Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.</td>
</tr>
<tr>
<td>9</td>
<td>Now create the ejb client, a console based application in the same way as explained in the EJB - Create Application chapter under topic Create Client to access EJB. Modify it as shown below.</td>
</tr>
</tbody>
</table>
### EJBClient EJBComponent

#### LibraryMessageBean.java

```java
package com.tutorialspoint.messagebean;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

@MessageDriven(name = "BookMessageHandler",
activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType",
                             propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destination",
                             propertyValue = "/queue/BookQueue")
})
public class LibraryMessageBean implements MessageListener {
    @Resource
    private MessageDrivenContext mdctx;

    @EJB
    LibraryPersistentBeanRemote libraryBean;

    public LibraryMessageBean(){}

    public void onMessage(Message message) {
        ObjectMessage objectMessage = null;
        try {
            objectMessage = (ObjectMessage) message;
            Book book = (Book) objectMessage.getObject();
            libraryBean.addBook(book);
        } catch (JMSException ex) {
            mdctx.setRollbackOnly();
        }
    }
}
```

### EJBTester EJBClient

#### EJBTester.java

```java
package com.tutorialspoint.test;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.jms.ObjectMessage;
```
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {
    BufferedReader brConsoleReader = null;
    Properties props;
    InitialContext ctx;
    {
        props = new Properties();
        try {
            props.load(new FileInputStream("jndi.properties"));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        try {
            ctx = new InitialContext(props);
        } catch (NamingException ex) {
            ex.printStackTrace();
        }
        brConsoleReader = new BufferedReader(new InputStreamReader(System.in));
    }
    public static void main(String[] args) {
        EJBTester ejbTester = new EJBTester();
        ejbTester.testMessageBeanEjb();
    }
    private void showGUI(){
        System.out.println("*******************");
        System.out.println("Welcome to Book Store");
        System.out.println("*******************");
        System.out.print("Options \n1. Add Book \n2. Exit \nEnter Choice: ");
    }
    private void testMessageBeanEjb(){
        try {
            int choice = 1;
            Queue queue = (Queue) ctx.lookup("/queue/BookQueue");
            QueueConnectionFactory factory = (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
            QueueConnection connection = factory.createQueueConnection();
            QueueSession session = connection.createQueueSession(false, QueueSession.AUTO_ACKNOWLEDGE);
            QueueSender sender = session.createSender(queue);
            while (choice != 2) {
                String bookName;
                showGUI();
                String strChoice = brConsoleReader.readLine();
                choice = Integer.parseInt(strChoice);
                if (choice == 1) {
                    System.out.println("Enter book name: ");
                    bookName = brConsoleReader.readLine();
                    Book book = new Book();
                    book.setName(bookName);
                    ObjectMessage objectMessage = session.createObjectMessage(book);
                    sender.send(objectMessage);
                } else if (choice == 2) {
                    showGUI();
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.

- In testStatefulEjb method, jndi lookup is done with name - "/queue/BookQueue" to obtain reference of queue available in Jboss. Then sender is created using queue session.

- Then user is shown a library store User Interface and he/she is asked to enter choice.

- If user enters 1, system asks for book name and sender sends the book name to queue. When JBoss container receives this message in queue, it calls our message driven bean's onMessage method. Our message driven bean then saves book using stateful session bean addBook method. Session Bean is persisting the book in database via EntityManager call.

- If user enters 2, then another jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object statefuljb again and listing of books is done.

Run Client to access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select run file.

Verify the following output in Netbeans console.
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 2
1. learn java
1. learn EJB
BUILD SUCCESSFUL (total time: 15 seconds)

- Output shown above states that our Message driven bean is receiving the message and storing book in persistent storage and books are retrieved from database.