

IBM WEBSHERE APPLICATION SERVER V6.1 FEATURE PACK FOR EJB 3.0– LAB EXERCISE

Adding a message-driven bean to the EJB 3.0 sample

What this exercise is about	1
Lab Requirements	1
What you should be able to do	2
Exercise Instructions	3
Part 1: Configure messaging resources on the application server	4
Part 2: Modify the sample application	8
Part 3: Develop a message-driven bean.....	12
Part 4: Create bindings files	15
Part 5: Test the Application	17
What you did in this exercise	18

What this exercise is about

The objective of this lab is to demonstrate how to create a message-driven bean as part of an EJB 3.0 application on WebSphere Application Server V6.1 with the Feature Pack for EJB 3.0.

The primary purpose of this exercise is to demonstrate the implementation and configuration of an EJB 3.0 message-driven bean. It is not necessarily intended to demonstrate "best practices" for overall application design.

You will first create service integration and JMS resources using the WebSphere Application Server administrative

You will create a message-driven bean using the Application Server Toolkit, and add it to the existing EJB 3.0 sample application that is provided with the feature pack. You will then create the appropriate bindings files, and modify the provided sample to access a JMS queue using the MDB.

Lab requirements

- This exercise assumes that WebSphere Application Server V6.1 with the Feature Pack for EJB 3.0 is already installed and an application server profile is already created with administrative security enabled.
- Lab source files (the Labfiles61EJB3 directory) must be downloaded and extracted. This directory is included in the archive file that contained this document (WASv61_EJB3_Labs.zip). The exercise will use <LAB_FILES> to refer to the directory in which you have extracted the archive file.

What you should be able to do

At the end of this lab you should be able to:

- Configure the Application Server Toolkit to develop EJB 3.0 artifacts
 - Package an enterprise application with EJB 3.0 modules for deployment to a local WebSphere Application Server installation
-

Exercise instructions

Security note: For the purposes of this exercise, it is assumed that administrative security has been enabled, and that a user with the name **wsdemo** and password **password** has full administrative rights. If a different administrator exists on your system, replace the sample user credentials with the actual user credentials when necessary.

For the purposes of this exercise, it is assumed that a profile was created using the default port values. If different port numbers are used on your system, substitute port numbers as necessary.

For the purposes of this exercise, the following directory locations are assumed. If different locations are in use in your environment, replace the sample paths with the actual path on your system

Location	Windows®	UNIX®/Linux®
WebSphere Application Server installation	C:\WebSphere\AppServer\	/opt/WebSphere/AppServer
Lab files location	C:\Labfiles61EJB3	var/tmp/Labfiles61EJB3

Part 1: Configure messaging resources on the application server

___ 1. Verify that server1 is started

___ a. Navigate to the profile's **bin** directory in a command prompt.

Windows

```
cd c:\WebSphere\AppServer\profiles\AppSrv01\bin
```

Linux

```
cd /opt/WebSphere/AppServer/profiles/AppSrv01/bin
```

___ b. Use the **serverStatus** command to verify that the application server is started.

Windows

```
serverStatus.bat server1 -username wsdemo
-p password
```

Linux

```
./serverStatus.sh server1 -username wsdemo
-p password
```

___ c. If the server status indicates STOPPED, then start the server:

Windows

```
startServer.bat server1 -username wsdemo
-p password
```

Linux

```
./startServer.sh server1 -username wsdemo
-p password
```

___ 2. Log in to the administrative console

___ a. Open a Web browser and navigate to <https://localhost:9043/ibm/console> (substituting the correct port number if necessary).

___ b. Enter your administrative user ID and password (such as *wsdemo* and *password*), and click **Log in**.

___ 3. Create a service integration bus.

___ a. From the administrative console's left menu, expand **Service integration**, and click **Buses**.

___ b. Click the **New** button in the "Buses" panel.

___ c. Enter the name **bus1**.

___ d. Clear the **Bus security** check box.

___ e. Click **Next**.

___ f. Click **Finish**.

___ 4. Add a bus member.

___ a. Click the newly created **bus1** link.

___ b. Click **Bus members**, found under the "Topology" heading.


___ c. Click **Add**.

- ___ d. Select server1 by ensuring that the **Server** radio button is selected, and that your server appears in the pop-up menu, similar to the image below.

Add a server, cluster or a WebSphere MQ server as a new member of the bus.

- ___ e. Click **Next**.
 - ___ f. Ensure that the **File store** radio button is selected, and click **Next**.
 - ___ g. Click **Next** again to accept the default settings for the file store.
 - ___ h. Click **Finish**.
- ___ 5. Create a queue.
- ___ a. In the administrative console's left menu, expand **Service integration** and click **Buses**.
 - ___ b. Click **bus1**.
 - ___ c. Click **Destinations**, under the heading "Destination resources".
 - ___ d. Click **New**.
 - ___ e. Select **Queue** and click **Next**.
 - ___ f. Enter the name **SIBQueue** in the "identifier" field and click **Next**.
 - ___ g. Ensure that the correct bus member is selected (you should have only one choice in this exercise) and click **Next**.
 - ___ h. Click **Finish**.
- ___ 6. Save your changes to the master repository.
- ___ a. Click **Save** in the "messages" box.
- ___ 7. Create a JMS connection factory
- ___ a. In the left menu, expand **Resources**, expand **JMS**, and click **Connection Factories**.
 - ___ b. Use the pop-up menu to select the server scope, **Node=<node-name>**, **Server=server1**.
 - ___ c. Click **New**.

- ___ d. Select **Default Messaging Provider** and click **OK**.
 - ___ e. Enter **counterQCF** in the “name” field.
 - ___ f. Enter **jms/counterQCF** in the “JNDI name” field.
 - ___ g. Select **bus1** from the “Bus name” menu.
 - ___ h. Scroll to the bottom of the page and click **OK**.
- ___ 8. Create a JMS queue.
- ___ a. From the already-expanded “JMS” sub-menu, click **Queues**.
 - ___ b. Use the pop-up menu to select the server scope, **Node=<node-name>**, **Server=server1**.
 - ___ c. Click **New**.
 - ___ d. Select **Default messaging provider** and click **OK**.
 - ___ e. Enter **counterQueue** in the “Name” field.
 - ___ f. Enter **jms/CounterQueue** in the “JNDI name” field.
 - ___ g. Select **bus1** from the “Bus name” menu.
 - ___ h. Select **SIBQueue** from the “Queue name” menu.
 - ___ i. Click **OK**.
- ___ 9. Create an activation specification.
- ___ a. From the already-expanded “JMS” sub-menu, click **Activation specifications**.
 - ___ b. Use the pop-up menu to select the server scope, **Node=<node-name>**, **Server=server1**.
 - ___ c. Click **New**.
 - ___ d. Select **Default messaging provider** and click **OK**.
 - ___ e. Enter **counterSpec** in the “Name” field.
 - ___ f. Enter **jms/counterSpec** in the “JNDI name” field.
 - ___ g. Enter **jms/counterQueue** in the “Desination JNDI name” field.
 - ___ h. Select **bus1** from the “Bus name” menu.
 - ___ i. Scroll to the bottom of the page and click **OK**.
- ___ 10. Save your changes to the master repository and restart the server for the changes to take effect.
- ___ a. Click **Save** in the “messages” box.
 - ___ b. Click **Logout** to log out of the administrative console.
 - ___ c. Stop the server using the command prompt window:

```
 startServer.bat server1 -username wsdemo
```

```
-password password
```

> Linux

```
./startServer.sh server1 -username wsdemo
```

```
-password password
```

___ d. Then start the server:

Windows

```
startServer.bat server1 -username wsdemo
```

```
-password password
```

> Linux

```
./startServer.sh server1 -username wsdemo
```

```
-password password
```

Part 2: Modify the sample application

In this section, you will be adding logic to the sample application to enable it to increment the counter by a variable number, provided by the user when submitting a web request, rather than always incrementing the counter by one.

- ___ 1. Add a new method, `incrementBy()` to the `StatelessCounterBean` class.
 - ___ a. Expand **EJB3Beans > src > com.ibm.websphere.ejb3sample.counter** in the Project Explorer pane.
 - ___ b. Double-click **StatelessCounterBean.java** to open it in the source code editor.
 - ___ c. Select **Go to line...** from the **Navigate** menu.
 - ___ d. Enter **50** and click **OK**. The text cursor should be placed immediately above the `getTheValue()` method.
 - ___ e. Enter the block of code shown below (you can also cut and paste it from **<LAB_FILES>\MDBLab\EJB3Beans.jar\com\ibm\websphere\ejb3sample\counter\StatelessCounterBean.java**)This code implements a new method, `incrementBy()`, that has the same logic as the existing `increment()` method, but takes a variable as the amount by which to increment the counter, instead of always incrementing the counter by one.

```
public int incrementBy(int k) {
    int result = 0;
    try {
        JPACounterEntity counter = em.find(JPACounterEntity.class,
CounterDBKey);

        if ( counter == null ) {
            counter = new JPACounterEntity();
            counter.setPrimaryKey(CounterDBKey);
            em.persist( counter );
        }

        counter.setValue( counter.getValue() + k );
        em.flush();
        em.clear();

        result = counter.getValue();
    } catch (Throwable t) {
        System.out.println("StatelessCounterBean:increment - caught
unexpected exception: " + t);
        t.printStackTrace();
    }
    return result;
}
```

- ___ f. **Save** and **close** `StatelessCounterBean.java`

___ 2. Add the `incrementBy()` method to the `LocalCounter` interface.

___ a. Double-click on **LocalCounter.java** in the Project Explorer pane.

___ b. Locate the line `public int getTheValue();` (line 11) and add the following line immediately below it.

```
public int incrementBy(int k);
```

___ 3. Add the `incrementBy()` method to the `RemoteCounter` interface.

___ a. Double-click on **RemoteCounter.java** in the Project Explorer pane.

___ b. Locate the line `public int getTheValue();` (line 11) and add the following line immediately below it.

```
public int incrementBy(int k);
```

___ 4. Add logic to the `EJBCount` Servlet that enables it to take a variable as input, and place the value onto a JMS queue for consumption by a message-driven bean.

___ a. Double-click on **EJBCount.java** in the Project Explorer pane.

___ b. Enter the following code above the `doGet()` method to inject instances of a JMS Queue and `ConnectionFactory` for use by the Servlet.

```
@Resource(name="counterQCF" )  
private ConnectionFactory counterQCF;
```

```
@Resource(name="counterQueue" )  
private Queue counterQ;
```

___ c. Select **Organize imports** from the **Source** menu.

___ d. You will be prompted to choose a type for the `ConnectionFactory` class. Select **javax.jms.ConnectionFactory** from the list and click **Next**.

___ e. You will be prompted to choose a type for the Queue class. Select **javax.jms.Queue** from the list and click **Finish**.

___ f. Replace the contents of the `doPost()` method (lines 37-54) with the following code. The code can also be cut and pasted from
<LAB_FILES>MDBLab\EJB3Beans.jar\com\ibm\websphere\ejb3sample\counter\EJBCount.java

```

public void doPost(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {

    String msg = null;
    int ejbCount = 0;
    Integer howMany = 0;
    String theCount = "0";

    try
    {
        theCount = req.getParameter("howmany");
        howMany = new Integer(theCount);
    }
    catch (NullPointerException npe)
    {
        howMany = 0;
        theCount = "0";
    }

    if(howMany > 0 && howMany < 99) {

        try{

            Connection conn = counterQCF.createConnection();
            conn.start();
            Session sess = conn.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            Message m = sess.createTextMessage(theCount);
            MessageProducer mp = sess.createProducer(counterQ);
            mp.send(m);
            msg = "Sent JMS message. <a
href=/ejb3sample/counter>Click here to see the resulting value</a>.";
            req.setAttribute("msg", msg);
            RequestDispatcher rd =
getServletContext().getRequestDispatcher("/EJBCount.jsp");
            rd.forward(req, res);
        }
        catch(Exception ex) {
            System.out.println("EJBCount: Failed to send JMS
message.");
            ex.printStackTrace();
        }
    }
    else{

        msg = "Error: you must enter a value between 1 and 99.";

        // Set attributes and dispatch the JSP.
        req.setAttribute("msg", msg);
        RequestDispatcher rd =
getServletContext().getRequestDispatcher("/EJBCount.jsp");
        rd.forward(req, res);
    }

}
}

```

- ___ g. Select **Organize imports** from the **Source** menu.
 - ___ h. You will be prompted to choose a type for the Connection class. Select **javax.jms.Connection** from the list and click **Next**.
 - ___ i. You will be prompted to choose a type for the MessageProducer class. Select **javax.jms.MessageProducer** from the list and click **Next**.
 - ___ j. You will be prompted to choose a type for the Message class. Select **javax.jms.Message** from the list and click **Next**.
 - ___ k. You will be prompted to choose a type for the Session class. Select **javax.jms.Session** from the list and click **Finish**.
 - ___ l. Compare the file to the provided example code in **<LAB_FILES>\MDBLab\EJB3Beans.jar\com\ibm\websphere\ejb3sample\counter\EJBCount.java** to ensure that you have made the right changes.
 - ___ m. **Save** and **close** EJBCount.java.
- ___ 5. Add a field to EJBCount.jsp to allow the input of a variable.
- ___ a. Expand **counter > WebContent** in the Project Explorer.
 - ___ b. Double-click EJBCount.jsp to open it in the source code editor.
 - ___ c. Locate the line that reads **Click on the Increment button to increment the count** and replace it with the following code.

Enter a value between 1 and 99, then click the Increment button to increment the count

**
**

A message will be placed on a JMS queue. A Message Driven Bean will consume that message, and use the Session Bean to increment the Entity counter by the specified amount.

**

**

How many: **<input type="text" size="2" maxlength="2" name="howmany">**

- ___ d. Compare the file to the provided example code in **<LAB_FILES>\MDBLab\counter.war\EJBCount.jsp** to ensure that you have made the right changes.
- ___ e. **Save** and **close** EJBCount.jsp

Part 3: Develop a message-driven bean

___ 1. Launch the Application Server Toolkit

___ a. Open a command prompt.

___ b. Navigate to the directory where you have installed the Application Server Toolkit

```
Windows cd c:\IBM\AST
```

```
> Linux cd /opt/IBM/AST
```

___ c. Start the AST and specify a workspace that you used in the previous exercise, when you configured the AST, using the paths shown below, or another path of your choice.

```
Windows ast.exe -data c:\LabFiles61EJB3\workspace
```

```
> Linux ./ast -data var/tmp/Labfiles61EJB3/workspace
```

___ 2. Create a new Java source file for the MDB class.

___ a. Ensure that you are in the J2EE perspective (“J2EE” will be in the title bar of the AST window). If not, use the **Window** menu to select **Open perspective > J2EE**.

___ b. Expand **EJB3Beans**, then expand **src** in the Project Explorer pane.

___ c. Right click on the package **com.ibm.websphere.ejb3sample.counter**, and select **New > Class** from the contextual menu.

___ d. Enter **EJB3CounterMDB** in the “Name” field.

___ e. Click **Add** next to the interfaces field, and add the interface **javax.jms.MessageListener**, by typing **MessageListener**, and then selecting **javax.jms.MessageListener** from the “matching types” field.

___ f. Click **OK**. The “New Java Class” window should look like the example below.



__ g. Click **Finish**.

___ 3. Annotate the source file to mark it as a message-driven bean.

__ a. Add the following code before the class definition:

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName="destinationName",propertyValue="counterQueue" ),
    @ActivationConfigProperty(propertyName="destinationType",propertyValue="javax.jms.Queue" )
})
```

___ 4. Inject an instance of the statelessCounter session bean for use by the MDB.

___ a. Locate the line that reads *public void onMessage(Message arg0) {* and add the following line immediately above it:

```
@EJB private LocalCounter statelessCounter;
```

___ 5. Add logic to the onMessage() method

Note: when a message is received by a message-driven bean, it is processed by the logic in the onMessage() method

___ a. Replace the line *// TODO Auto-generated method stub* with the code below.

```
int.ejbCount = 0;
int.howMany = 0;

try {
    TextMessage.txt = (TextMessage) arg0;
    howMany = new Integer(txt.getText());
    System.out.println("EJB3CounterMDB: message received: " +
txt.getText());
}
catch(Exception e) {
    System.out.println("EJB3CounterMDB: failed to read message");
    e.printStackTrace();
}

if(howMany > 0) {
    try{
       .ejbCount = statelessCounter.incrementBy(howMany);
        System.out.println("EJB3CounterMDB: incrementing counter by " +
howMany + ". new total is " +.ejbCount);
    }
    catch (RuntimeException re) {
        System.out.println("Error - increment() method on EJB
failed!");
        re.printStackTrace();
    }
}
```

___ b. Select **Organize imports** from the **Source** menu. The errors should disappear from your code when the import statements are added.

___ c. **Save** the file.

___ d. Your source code file should now look like the file **<LAB_FILES>\MDBLab\EJB3Beans.jar\com\ibm\websphere\ejb3sample\counter\EJB3CounterMDB.java**. You can use this file to verify correctness, or cut and paste from it into your Java editor instead of typing the above code manually.

___ e. **Save and close** EJB3CounterMDB.java

Part 4: Create bindings files

___ 1. Create `ibm-ejb-jar-bnd.xml` and populate it with binding information for the message-driven bean.

- ___ a. Expand **EJB3Beans** > **src** in the Project Explorer pane.
- ___ b. Right-click on **META-INF** and select **New > File**.
- ___ c. Name the file **ibm-ejb-jar-bnd.xml** and click **OK**.
- ___ d. Enter the following data, or cut and paste it from
<LAB_FILES>\MDBLab\EJB3Beans.jar\META-INF\ibm-ejb-jar-bnd.xml.

```
<ejb-jar-bnd
  xmlns="http://websphere.ibm.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee
  http://websphere.ibm.com/xml/ns/javaee/ibm-ejb-jar-bnd_1_0.xsd"
  version="1.0">

  <message-driven name="EJB3CounterMDB">
    <jca-adapter activation-spec-binding-name="jms/counterSpec"
      destination-binding-name="jms/counterQueue" />
  </message-driven>
</ejb-jar-bnd>
```

- ___ e. **Save** and **close** `ibm-ejb-jar-bnd.xml`.

___ 2. Populate the Web deployment descriptor with information about the queue and connection factory that the Servlet will need to use.

- ___ a. Expand **counter** > **Web Content** > **WEB-INF** in the Project Explorer pane.
- ___ b. Double-click **web.xml**.
- ___ c. Click the **References** tab in the editor pane.
- ___ d. Create a resource reference for the connection factory
 - 1) Click **Add...** under the "References" field.
 - 2) Select **Resource reference** and click **Next**.
 - 3) Enter **counterQCF** in the "Name" field.
 - 4) Select **javax.jms.ConnectionFactory** from the "Type" menu.
 - 5) Select **Container** from the "Authentication" menu.
 - 6) Click **Finish**.
 - 7) Enter **jms/counterQCF** in the "JNDI name" field under the "WebSphere Bindings" heading (near the bottom-right of the panel).
 - 8) **Save** `web.xml`.

___ e. Create a message destination reference for the queue.

- 1) Click **Add...** under the “References” field.
- 2) Select **Message destination reference** and click **Next**.
- 3) Enter **counterQueue** in the “Name” field.
- 4) Select **Link this reference to a message destination** and click **Add**.
- 5) Create a new destination named **jms/counterQueue** and click **OK**. (If you receive an error, click cancel. The necessary information has been entered.)
- 6) Click **Next**.
- 7) Select **javax.jms.Queue** from the “Type” menu.
- 8) Select **ConsumesProduces** from the “Usage” menu.
- 9) Click **Finish**.
- 10) Enter **jms/counterQueue** in the “JNDI name” field under the “WebSphere Bindings” heading (near the bottom-right of the panel).

___ f. **Save** and **close** web.xml

Part 5: Test the application

In this section you will test the sample application that you built and deployed.

- ___ 1. Uninstall the existing version of the sample application that you installed in the previous exercise. This application must be uninstalled, rather than updated, because it uses a local Derby database that is stored inside the installed application's directory.
 - ___ a. Click the **Servers** tab in the AST.
 - ___ b. Right-click the server name and select **Start** from the pop-up menu to start the server if it is not already started.
 - ___ c. Once the server has started, right-click the server name again and select **Add and remove projects**.
 - ___ d. Highlight **EJB3CounterSample** and click **Remove**.
 - ___ e. Click **Finish**.
 - ___ f. Right-click the server name and select **Stop** from the pop-up menu.
 - ___ g. Once the server has stopped, navigate to **<PROFILE_HOME>/installedApps** and delete the directory **EJB3CounterSample.ear**

Note: You must manually delete the directory, because the local Derby database file cannot be deleted while the Java process is running.

- ___ h. In the AST, right-click the server name again and select **Add and remove projects**.
 - ___ i. Highlight **EJB3CounterSample** and click **Add**.
 - ___ j. Right-click the server name and select **Publish**. This will start the server and deploy your application.
- ___ 2. Launch a Web browser and navigate to <http://localhost:9080/ejb3sample/counter>. (You may need to use a different port number, depending on your configuration.)
 - ___ 3. Verify that you see a page titled "EJB 3.0 and JPA 1.0 Counter Sample".
 - ___ 4. Enter a value in the "How many" field.
 - ___ 5. Click the **Increment** button to increment the counter using a JPA entity.
 - ___ 6. Click the resulting link to reload the page and see the new value of the counter.
 - ___ 7. Note in the console log displayed in the AST that you see, in addition to messages written by the Entity and by the session bean, you also see messages that are written by the AuditInterceptor class. This Interceptor was written to log the name of each method that is called. Inspect AuditInterceptor.java for details.

What you did in this exercise

In this exercise you configured the Application Server Toolkit to support development of EJB 3.0 modules. To do so, you created a server configuration to use a WebSphere Application Server with the Feature Pack for EJB 3.0 installed, and configured the AST to use the resources provided by the Feature Pack for EJB 3.0.

You also created a new enterprise application, and modified it to run under WebSphere Application Server V6.1 with the Feature Pack for EJB 3.0. You created the EJB module as a Utility JAR project to prevent the addition of a J2EE 1.4 EJB deployment descriptor, and then modified the application deployment descriptor to include this module as an EJB module.

You then deployed this application to WebSphere Application Server using the publishing function of the AST, and verified that the configuration and packaging were correct by running the sample application.

You are now ready to use the AST to modify and add to the sample application to further your EJB 3.0 learning experience.