Java CAPS 6
Using JCA, Note 6

JMS–Triggered JCA with Oracle and Batch Local File

Michael Czapski, July 2008

Table of Contents

1 Introduction

Rather then inventing an example to discuss and illustrate the use of the Oracle JCA Adapter let's build a solutions that uses the Oracle JCA Adapter and shows additional Java CAPS 6 facilities of interest.

Let's take the example from the "*Java CAPS Basics*: *Implementing Common EAI Patterns* Companion CD" book, ISBN: 0-13-713071-6, Chapter 11 "Scalability and Resilience", Section 11.2 "Exception Handling", subsection 11.2.1 "Exceptions in Java Collaborations", 11.2.1.1 "JMS-Triggered Java Collaborations". The book from which this section comes is available on the Companion CD. Let's re-work this example using Java CAPS 6 JCA Adapters.

2 Solution outline

This example illustrates exception handling involving a JMS-triggered JCA Message-Driven Bean.

The MDB is designed to receive a JMS message, update a database table row with the value of the text message, write the text message, together with the timestamp, to a file and finish.

When triggered, the MDB will log the invocation to the server.log. It will then attempt to update a record in a database table using table EMP in the default Oracle's sample schema SCOTT. Finally it will attempt to write a record to a file using the Batch Local File JCA Adapter. After each step, the MDB will have an opportunity to throw an exception. The input message will contain one of the literals enumerated in Table 2-1, each of which allows the MDB to execute a specific set of functionality before causing an exception.

Table 2-1 *Message literals for exercising execution paths*

S1	Throw exception after emitting a log message
S2	Throw exception after executing S1 followed by an update to a database table
S3	Throw an exception after executing S2 followed by write to a file

Any other literal will cause the MDB to complete without throwing an exception.

3 External Systems Preliminaries

Let's ensure we have a record to update in the EMP table. Listing 3-1 illustrates the commands used to ensure that the EMP table is ready for execution of the exercise, and their output.

Listing 3-1 *Clear and inspect EMP table in preparation for exercise execution*

```
SQL> delete from scott.emp where ename = 'czapski';

1 row deleted.

SQL>  insert into scott.emp (empno, ename, job, mgr, hiredate, sal,
comm) values (7777,'czapski','clerk',7777,'03/dec/81',1200,200);

1 row created.

SQL> commit;

Commit complete.

SQL> select * from scott.emp where ename='czapski';

     EMPNO ENAME   JOB          MGR HIREDATE       SAL     COMM  DEPTNO
---------- ------- --------- ----- --------- ------- ------- -------
      7777 czapski GOOD       7777 08/DEC/81    2000     102       10

SQL>
```

4 Connection Pools and JNDI Resources

Since the MDB will use a JMS Queue, an Oracle JCA Adapter, and Batch Local File JCA Adapter we must create and configure corresponding Connection Pools and related JNDI references.

For the JMS JCA Adapter we could create a JMS Queue qJMSTriggeredJCA under Resources -> Connectors -> Admin Object Resources, and the corresponding Queue qJMSTriggeredJCA_DLQ for undeliverable messages but we don't have to so we will not do this. The JMS JCA Adapter Wizard allows us to use JNDI references to queues or to use queue names directly. We will use queue names directly. You do as you think is best for you.

For the Oracle JCA Adapter we must create a Connection Pool under the Resources -> Connectors -> Connector Connection Pools. Let's call it "ora-lt-localhost-jcaps511-scott" to indicate that it will be an Oracle DB Connection Pool, it will be configured to support Local Transaction, the RDBMS instance is called jcaps511, is running on the localhost and the user schema will be SCOTT. The Connection Pool in Resources -> Connectors -> Connector Connection Pools does have provisions for configuring the host and the credentials. The corresponding Connection Pool under the CAPS -> Connector Connection Pools does. As we create the Resources -> Connectors -> Connector Connection Pools pool the corresponding CAPS -> Connector Connection Pools gets created automatically. In addition, we will require a JNDI Name corresponding to the connection pool. We will create it under the Resources -> Connectors -> Connector Resources, name is "jndi-ora-lt-localhost-jcaps511-scott", and configure it to point to the pool named "ora-lt-localhost-jcaps511-scott" pool.

Let's illustrate the steps.

Start the Application Server Admin Console and expand the Resources tree until the list of Connector Connection Pools is displayed in the right hand pane, Figures 4-1 through 4-3 illustrate major steps.

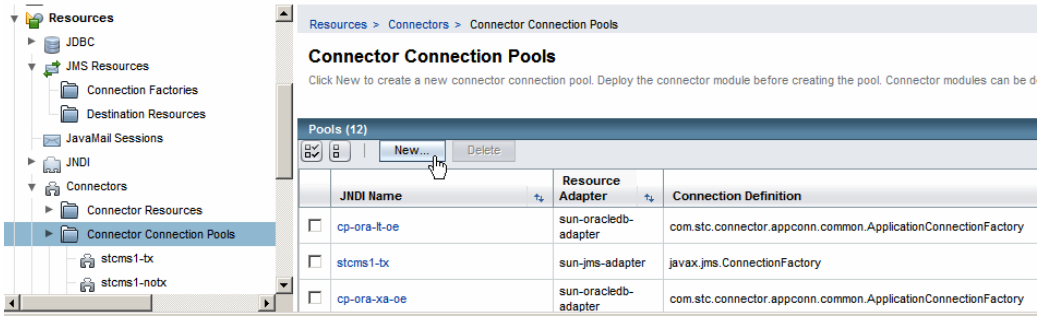

Figure 4-1 Start the New Connector Connection Pool creation process

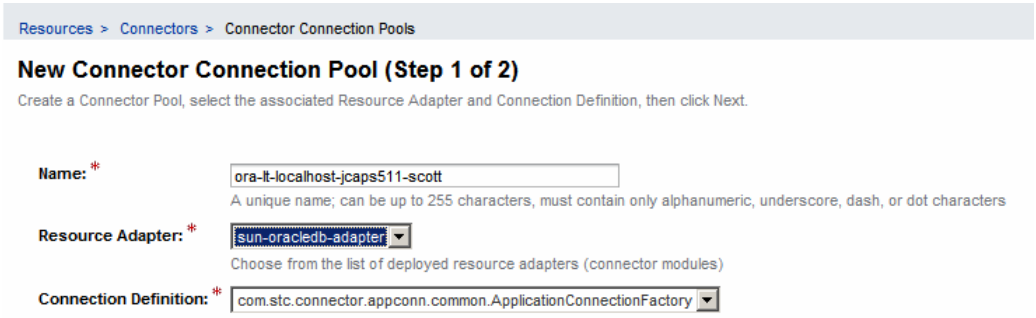

Figure 4-2 Name the pool, choose the adapter and the connection factory

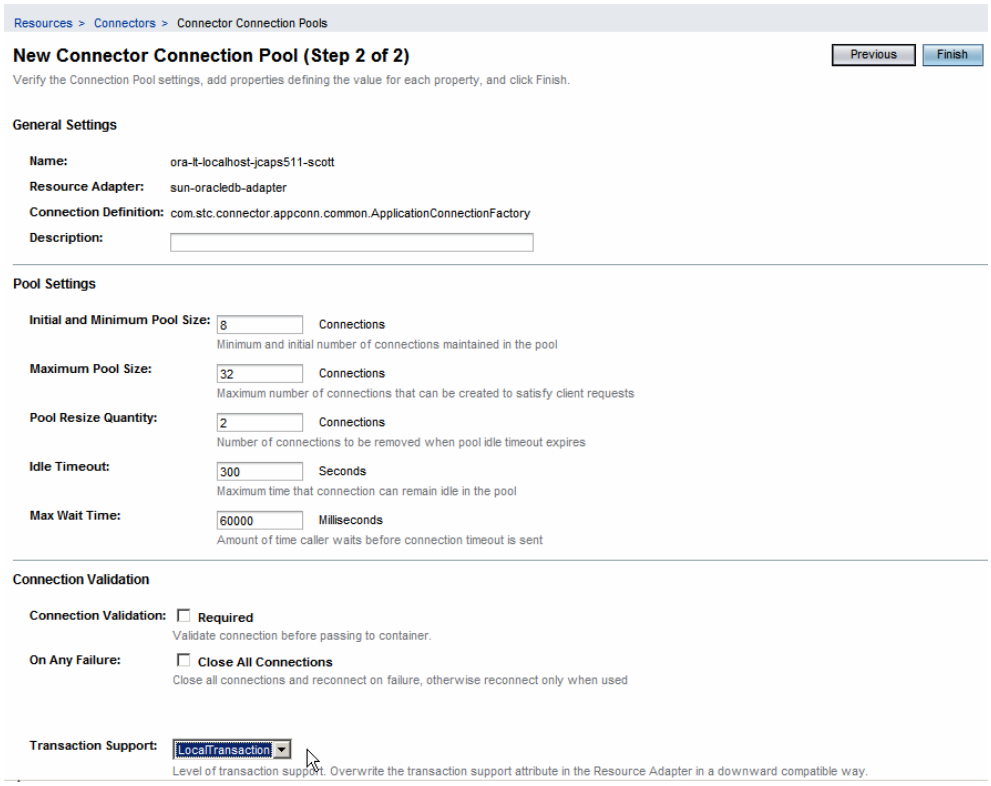

Figure 4-3 Choose LocalTransaction for Transaction Support and Finish

Let's now configure the database host, instance and credentials under the CAPS -> Connector Connection Pools. Figures 4-4 and 4-5 illustrate major steps.

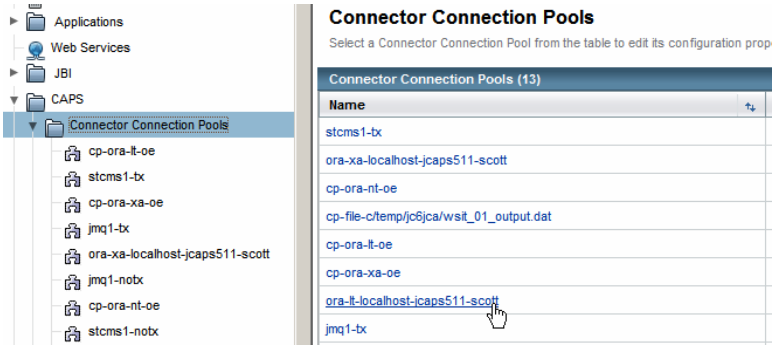

Figure 4-4 Click on the pool name

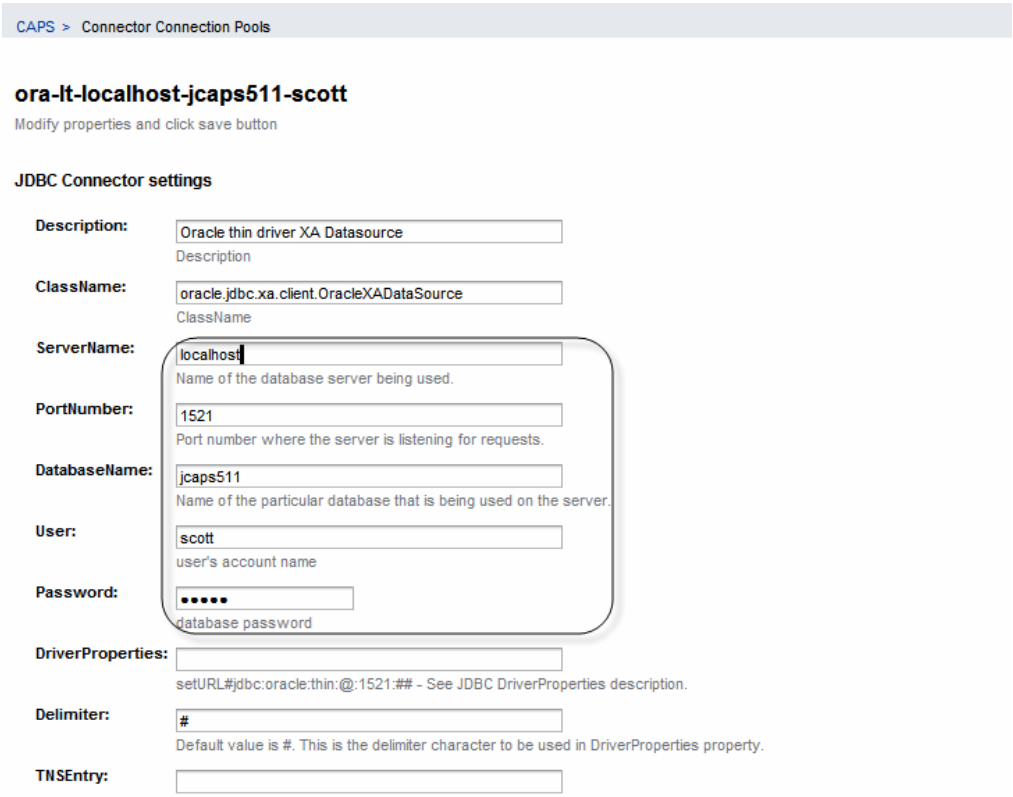

Figure 4-5 Configure host, port, SID and credentials, and Save

Finally, let's create the JNDI Name that will be provided to the JCA configuration Wizard in NetBeans. Figures 4-6 and 4-7 illustrate this.

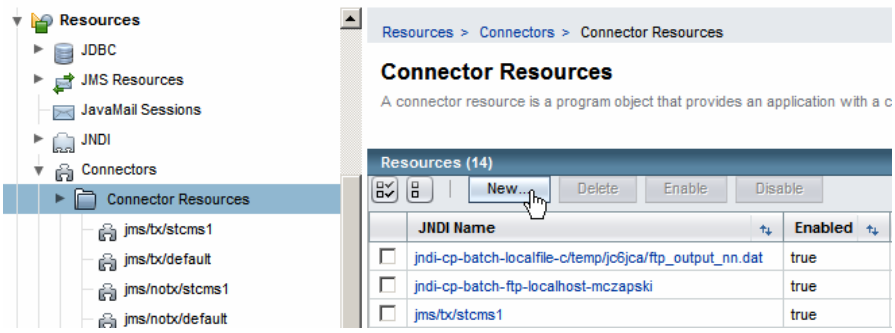

Figure 4-6 Create a new Connector Resource reference

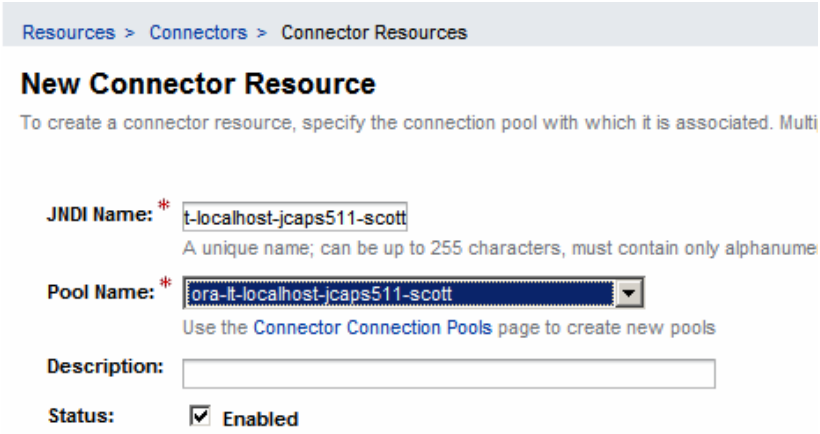

Figure 4-7 Name the JNDI reference and choose the correct pool

Pay close attention to the pool you are choosing. Alas, as it is at present, there may be a rather large list of connection pools in the drop down.

This gives us the connection pool and the JNDI reference to the connection pool we need for the oracle JCA Adapter we will be using. This connection pool resource can be reused in other projects that use the same database instance, on the same host, with the same credentials and that don't mind sharing the connection pool.

For the batch local file JCA Adapter we also must create a Connection Pool under the Resources -> Connectors -> Connector Connection Pools. Let's call it "BatchLocal-c/temp/jc6jca/jmstriggeredjca_nn.dat", to indicate that it will be Batch Local File Connection Pool, it will use a directory at c:/temp/jc6jca and the output file name will be jmstriggeredjca_%d.dat, where %d will be replaced by a serial number at runtime. The Connection Pool in Resources -> Connectors -> Connector Connection Pools does not have provisions for configuring the director and file name. The corresponding Connection Pool under the CAPS -> Connector Connection Pools does. As we create the Resources -> Connectors -> Connector Connection Pools pool the corresponding CAPS -> Connector Connection Pools gets created automatically. In addition, we will require a JNDI Name corresponding to the connection pool. We will create it under the Resources -> Connectors -> Connector Resources, name is "jndi-BatchLocal-c/temp/jc6jca/jmstriggeredjca_nn.dat", and configure it to point to the pool named "BatchLocal-c/temp/jc6jca/jmstriggeredjca_nn.dat".

Since the process is identical to that we went through for the Oracle JCA Adapter only key illustrations will be shown.

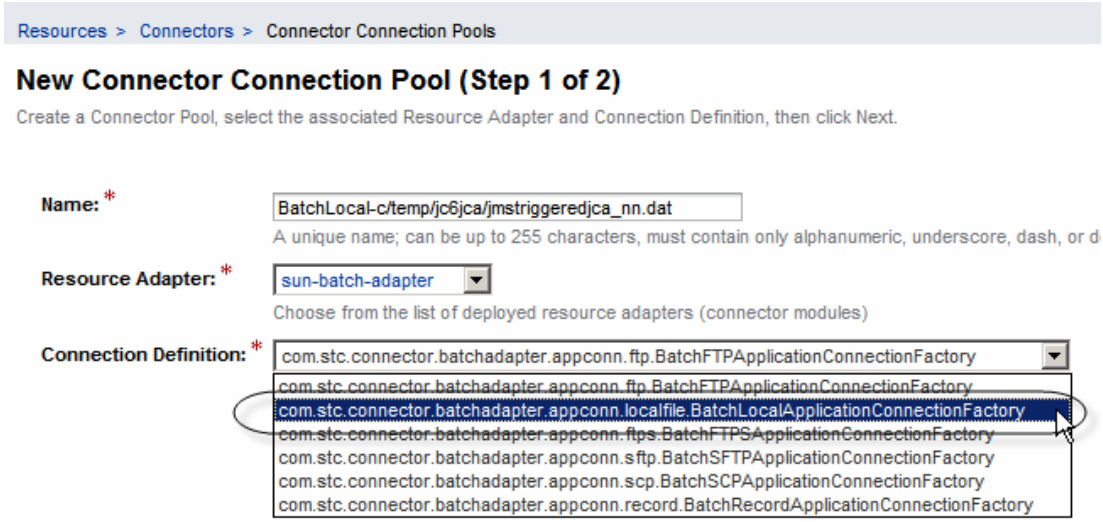

Figure 4-8 Name the new Connection Pool, choose the Adapter and the Connection factory

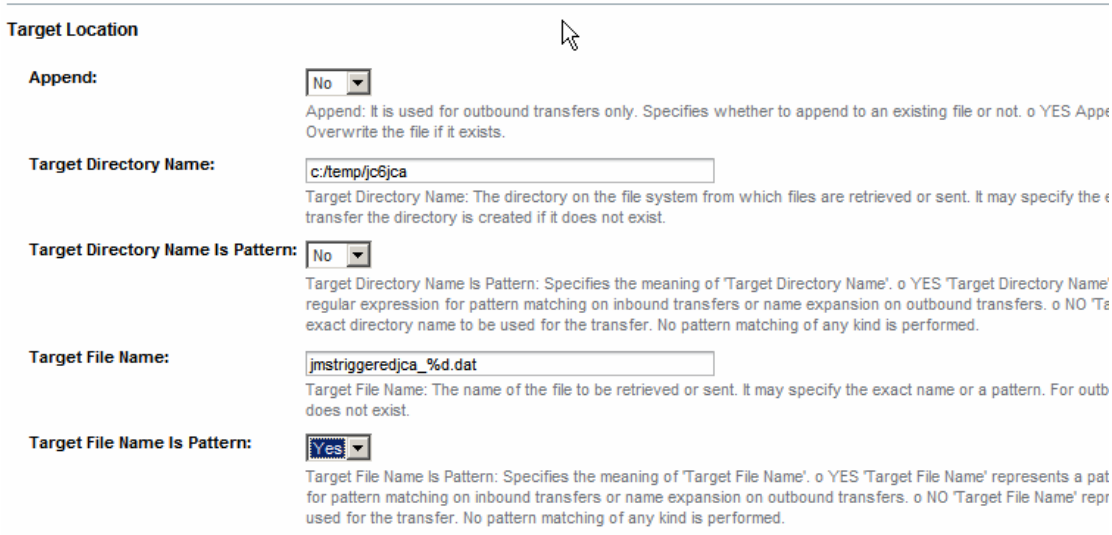

Figure 4-9 Configure target directory and file name

Make sure to set the Append property to “Yes”.

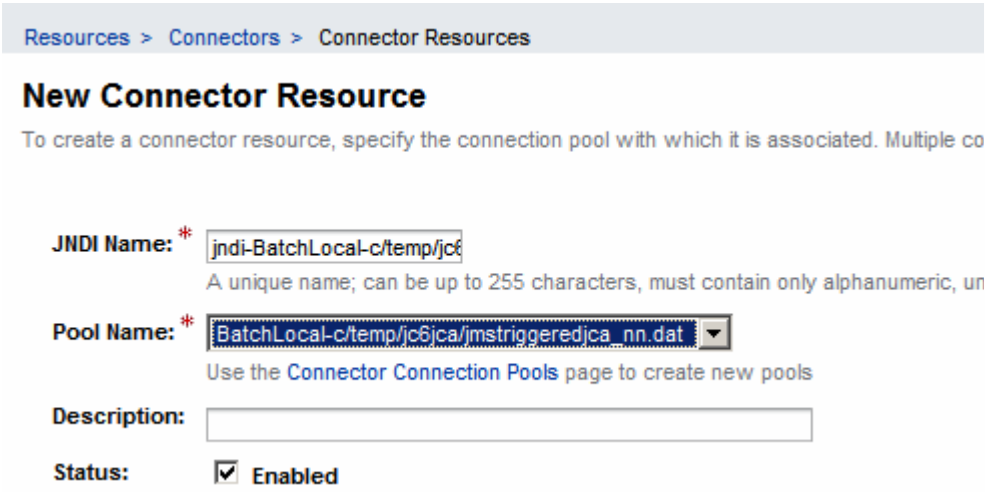

Figure 4-10 Create a new JNDI reference to the pool

This completes creation and configuration of all necessary connection pools and references.

5 Project Group and Project

As I am in a habit of doing, let's create a Project Group to contain the projects that will form part of this solution. Let's call this project group Scalability_and_Resilience_JMS-Triggered_JCA_MDB.

In the newly created project group let's create an Enterprise -> EJB Module project called jcaJMSTriggeredJCA_EJBM. In this project we will create all other artefacts. Figure 5-1 illustrates this.

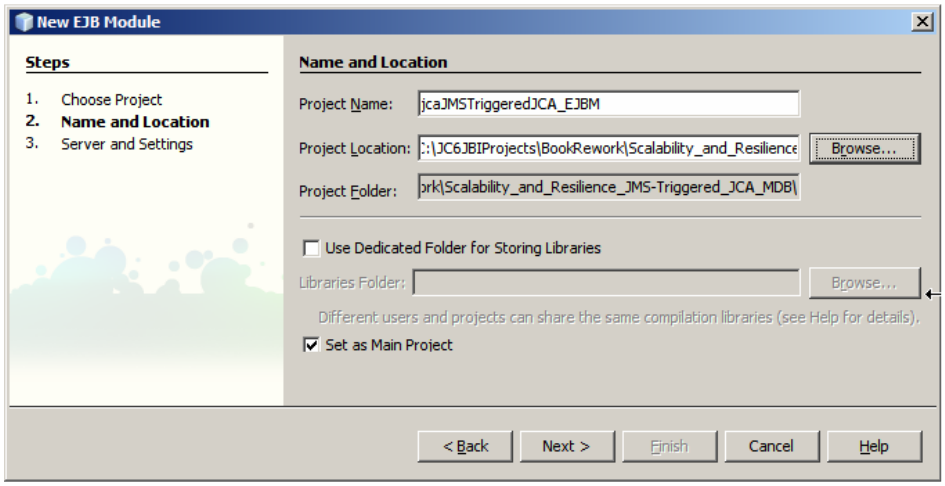

Figure 5-1 Create and name the Enterprise -> EJB Module project

6 Database OTD and the MDB Logic

To have the MDB update the record we will create an Oracle table OTD, tblEMP, for the SCOTT.EMP table. In Java CAPS 6 an Oracle OTD can be created two ways. It can be created in the repository-based project and imported into a JCA MDB project. It can also be created directly in the JCA MDB project. We will use the latter method.

Right-click on the name of the EJB Module project, jcaJMSTriggeredJCA_EJBM, choose New -> Other …, choose SOA -> Oracle Otd Wizard and follow the process as instructed. Figures 6-1 illustrates the step at this point.

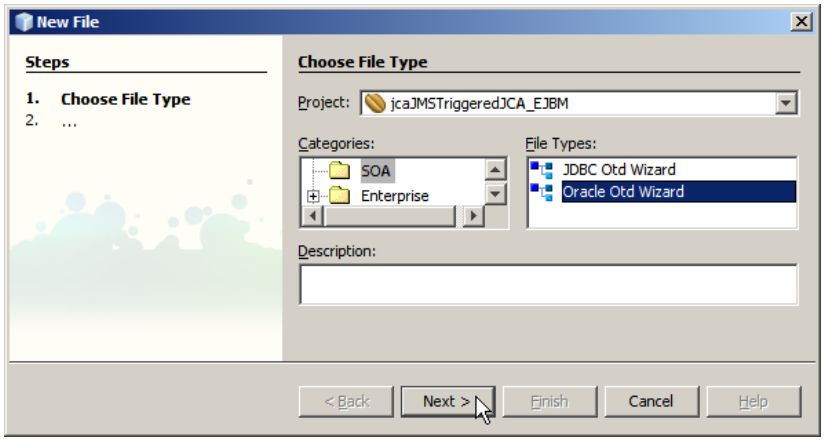

Figure 6-1 Choose SOA -> Oracle Otd Wizard

The Oracle Otd Wizard is very similar to its 5.1 equivalent. Figures 6-2 through 6-12 illustrate the steps involved in creating an Oracle Table-based OTD called otdSCOTT_EMP and its corresponding XML Schema definition.

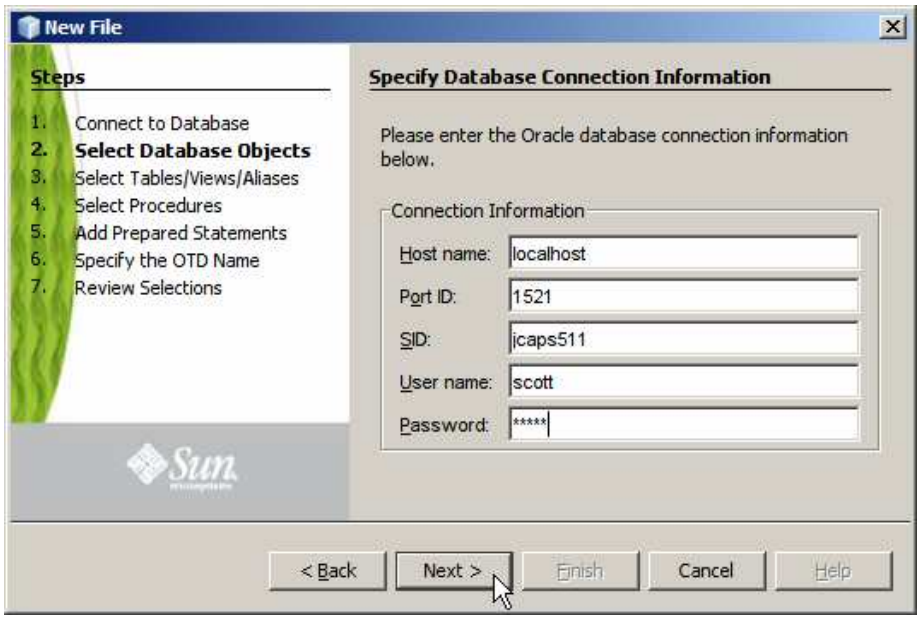

Figure 6-2 Provide Database details

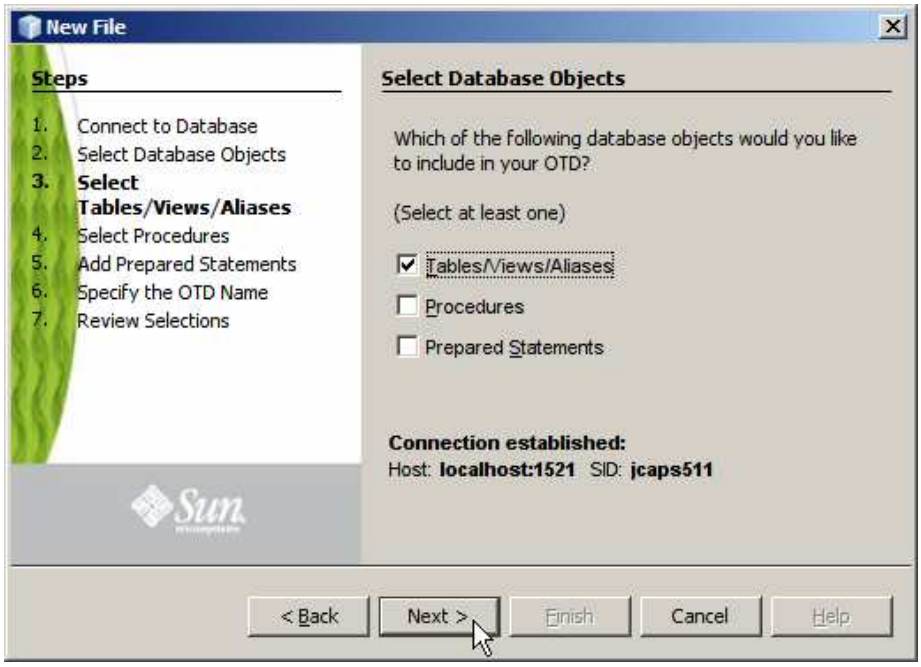

Figure 6-3 Choose tables/views/aliases

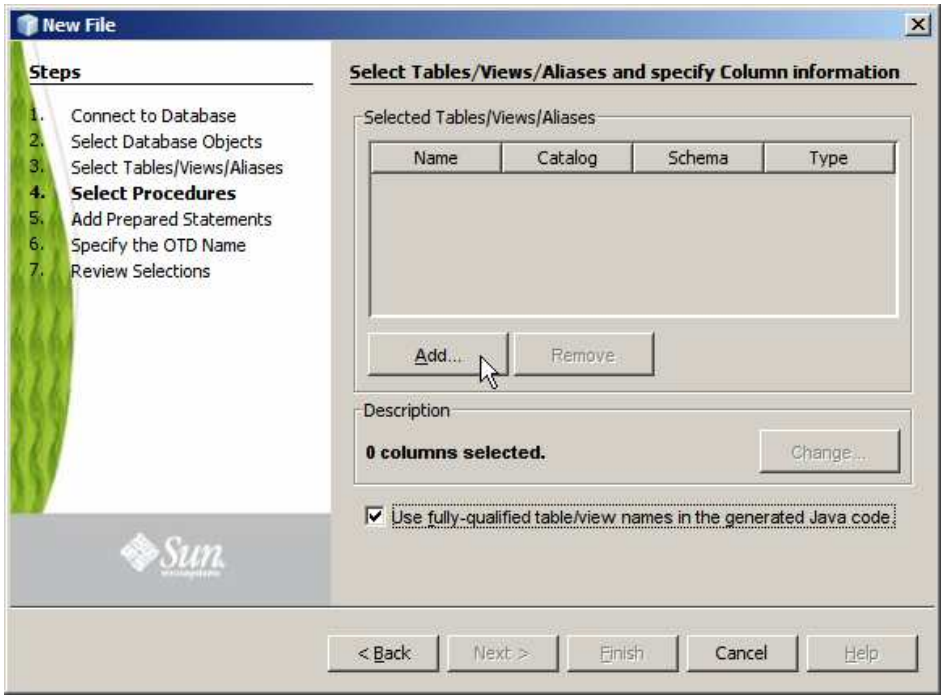

Figure 6-4 Choose to use fully qualified names and click Add …

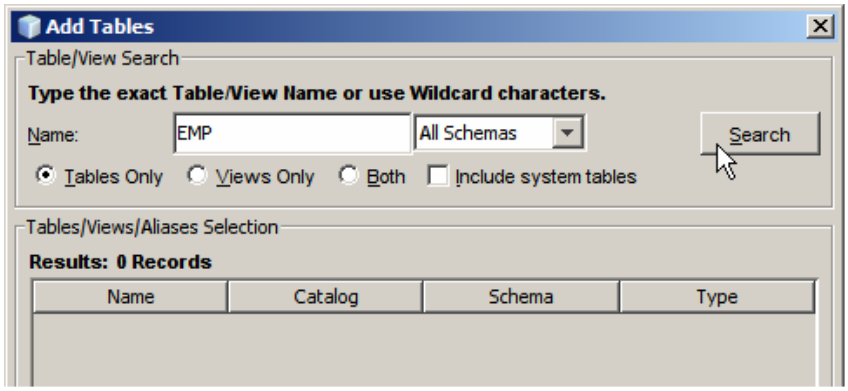

Figure 6-5 Type EMP and click Search

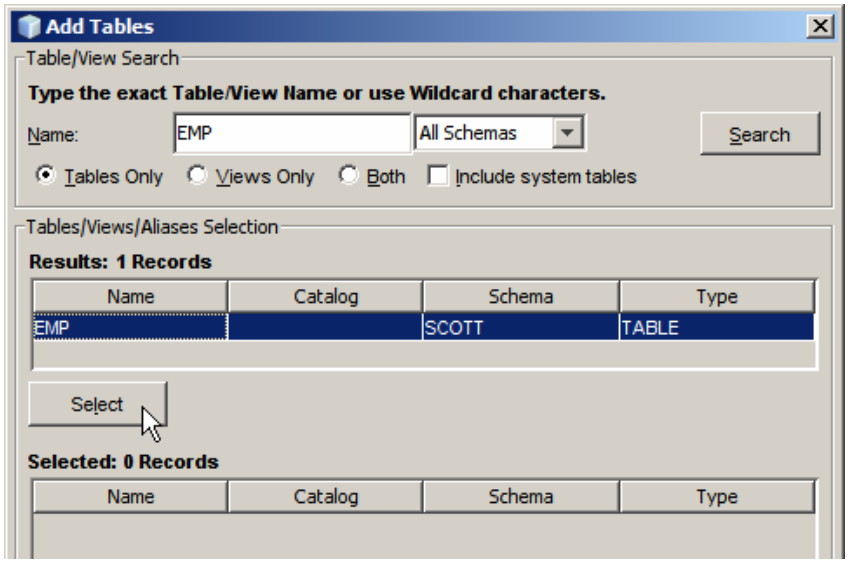

Figure 6-6 Select the table and click Select …

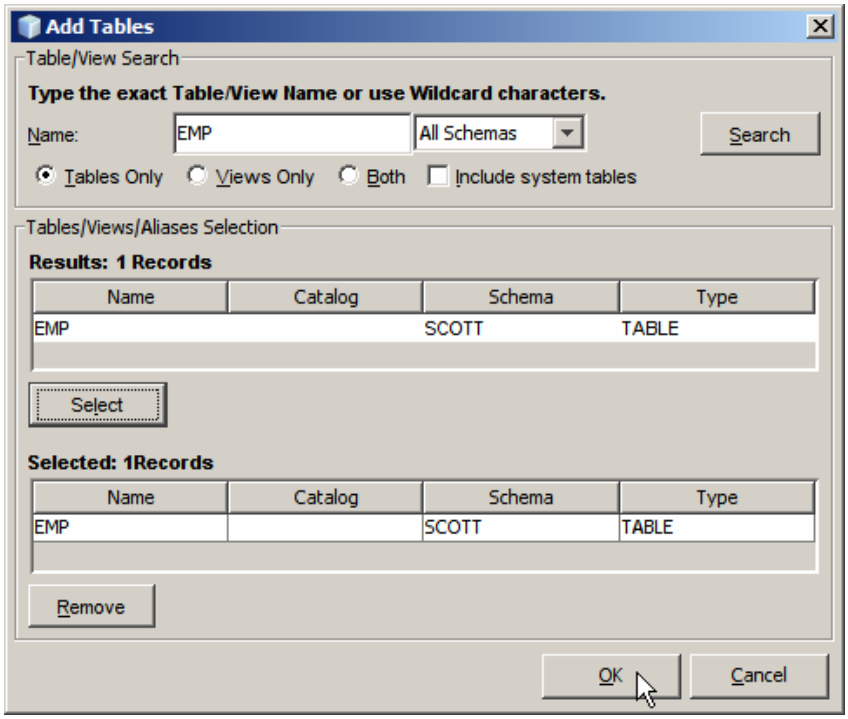

Figure 6-7 Click OK

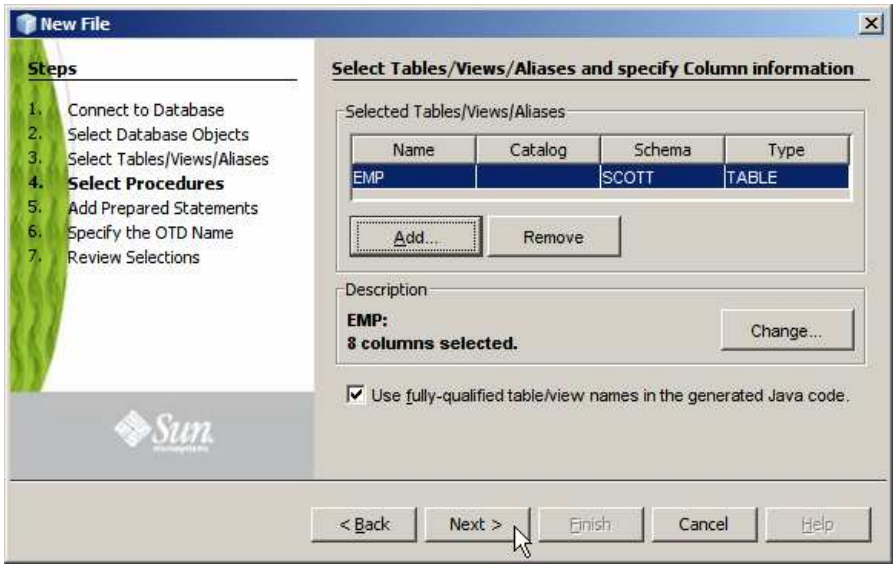

Figure 6-8 Click Next

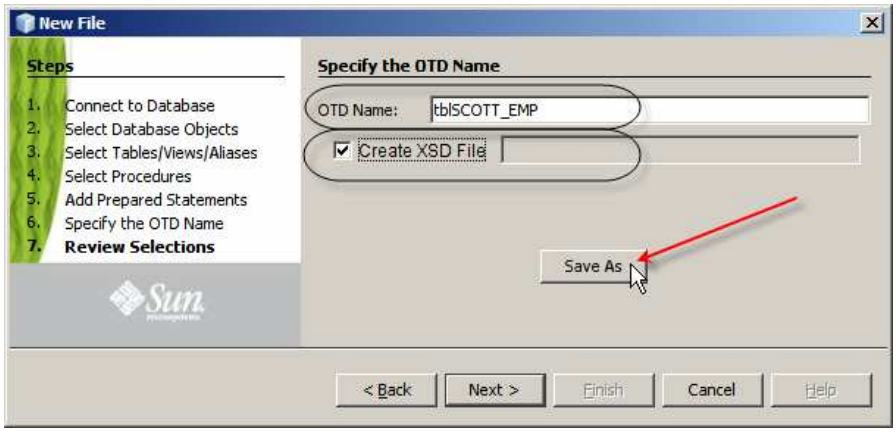

Figure 6-9 Enter OTD name, check Create XSD and click Save As

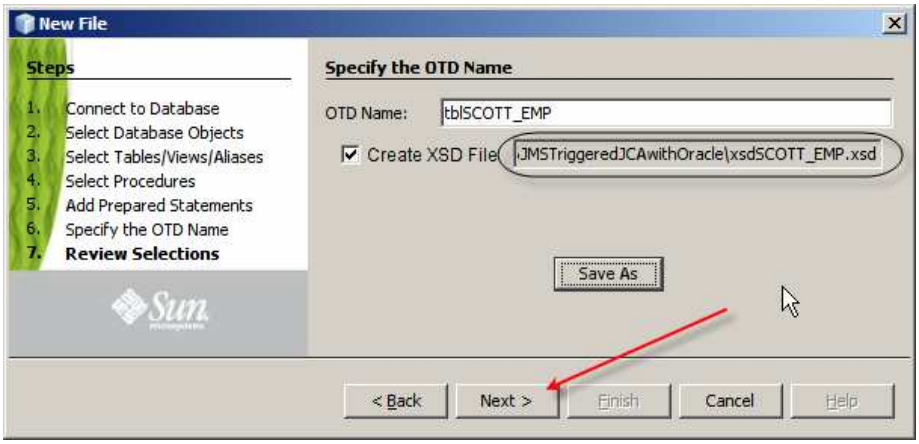

Figure 6-10 Once you located the directory to which to write the XSD and named it click Next

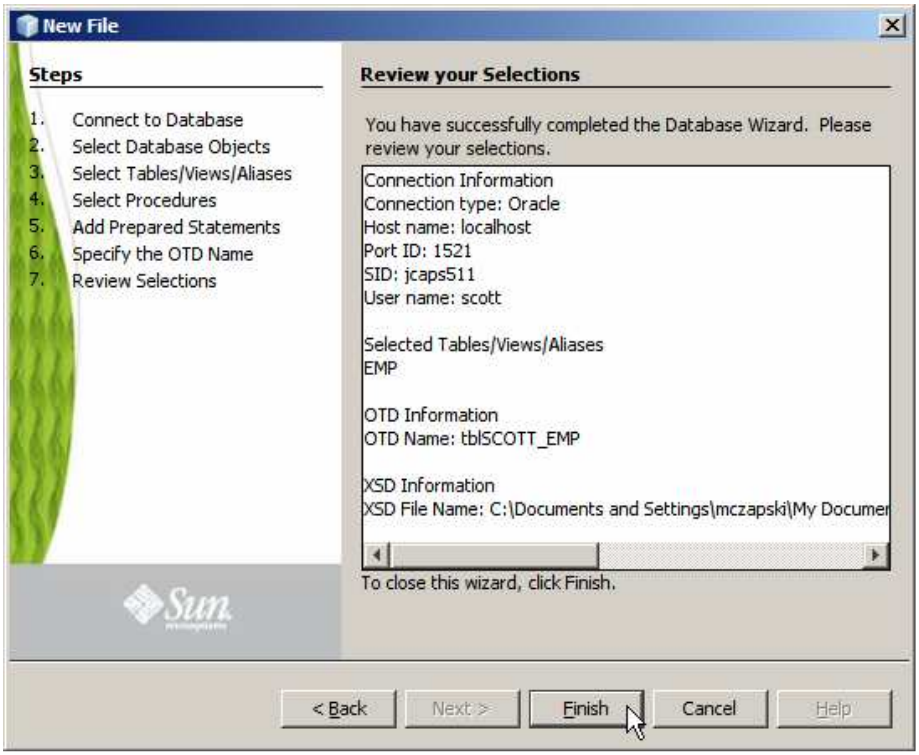

Figure 6-11 Click Finish and wait for the OTD to be created

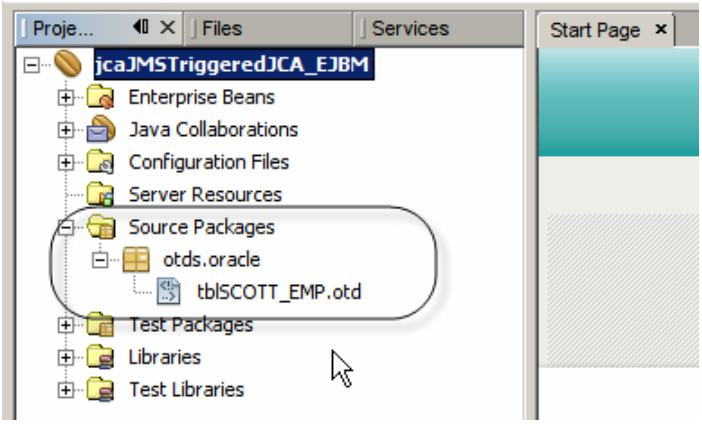

Figure 6-12 Notice one created in the Source Packaged folder

With the preliminaries over we can proceed to create the JCA Message-Driven Bean itself.

The Message Driven Bean, jcaJMSTriggeredJCA, shown in Listing 6-1, is triggered by the JMS JCA Adapter and uses an Oracle JCA Adapter with a table OTD, tblEMP, and a Batch Local File JCA Adapter. We will create this JCA MDB a step at a time, with illustrations following the listing.

Listing 6-1 *jcaJMSTriggeredJCA MDB receive method source*

```
public void receive
    (com.stc.connectors.jms.Message input
    ,tblEMP.TblEMPOTD U_tblEMP
    ,com.stc.eways.batchext.BatchLocal W_BatchLocalFile )
        throws Throwable
{
    logger.fine( "\n===>>> Entered jcdJMSTriggeredJCD" );

    if (input.getTextMessage().equalsIgnoreCase( "S1" )) {
        String sMsg = "Throwing exception on S1";
        logger.fine( "\n===>>> " + sMsg );
        throw new Exception( sMsg );
    }

    U_tblEMP.getEMP().update( "ENAME = 'czapski'" );
    logger. fine ( "\n===>>> Did select" );


    boolean blHavNext = U_tblEMP.getEMP().next();
    U_tblEMP.getEMP().setJOB( input.getTextMessage() );
    U_tblEMP.getEMP().updateRow();
    logger. fine ( "\n===>>> After DB Update" );

    if (input.getTextMessage().equalsIgnoreCase( "S2" )) {
        String sMsg = "Throwing exception on S2 after DB Update";
        logger. fine ( "\n===>>> " + sMsg );
        throw new Exception( sMsg );
    }

    String sTimestamp = "" + (new java.util.Date()).getTime();
    String sPayload = sTimestamp + ":" + input.getTextMessage() + "\n";
    W_BatchLocalFile.getClient().setPayload( sPayload.getBytes() );
    W_BatchLocalFile.getClient().put();
    logger. fine ( "\n===>>> After File PUT" );

    if (input.getTextMessage().equalsIgnoreCase( "S3" )) {
        String sMsg =
             "Throwing exception on S3 after DB Update and File Write";
        logger. fine ( "\n===>>> " + sMsg );
        throw new Exception( sMsg );
    }

    logger. fine ( "\n===>>> Exiting normally with trigger "
        + input.getTextMessage() );
}
```

Let’s start by creating a JCA MDB, jcaJMSTriggeredJCA, as shown in Figures 6-13 through 6-17.

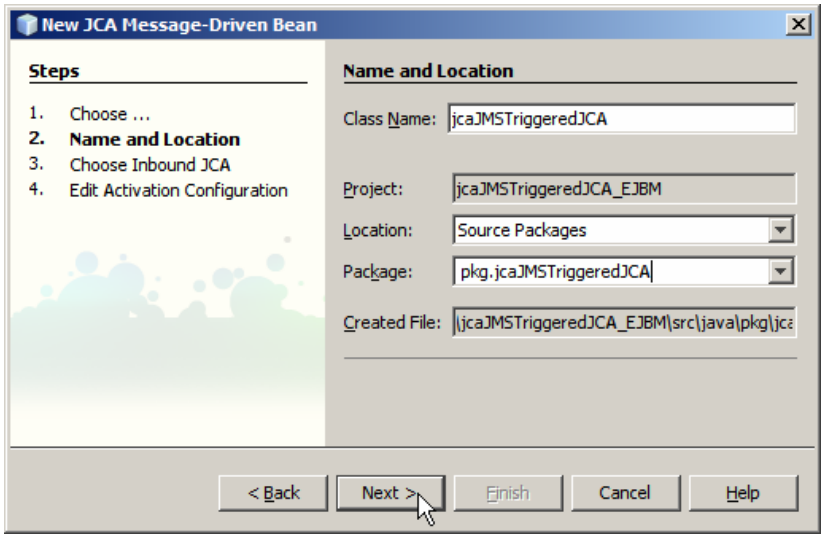

Figure 6-13 Name the JCA MDB

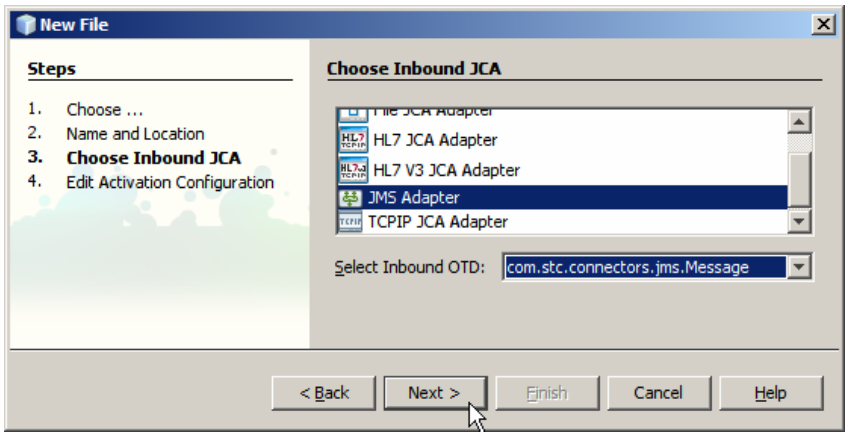

Figure 6-14 Choose the JMS Adapter and the com.stc.connectors.jms.Message message

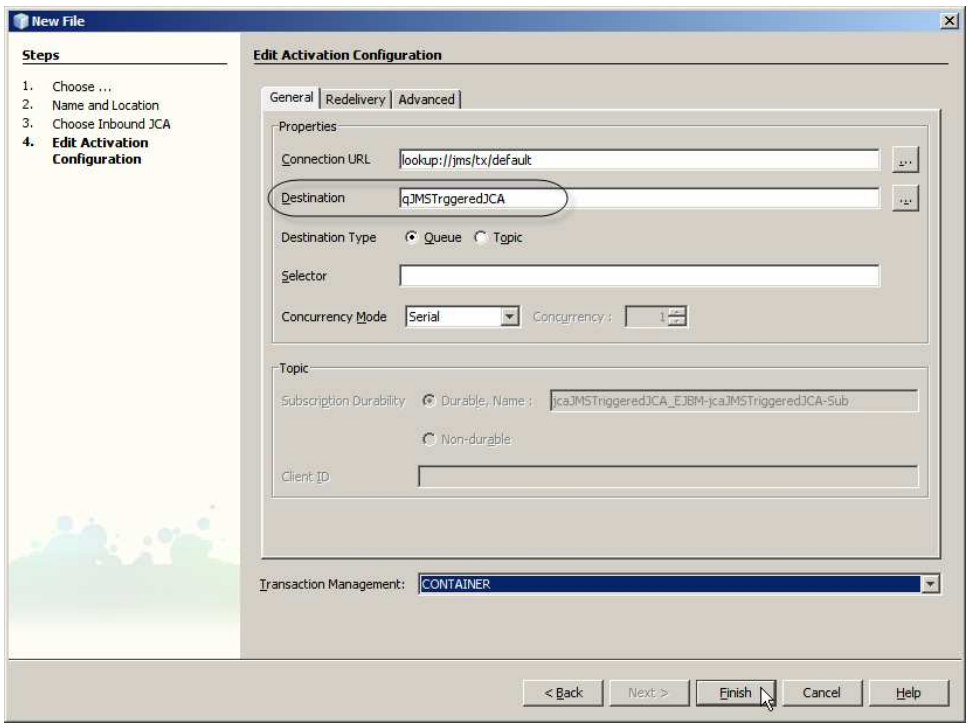

Figure 6-15 Name the Destination

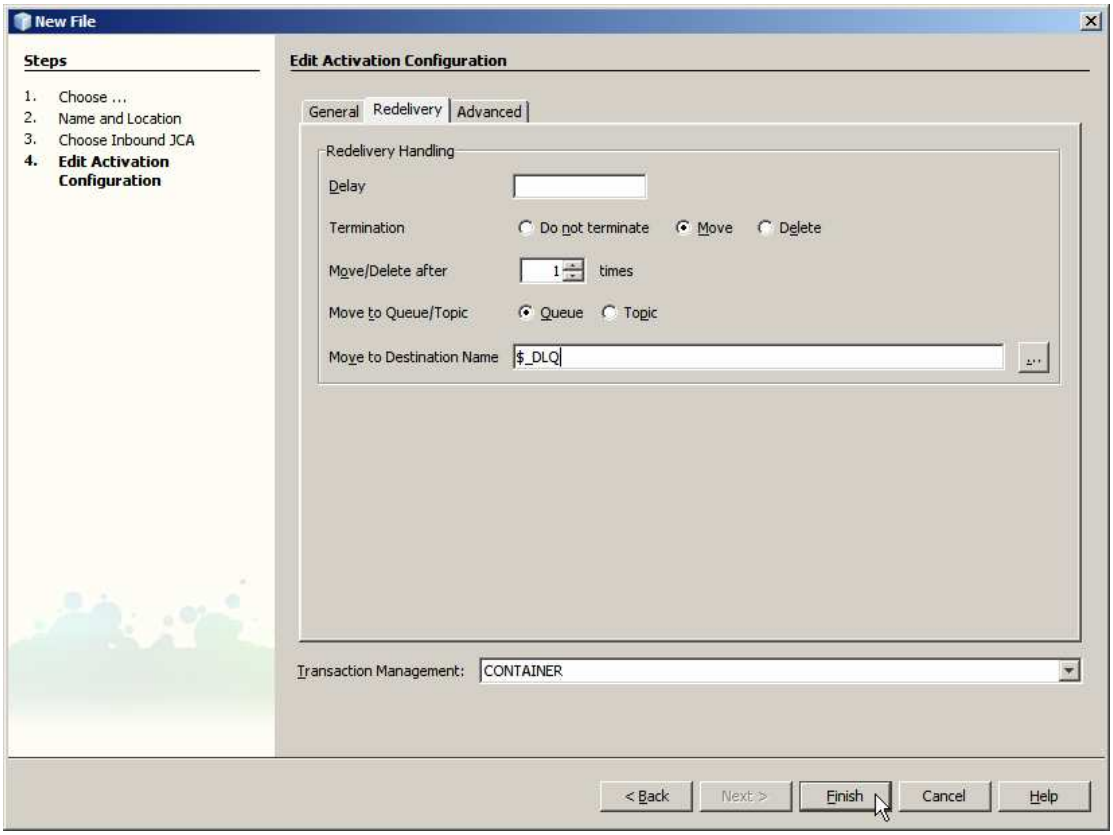

Figure 6-16 Configure redelivery handling to have the message moved to DLQ on failure

```
23  @MessageDriven(name="pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA")
24  @TransactionManagement(value=TransactionManagementType.CONTAINER)
25  public class jcaJMSTriggeredJCA implements MessageListener {
26
27      private static final Logger logger = Logger.getLogger(jcaJMSTriggeredJCA.class.getName
28
29      public jcaJMSTriggeredJCA() {
30      }
31
32      /**
33       * Invoked by JCA when a message is received
34       *
35       * @param message the message passed to the listener
36       */
        public void onMessage(Message message) {
38          try {
39              com.stc.connectors.jms.Message jmsOtdMessage = com.stc.connectors.jms.Message.
40              _invoke_receive(jmsOtdMessage);
41          } catch (java.lang.Throwable t) {
42              ectx.setRollbackOnly();
43              logger.log(Level.WARNING, "Failed to invoke _invoke_receive: " + t, t);
44              return;
45          }
46      }
47
48      private void receive(com.stc.connectors.jms.Message jmsOtdMessage) throws java.lang.Ex
49
50      }
51
```

Figure 6-17 Boilerplate JCA MDB code

Once the wizard completes the JCA MDB code will be available for editing – see Figure 6-17. Notice the receive method with a single argument of type com.stc.connectors.jms.Message, the type we selected when configuring the JCA Adapter through the wizard, named “jmsOtdMessage”. Let’s rename this argument to “input”. Once

we do this the signature of the receive method will be identical to that which one would see in a JMS-triggered Java Collaboration Definition in Java CAPS 5.x.

Let's add the Oracle JCA Adapter. Drag the Oracle JCA from the palette to the source window, as shown in Figure 6-18. It does not matter where in the source window one completes the drag action.

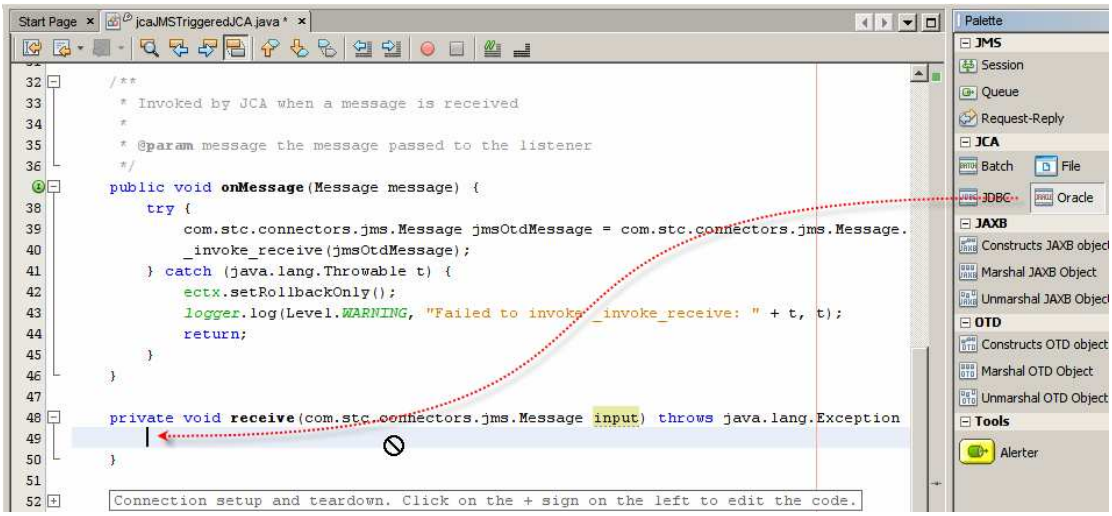

Figure 6-18 Add the Oracle JCA Adapter – begin configuration

From the beginning of this section recall creating the Oracle Table-based OTD, tblSCOTT_EMP. The Orale JCA configuration wizard requires us to specify the OTD which to use. Choose the once created earlier, as shown in Figure 6-19, and click Next.

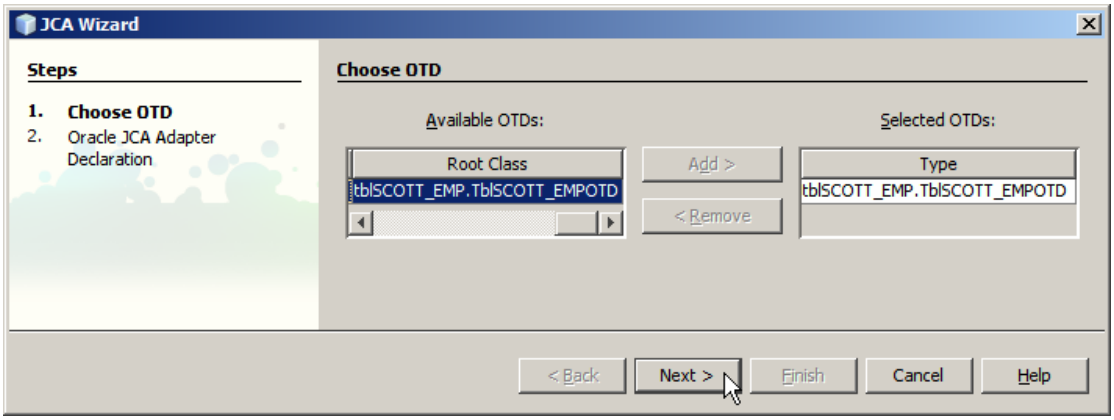

Figure 6-19 Choose tblSCOTT_EMP OTD

Accept the method name, receive, choose the JNDI reference to the Oracle Connection Pool, jndi-ora-lt-localhost-jcaps511-scott, which was created earlier, name the Local Variable U_tblEMP and click Finish. Figure 6-20 illustrates this.

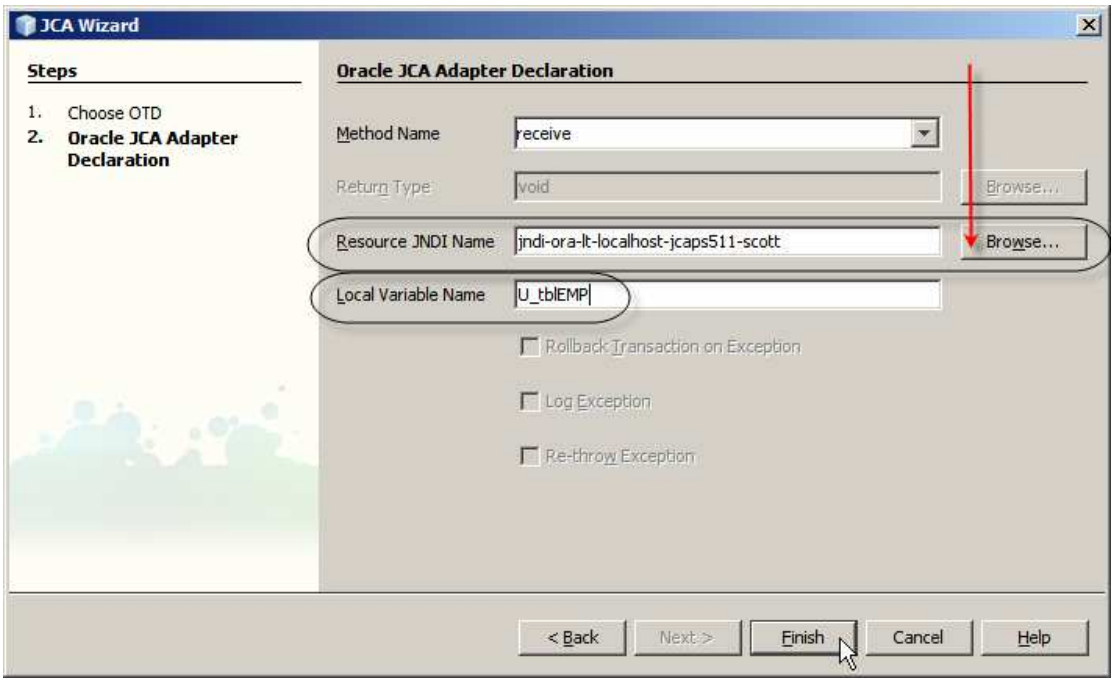

Figure 6-20 Configure Oracle JCA Adapter

To keep the variable names the same as in the code in the book example let's rename the Oracle OTD name, which was mangled by the wizard, from U_tblEMPOTD to U_tblEMP, as we intended all along. The receive method signature now looks like that shown in Figure 6-21. Note U_tblEMP, as renamed from the wizrd-provided U_tblEMPOTD.

```
48      private void receive
49              (com.stc.connectors.jms.Message input
50              ,tblSCOTT_EMP.TblSCOTT_EMPOTD U_tblEMP)
51                  throws java.lang.Exception {
```

Figure 6-21 receive method signature with JMS and Oracle OTD arguments

Let's now add the Batch JCA Adapter, making sure to rename the argument in the recveive method signature from

Figures 6-22 through 6-24 illustrate the process.

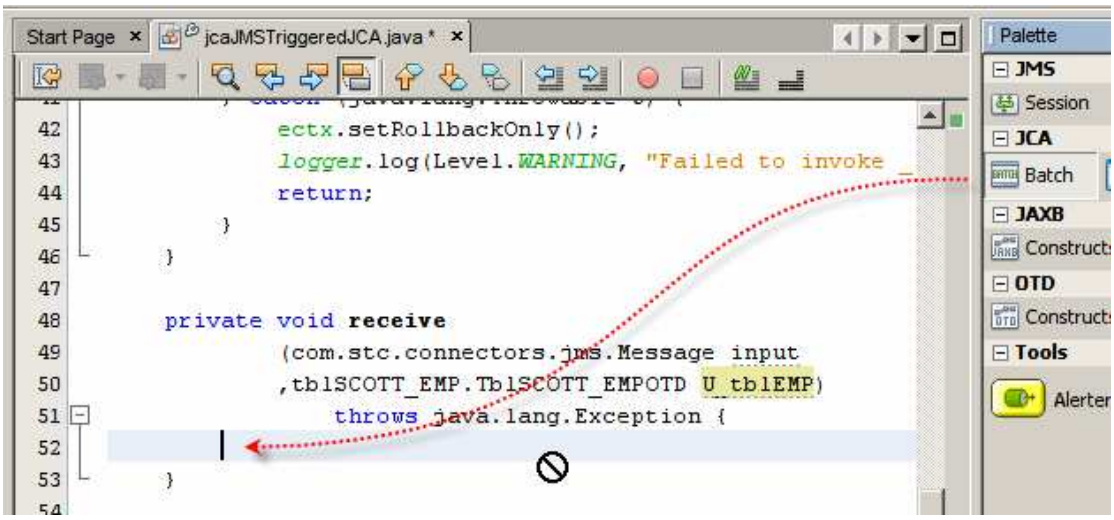

Figure 6-22Drag the Batch JCA Adapter to the source window

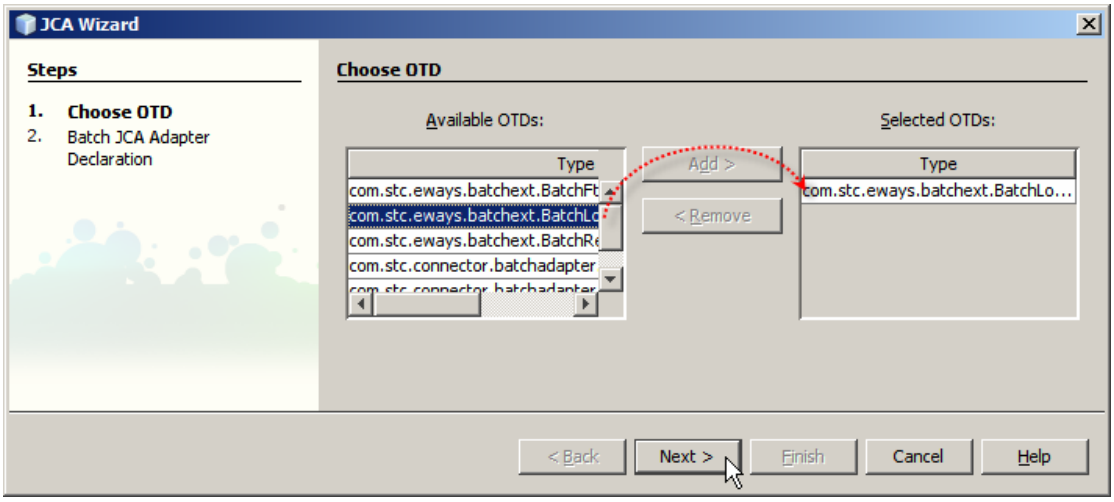

Figure 6-23 Choose the BatchLocal OTD and click Next

Figure 6-24 Accept method name, choose JNDI name and provide name for the variable

Rename the wizard-provided argument name, W_BatchLocalFileOTD, to W_BatchLocalFile, as we intended. You can keep the name but if you do the code in Listing 6-1 will have to be modified to use the new variable name. The receive method signature, after re-formatting, now looks like that shown in Figure 6-25.

```
48      private void receive
49              (com.stc.connectors.jms.Message input
50              ,tblSCOTT_EMP.TblSCOTT_EMPOTD U_tblEMP
51              ,com.stc.eways.batchext.BatchLocal W_BatchLocalFileOTD)
52                  throws java.lang.Exception {
53
54      }
```

Figure 6-25 receive method signature with all JCA Adapters included

To complete the MDB let's add the slab of code from the method body in Listing 6-1 as the receive method body. If you are transcribing Java CAPS 5.x code verbatim logger.debug(…)

and similar statements will be flagged as errors. This is because JCA MDBs use java.uril.logging, rather then the jog4j method names, which were jog4j in ICAN 5.0 and were emulated for compatibility in Java CAPS 5.1. Rename all occurrences of logger.debug to logger.fine and logger.error to logger.sever.

Build and deploy the project.

If you are interested in seeing what the MDB does at runtime enable verbose logging for selected logger categories. For example set the following using the Application Server Admin Console: Application Server -> Logging -> Log Levels, see Figures 6-26.

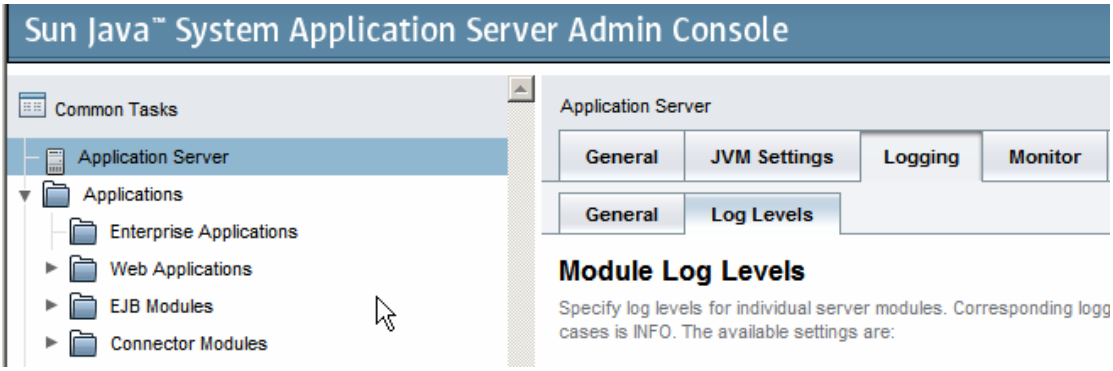

Figure 6-26Locating logging configuration

jcaJMSTriggeredJCA	FINEST
STC.eWay.batch	FINEST
STC.eWay.DB.Oracle	FINEST

If needs be, add logging properties with these names and values.

7 Exercise the solution

We will use the Enterprise Manager to inject messages into the solution to exercise different logic paths. Our starting point is a database table with the record for ENAME = ‘czapski’ containing the value “clerk” in the JOB column and the directory with no output file. Listing 7-1 illustrates the SQL command and its output.

Listing 7-1 *Select specific row from the EMP table*

```
SQL> select * from scott.emp where ename='czapski';

     EMPNO ENAME    JOB       MGR HIREDATE       SAL     COMM     DEPTNO
---------- -------- ------ ----- --------- ------- ------- ----------
      7777 czapski  clerk   7777 03/DEC/81     200      200         10
```

Let’s first exercise the “happy path” by submitting a message with the contents “AA”. Figure 7-1 highlights notable points in the Enterprise Manager display that may assist in manually submitting a message to a JMS queue. This message will not trigger an exception.

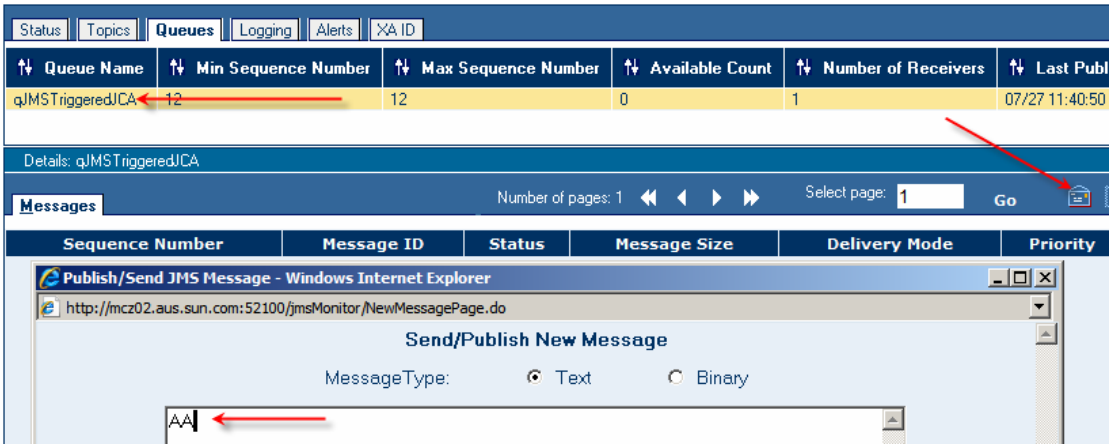

Figure 7-1 *Manually submitting a message to a JMS queue*

Once the MDB executes, the database table will be updated and the file with the timestamped entry will be created. Listing 7-1 illustrates the content of the table row after the update. Figure 7-2 illustrates the content of the file after execution of the project.

Listing 7-1 *Updated EMP table*

```
SQL> select * from scott.emp where ename='czapski';

     EMPNO ENAME      JOB   MGR HIREDATE       SAL      COMM      DEPTNO
---------- ---------- --- ----- --------- ------- ------- ----------
      7777 czapski    AA   7777 03/DEC/81     200     200          10
```

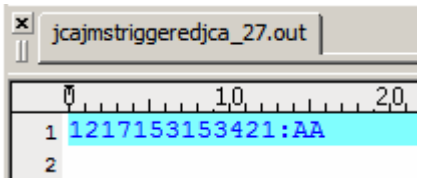

Figure 7-2 *Content of the file after execution of the project*

Let's now submit a message containing the literal "S1". The MDB will throw an exception before the logic gets to database update and file write. Given that the JMS Adapter is configured to try at most once then move the message to a Dead Letter Queue, we will see one attempt at MDB execution. The expectation is that neither the database update nor the file write will be executed, so there will be no changes in either resource.

The server.log shows an exception with the message being moved to a Dead Letter Queue. The exception messages are shown in Listing 7-2.

Listing 7-2 *Exception messages after submission of "S1" as a message*

```
[#|2008-07-27T20:09:25.046+1000|WARNING|sun-appserver9.1|pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA|_ThreadID=28;_Threa
dName=JMS Async S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;_RequestID=5c55478c-0490-4f67-b4a3-fb5dc998a0
34;|Failed to invoke _invoke_receive: java.lang.Exception: Throwing exception on S1
java.lang.Exception: Throwing exception on S1
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA.receive(jcaJMSTriggeredJCA.java:59)
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA._invoke_receive(jcaJMSTriggeredJCA.java:110)
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA.onMessage(jcaJMSTriggeredJCA.java:40)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at com.sun.enterprise.security.application.EJBSecurityManager.runMethod(EJBSecurityManager.java:1067)
        at com.sun.enterprise.security.SecurityUtil.invoke(SecurityUtil.java:176)
        at com.sun.ejb.containers.BaseContainer.invokeTargetBeanMethod(BaseContainer.java:2899)
        at com.sun.ejb.containers.BaseContainer.intercept(BaseContainer.java:3990)
        at com.sun.ejb.containers.MessageBeanContainer.deliverMessage(MessageBeanContainer.java:1111)
        at com.sun.ejb.containers.MessageBeanListenerImpl.deliverMessage(MessageBeanListenerImpl.java:74)
        at com.sun.enterprise.connectors.inflow.MessageEndpointInvocationHandler.invoke(MessageEndpointInvocationHandl
er.java:179)
        at $Proxy64.onMessage(Unknown Source)
        at com.stc.jmsjca.core.Delivery.deliverToEndpoint(Delivery.java:1075)
        at com.stc.jmsjca.core.SerialDelivery.onMessage(SerialDelivery.java:246)
        at com.stc.jms.client.SessionImpl.processAsyncDataAvailable(SessionImpl.java:1661)
        at com.stc.jms.client.SessionImpl.processAsync(SessionImpl.java:1750)
        at com.stc.jms.client.SessionImpl.asyncEnter(SessionImpl.java:1778)
        at com.stc.jms.client.SessionImpl.access$500(SessionImpl.java:56)
        at com.stc.jms.client.SessionImpl$1.run(SessionImpl.java:1556)
        at com.stc.jms.client.SessionImpl$2.run(SessionImpl.java:1809)
        at java.lang.Thread.run(Thread.java:619)
|#]

[#|2008-07-27T20:09:25.062+1000|INFO|sun-appserver9.1|javax.enterprise.resource.resourceadapter|_ThreadID=28;_ThreadNa
me=JMS Async S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|In getLocalTransaction|#]

[#|2008-07-27T20:09:25.062+1000|INFO|sun-appserver9.1|javax.enterprise.resource.resourceadapter|_ThreadID=28;_ThreadNa
me=JMS Async S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|
|#]

[#|2008-07-27T20:09:25.078+1000|INFO|sun-appserver9.1|com.stc.jmsjca.core.Delivery|_ThreadID=28;_ThreadName=JMS Async
S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|JMSJCA-E027: Message with msgid=[ID:85afd:11b6371f4a6:1c28:c
0a83c03:11b63fe7427:5700402979664b4c9cef42553d265356] was seen 1 times. It will be forwarded (moved) to queue qJMSTrig
geredJCA_DLQ with msgid [ID:f6f97:11b6371f4a7:1c28:c0a83c03:11b63fe7456:8e572464bf464c0ab8c237abb3259a59]|#]
```

As expected, inspection of the table and the file shows no changes. The process failure occurred before any changes could be made.

Let's now submit a message with the literal "S2". According to MDB's logic the database update will be executed, but the file write will not be executed. It is expected that even though the update will have been executed there will be no change to the database table because the transaction will have been rolled back. Indeed, the server.log fragment in Listing 7-3 shows the log messages supporting this statement.

Listing 7-3 *Exception messages after submission of "S2" as a message*

```
[#|2008-07-27T20:13:58.781+1000|WARNING|sun-appserver9.1|pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA|_ThreadID=28;_Threa
dName=JMS Async S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;_RequestID=71cd1876-a120-443e-89e8-efb5992634
a3;|Failed to invoke _invoke_receive: java.lang.Exception: Throwing exception on S2 after DB Update
java.lang.Exception: Throwing exception on S2 after DB Update
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA.receive(jcaJMSTriggeredJCA.java:74)
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA._invoke_receive(jcaJMSTriggeredJCA.java:110)
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA.onMessage(jcaJMSTriggeredJCA.java:40)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at com.sun.enterprise.security.application.EJBSecurityManager.runMethod(EJBSecurityManager.java:1067)
        at com.sun.enterprise.security.SecurityUtil.invoke(SecurityUtil.java:176)
        at com.sun.ejb.containers.BaseContainer.invokeTargetBeanMethod(BaseContainer.java:2899)
        at com.sun.ejb.containers.BaseContainer.intercept(BaseContainer.java:3990)
        at com.sun.ejb.containers.MessageBeanContainer.deliverMessage(MessageBeanContainer.java:1111)
        at com.sun.ejb.containers.MessageBeanListenerImpl.deliverMessage(MessageBeanListenerImpl.java:74)
        at com.sun.enterprise.connectors.inflow.MessageEndpointInvocationHandler.invoke(MessageEndpointInvocationHandl
er.java:179)
        at $Proxy64.onMessage(Unknown Source)
        at com.stc.jmsjca.core.Delivery.deliverToEndpoint(Delivery.java:1075)
        at com.stc.jmsjca.core.SerialDelivery.onMessage(SerialDelivery.java:246)
        at com.stc.jms.client.SessionImpl.processAsyncDataAvailable(SessionImpl.java:1661)
        at com.stc.jms.client.SessionImpl.processAsync(SessionImpl.java:1750)
        at com.stc.jms.client.SessionImpl.asyncEnter(SessionImpl.java:1778)
        at com.stc.jms.client.SessionImpl.access$500(SessionImpl.java:56)
        at com.stc.jms.client.SessionImpl$1.run(SessionImpl.java:1556)
        at com.stc.jms.client.SessionImpl$2.run(SessionImpl.java:1809)
        at java.lang.Thread.run(Thread.java:619)
|#]

[#|2008-07-27T20:13:58.781+1000|INFO|sun-appserver9.1|javax.enterprise.resource.resourceadapter|_ThreadID=28;_ThreadNa
me=JMS Async S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|In getLocalTransaction|#]

[#|2008-07-27T20:13:58.781+1000|INFO|sun-appserver9.1|javax.enterprise.resource.resourceadapter|_ThreadID=28;_ThreadNa
me=JMS Async S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|
|#]

[#|2008-07-27T20:13:58.796+1000|INFO|sun-appserver9.1|com.stc.jmsjca.core.Delivery|_ThreadID=28;_ThreadName=JMS Async
S0;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|JMSJCA-E027: Message with msgid=[ID:6704f:11b6371f4a9:1c28:c
0a83c03:11b6402a15e:fa037958ece34019a0f5323a9acc4db1] was seen 1 times. It will be forwarded (moved) to queue qJMSTrig
geredJCA_DLQ with msgid [ID:9065d:11b6371f4a7:1c28:c0a83c03:11b6402a18c:8e572464bf464c0ab8c237abb3259a59]|#]
```

Inspection of the database table shows no change.

Inspection of the output file shows no change either. The code section that would have updated the file was never executed.

Finally, let's submit a message with the literal "S3". MDB logic dictates that the database must be updated and a record must be written to a file before throwing an exception. Since the file is not a transactional resource, even if the exception is thrown and the database update is not committed, the file write will still succeed.

The server.log fragment in Listing 7-4 shows the execution trace with both the database update and file write messages.

Listing 7-4 *Exception messages after submission of "S3" as a message*

```
[#|2008-07-27T20:18:46.296+1000|WARNING|sun-appserver9.1|pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA|_ThreadID=32;_Threa
dName=JMS Async S12;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;_RequestID=3aad9e01-b68c-4927-97c3-d6f904875
e48;|Failed to invoke _invoke_receive: java.lang.Exception: Throwing exception on S3 after DB Update and File Write
java.lang.Exception: Throwing exception on S3 after DB Update and File Write
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA.receive(jcaJMSTriggeredJCA.java:86)
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA._invoke_receive(jcaJMSTriggeredJCA.java:110)
        at pkg.jcaJMSTriggeredJCA.jcaJMSTriggeredJCA.onMessage(jcaJMSTriggeredJCA.java:40)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at com.sun.enterprise.security.application.EJBSecurityManager.runMethod(EJBSecurityManager.java:1067)
        at com.sun.enterprise.security.SecurityUtil.invoke(SecurityUtil.java:176)
        at com.sun.ejb.containers.BaseContainer.invokeTargetBeanMethod(BaseContainer.java:2899)
        at com.sun.ejb.containers.BaseContainer.intercept(BaseContainer.java:3990)
        at com.sun.ejb.containers.MessageBeanContainer.deliverMessage(MessageBeanContainer.java:1111)
        at com.sun.ejb.containers.MessageBeanListenerImpl.deliverMessage(MessageBeanListenerImpl.java:74)
        at com.sun.enterprise.connectors.inflow.MessageEndpointInvocationHandler.invoke(MessageEndpointInvocationHandl
er.java:179)
        at $Proxy65.onMessage(Unknown Source)
        at com.stc.jmsjca.core.Delivery.deliverToEndpoint(Delivery.java:1075)
        at com.stc.jmsjca.core.SerialDelivery.onMessage(SerialDelivery.java:246)
        at com.stc.jms.client.SessionImpl.processAsyncDataAvailable(SessionImpl.java:1661)
        at com.stc.jms.client.SessionImpl.processAsync(SessionImpl.java:1750)
        at com.stc.jms.client.SessionImpl.asyncEnter(SessionImpl.java:1778)
        at com.stc.jms.client.SessionImpl.access$500(SessionImpl.java:56)
        at com.stc.jms.client.SessionImpl$1.run(SessionImpl.java:1556)
        at com.stc.jms.client.SessionImpl$2.run(SessionImpl.java:1809)
        at java.lang.Thread.run(Thread.java:619)
|#]

[#|2008-07-27T20:18:46.312+1000|INFO|sun-appserver9.1|javax.enterprise.resource.resourceadapter|_ThreadID=32;_ThreadNa
me=JMS Async S12;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|In getLocalTransaction|#]

[#|2008-07-27T20:18:46.312+1000|INFO|sun-appserver9.1|javax.enterprise.resource.resourceadapter|_ThreadID=32;_ThreadNa
me=JMS Async S12;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|
|#]

[#|2008-07-27T20:18:46.312+1000|INFO|sun-appserver9.1|com.stc.jmsjca.core.Delivery|_ThreadID=32;_ThreadName=JMS Async
S12;Context=jcaJMSTriggeredJCA-jcaJMSTriggeredJCA-Context;|JMSJCA-E027: Message with msgid=[ID:89209:11b6371f4ae:1c28:
c0a83c03:11b6407041b:51105145b5974281a025718a7d4c074c] was seen 1 times. It will be forwarded (moved) to queue qJMSTri
ggeredJCA_DLQ with msgid [ID:232d4:11b6371f4af:1c28:c0a83c03:11b640704a8:3d83b48820054d8a830b604acf32dec0]|#]
```

As expected, the database was not changed.

Inspection of the file shows that it was updated.

8 Conclusion

One lesson from this example is to place invocation of nontransactional resources after invocation of transactional resources if logic permits. Another lesson is to consider breaking up logic into transactional and nontransactional units to minimize the complexity of exception handling.

A solution designer can take advantage of the JMS redelivery handling to handle exceptions at a MDB level. The built-in JMS redelivery mechanism can be utilized to overcome transient exception-causing conditions, such as temporary database unavailability, without requiring explicit logic in MDBs. The designer must, however, consider side-effects arising out of access to nontransactional resources, to minimize the adverse impact of retry attempts on these resources.

If the MDB does not throw an exception, the message that triggered it will be consumed and the transaction that spans the MDB will complete. If the MDB handles exceptions that arise during its execution, and does not rethrow any, the message that triggered it will also be consumed.