

TIBCO Enterprise Message Service™

Application Integration Guide

*Software Release 4.3
February 2006*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN *TIBCO ENTERPRISE MESSAGE SERVICE USER'S GUIDE*). USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIB, TIBCO, Information Bus, The Power of Now, TIBCO ActiveEnterprise, TIBCO Adapter, TIBCO Hawk, TIBCO Rendezvous, TIBCO Enterprise, TIBCO Enterprise Message Service, and the TIBCO logo are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, J2EE, JMS and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the *readme.txt* file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

Copyright © 1999–2006 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Figures	ix
Tables	xi
Preface	xiii
Related Documentation	xiv
TIBCO Enterprise Message Service Documentation	xiv
Other TIBCO Product Documentation	xiv
Third Party Documentation	xiv
How to Contact TIBCO Customer Support	xvi
 Chapter 1 Using JNDI With Third-Party Naming/Directory Services.	 1
Overview of Using JNDI With Third-Party Naming/Directory Services	2
Storing Administered Objects in a Naming/Directory Service	3
Retrieving Administered Objects from a Naming/Directory Service.	7
 Chapter 2 Overview of Third-Party Application Servers	 9
Third Party Application Servers	10
 Chapter 3 Integrating With JBoss 4.0.2	 11
Overview of Integrating With JBoss 4.0.2	12
Get the Example MDB Working Using JBossMQ	13
Get the Example MDB Working Using TIBCO Enterprise Message Service.	16
Modify the Example to use SSL Communications.	20
Adding the SSL JAR Files to the CLASSPATH for the JBoss Server.	20
Configuring the TIBCO Enterprise Message Service Server for SSL.	20
Configuring JBoss for SSL-based JMS Communications.	21
Stop and restart the JBoss server.	22
Adding the SSL JAR Files to the CLASSPATH for the Client Program	22
Adding the SSL JNDI Properties for the Client Program	22
Modify and Rebuild the Client.	23
Re-Run the Client Program	23
Container-Managed Transactions (XA)	24

Chapter 4 Integrating With JBoss 3.2.3	25
Overview of Integrating With JBoss 3.2.3	26
Get the Example MDB Working Using JBossMQ	27
Get the Example MDB Working Using TIBCO Enterprise Message Service	30
Modify the Example to use SSL Communications	34
Adding the SSL JAR Files to the CLASSPATH for the JBoss Server	34
Configuring the TIBCO Enterprise Message Service Server for SSL	34
Configuring JBoss for SSL-based JMS Communications	35
Stop and restart the JBoss server	36
Adding the SSL JAR Files to the CLASSPATH for the Client Program	36
Adding the SSL JNDI Properties for the Client Program	36
Modify and Rebuild the Client	37
Re-Run the Client Program	37
Container-Managed Transactions (XA)	38
Chapter 5 Integrating With JBoss 3.0.4	39
Overview of Integrating With JBoss 3.0.4	40
Get the Example MDB Working Using JBossMQ	41
Get the Example MDB Working Using TIBCO Enterprise Message Service	45
Modify the Example to use SSL Communications	49
Adding the SSL JAR Files to the CLASSPATH for the JBoss Server	49
Configuring the TIBCO Enterprise Message Service Server for SSL	49
Configuring JBoss for SSL-based JMS Communications	50
Stop and restart the JBoss server	51
Adding the SSL JAR Files to the CLASSPATH for the Client Program	51
Adding the SSL JNDI Properties for the Client Program	51
Modify and Rebuild the Client	52
Re-Run the Client Program	52
Container-Managed Transactions (XA)	53
Chapter 6 Integrating With Borland Enterprise Server 5.1	55
Configure Borland Enterprise Server to use TIBCO Enterprise Message Service	56
Configure TIBCO Enterprise Message Service for the Example Program	59
Configure Borland Enterprise Server for the Example Message Driven Bean	60
Using Container-Managed XA Transactions	60
Using XA Transactions That Are Not Container-Managed	62
Building and Deploying the Example MDB and the Example Client	64
Running This Example	65
Modifying This Example to use SSL Communications	66

Chapter 7 Integrating With Borland Enterprise Server 5.0	69
Configure Borland Enterprise Server to use TIBCO Enterprise Message Service	70
Configure TIBCO Enterprise Message Service for the Example Program	73
Configure Borland Enterprise Server for the Example Message Driven Bean	74
Building and Deploying the Example MDB and the Example Client	75
Running This Example.	76
Modifying This Example to use SSL Communications	77
 Chapter 8 Integrating With WebLogic Server 8.1	 81
Running the Example MDB with WebLogic Server	82
Configuring the Example MDB.	83
Adding TIBCO Enterprise Message Service to the WebLogic CLASSPATH	83
Creating Foreign JMSConnection, JMSConnectionFactory, and JMSDestination in WebLogic	83
Creating the Example MDB Destination Object Inside TIBCO EMS	84
Modifying the weblogic-ejb-jar.xml file for MDB	85
Modifying the Client Program to Use TIBCO Enterprise Message Service JNDI.	85
Rebuilding and Redeploying the Example MDB	87
Running the Example MDB Client	88
Modifying this Example to Use SSL Communication.	89
Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH.	89
Configure the TIBCO Enterprise Message Service Server for SSL	89
Modify the foreign JMSConnectionFactory in WebLogic to point to an SSLConnectionFactory	90
Modify the Example Client Program for SSL-Based Communication.	90
Rebuilding and Redeploying the Example MDB.	90
Running the Example MDB Client with SSL.	91
Modifying this Example to use Container Managed Transactions and XA	92
Modify the foreign JMSConnectionFactory in WebLogic to point to a XAConnectionFactory.	92
Create a JMS Connection factory that supports XA	92
Modifying the WebLogic Deployment files to make MDB to use transactions	92
 Chapter 9 Integrating With WebLogic Server 7.0	 95
Running the Example MDB with WebLogic Server	96
Configuring the Example MDB.	97
Adding TIBCO Enterprise Message Service to the WebLogic Server CLASSPATH	97
Modifying the MDB Deployment Descriptor for TIBCO Enterprise Message Service.	97
Modifying the Client Program to Use TIBCO Enterprise Message Service JNDI.	98
Creating the Example MDB Destination Object Inside TIBCO EMS	99
Rebuilding and Redeploying the Example MDB	100
Running the Example MDB Client	101
Modifying this Example to Use SSL Communication.	102

Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH	102
Configure the TIBCO Enterprise Message Service Server for SSL	102
Configure Example MDB for SSL-Based Communication	103
Modify the Example Client Program for SSL-Based Communication	103
Rebuilding and Redeploying the Example MDB	103
Running the Example MDB Client with SSL	104
Modifying this Example to use Container Managed Transactions and XA	105
Create a JMS Connection factory that supports XA	105
Modifying the Weblogic Deployment files to make MDB to use transactions	105
Chapter 10 Integrating With WebLogic Server 6.1	107
Using TIBCO Enterprise Message Service With WebLogic Server	108
Using TIBCO Enterprise Message Service with WebLogic Server Message Driven Beans	114
Modifying This Example to use SSL Communication	118
Chapter 11 Integrating With IBM WebSphere Application Server Version 5	121
Overview of Integrating With IBM WebSphere	122
Get the sample MDB running with the WebSphere Embedded JMS Provider	123
Get the Publish and Subscribe Sample Working	123
Get the Point-to-Point Sample Working	124
Get the Sample MDB running with TIBCO Enterprise Message Service	126
Create the TIBCO Enterprise Message Service Administered Objects	126
Configure WebSphere for the TIBCO Enterprise Message Service JNDI Provider	126
Add TIBCO Enterprise Message Service as a JMS Provider to the Application Server	126
Configure JNDI Bindings for TIBCO Enterprise Message Service Connection Factories for the Application Server	127
Configure JNDI Bindings for TIBCO Enterprise Message Service Destinations for the Application Server	128
Create new Listener Ports for TIBCO Enterprise Message Service	129
Reassemble the Sample MDBs to Use the New TIBCO Enterprise Message Service Listener Ports	130
Redefine the Resource Reference and Resource Environment Reference for the Point-to-Point Sample MDB	131
Redefine the Resource Environment References in the Application Client Samples	132
Add TIBCO Enterprise Message Service as a JMS Provider to the Application Client	133
Configure the JNDI bindings for TIBCO Enterprise Message Service Connection Factories for the Application Client	134
Update the Deployed Application on the Server	134
Run the Samples Application Client	136
Modify the Samples to Use SSL Communications	137
Enable SSL in the TIBCO Enterprise Message Service Server	137
Create JNDI Names for the SSL Queue and Topic Connection Factories	137
Add the Additional SSL JNDI Properties to WebSphere	138
Configure SSL Communications Between the Application Server and the TIBCO Enterprise Message Service Server	138

Configure SSL Communications between the Point-to-Point Sample MDB and the TIBCO Enterprise Message Service Server.	139
Configure SSL Communications between the Application Client and the TIBCO Enterprise Message Service Server.	140
Update the Deployed Application on the Server.	141
Run the Samples Application Client	142
Chapter 12 Integrating With Sun Java System Application Server 7	143
Run the MDB Sample with Built-In JMS	144
Run the MDB Sample with TIBCO EMS	145
Configure Application Server	145
Register JMS Resources with Application Server	145
Run the Sample	145
Run the MDB Sample with TIBCO EMS using SSL.	147
Configure the EMS Server	147
Java Security Policy	147
Configure Application Server	147
Index	151

Figures

Figure 1	Object lookup in TIBCO Enterprise Message Service server.	3
Figure 2	Object created locally by the client.	5

Tables

Table 1	TIBCO Enterprise Message Service and WebLogic Server JMS implementations	108
---------	--	-----

Preface

TIBCO Enterprise Message Service™ software lets application programs send and receive messages according to the Java Message Service (JMS) protocol. It also integrates with TIBCO Rendezvous and TIBCO SmartSockets message products.



This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

Topics

- *Related Documentation, page xiv*
- *How to Contact TIBCO Customer Support, page xvi*

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Enterprise Message Service Documentation

The following documents form the TIBCO Enterprise Message Service documentation set:

- *TIBCO Enterprise Message Service User's Guide* Read this manual to gain an overall understanding of the product, its features, and configuration.
- *TIBCO Enterprise Message Service Installation* Read the relevant sections of this manual before installing this product.
- *TIBCO Enterprise Message Service Application Integration Guide* This manual presents detailed instructions for integrating TIBCO Enterprise Message Service with third-party products.
- *TIBCO Enterprise Message Service C & COBOL API Reference* The C API reference is available in HTML and PDF formats.
- *TIBCO Enterprise Message Service Java API Reference* The Java API reference is available as JavaDoc, and you can access the reference only through the HTML documentation interface.
- *TIBCO Enterprise Message Service .NET API Reference* The .NET API reference is available in PDF and HTML format.
- *TIBCO Enterprise Message Service Release Notes* Release notes summarize new features, changes in functionality, and closed issues. This document is available only in PDF format.

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO Rendezvous™ software
- TIBCO SmartSockets™ software

Third Party Documentation

- Java™ Message Service specification, available through java.sun.com/products/jms/index.html

- *Java™ Message Service* by Richard Monson-Haefel and David A. Chappell, O'Reilly and Associates, Sebastopol, California, 2001.
- *JBoss 3.2 Administration and Development Documentation* by Scott Stark and Marc Fleury and The JBoss Group, Que/Sams, 2003.
- *JBoss 2.4 Administration and Development Documentation* by Scott Stark and Marc Fleury and The JBoss Group, Que/Sams, 2002.

How to Contact TIBCO Customer Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support Services as follows.

- For an overview of TIBCO Support Services, and information about getting started with TIBCO Product Support, visit this site:

<http://www.tibco.com/services/support/default.jsp>

- If you already have a valid maintenance or support contract, visit this site:

<http://support.tibco.com>

Entry to this site requires a username and password. If you do not have a username, you can request one.

Chapter 1

Using JNDI With Third-Party Naming/Directory Services

TIBCO Enterprise Message Service™ allows you to work with third-party naming/directory service products. This chapter describes how to integrate these products with TIBCO Enterprise Message Service.

Topics

- *Overview of Using JNDI With Third-Party Naming/Directory Services, page 2*
- *Storing Administered Objects in a Naming/Directory Service, page 3*
- *Retrieving Administered Objects from a Naming/Directory Service, page 7*

Overview of Using JNDI With Third-Party Naming/Directory Services

TIBCO Enterprise Message Service supports the storage (binding) and retrieval (look-up) of ConnectionFactories and Destinations in third-party naming or directory services. Examples of such services are an LDAP server, the RMI registry, or the file system.



Third-party naming or directory servers are separate products that must be installed and set up independently of TIBCO Enterprise Message Service. This is usually done by a system administrator.

To use a third-party directory service, you must have a JNDI provider for that specific type of service. A JNDI provider presents a common API to the service regardless of the service type or service vendor, much like a JDBC driver presents a common API on top of various vendors' databases.

The Java 2 SDK, contains JNDI providers for LDAP and RMI registry, among others. A JNDI provider for the file system can be downloaded from the JNDI home page at java.sun.com.

From a client perspective, looking up administered objects is accomplished in virtually the same way regardless of whether the object is in a third-party naming/directory service or in the TIBCO Enterprise Message Service server. For third-party services, the only prerequisite is that the objects must have previously been stored there. That is, the object must be bound to a name in the context of that service. This is usually a task performed by a system administrator.



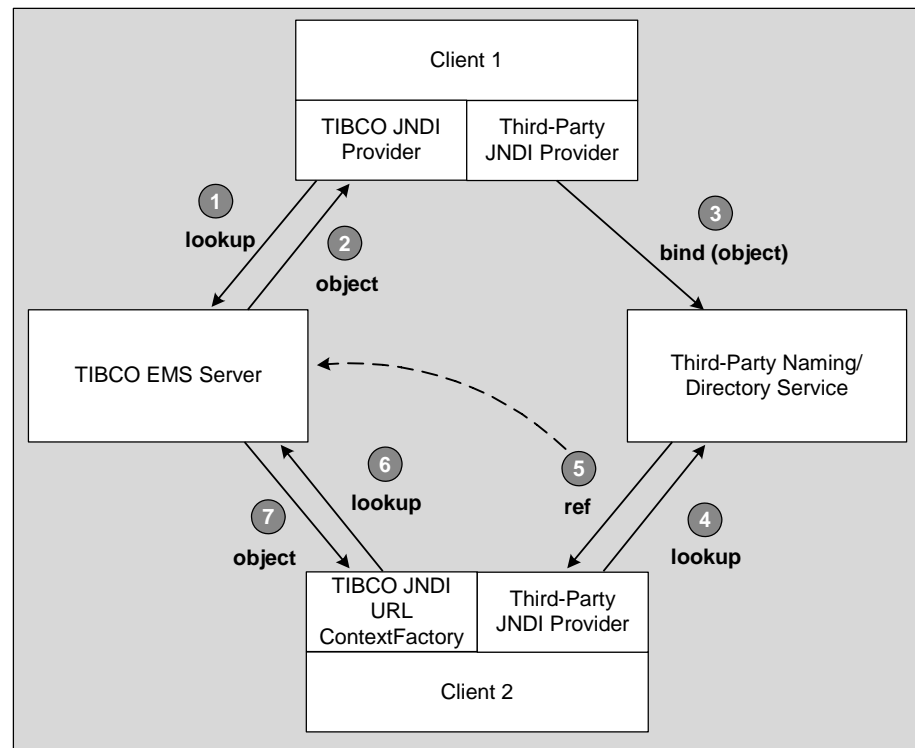
There is no automatic synchronization of administered objects between the TIBCO Enterprise Message Service server and any foreign naming/directory service. Keeping the two synchronized is the responsibility of the system administrator.

Storing Administered Objects in a Naming/Directory Service

All TIBCO Enterprise Message Service administered objects implement the JNDI “Referenceable” interface. This means that when they are bound in a foreign naming/directory service, what is physically stored there is not the serialized object itself, but rather a “Reference” object that knows how to re-create the original object when the object is looked up.

There are two forms of Reference objects that are stored, and which form is used depends on the origin of the original object. If the original object was looked up in the TIBCO Enterprise Message Service server, then the Reference object that gets stored for the object contains a URL pointer to the originating server. When this object is looked up in the foreign naming/directory service, the JNDI provider follows the associated URL and retrieves the object from the TIBCO Enterprise Message Service server. Figure 1 illustrates this case.

Figure 1 Object lookup in TIBCO Enterprise Message Service server

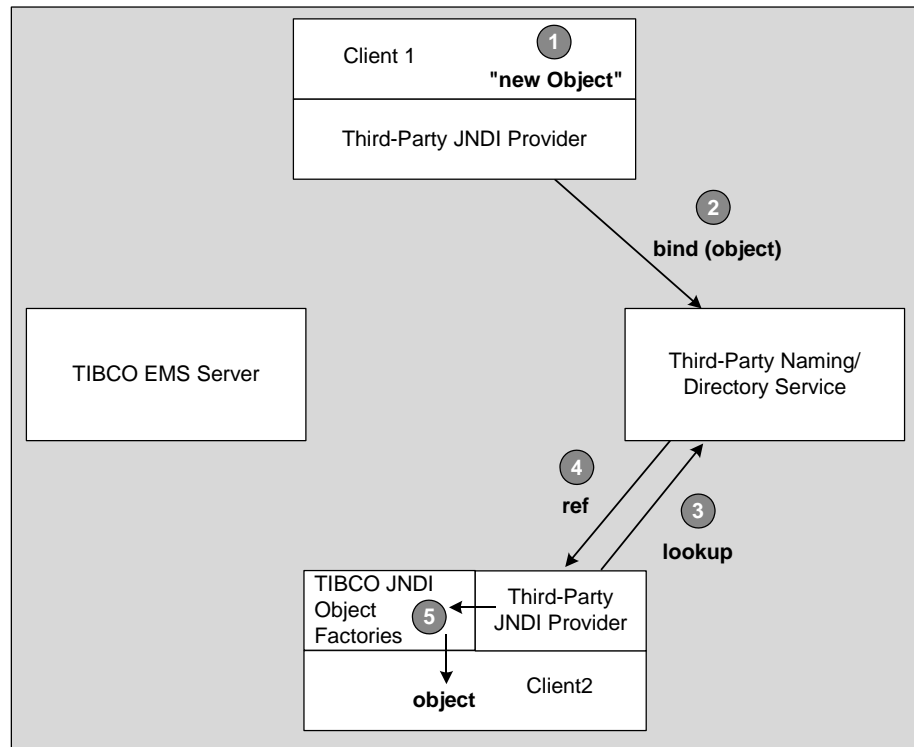


In Figure 1, the following occurs:

1. Client 1 requests a lookup of an object in TIBCO Enterprise Message Service server by way of the JNDI provider supplied in TIBCO Enterprise Message Service.
2. The TIBCO Enterprise Message Service server returns the object to Client 1.
3. Client 1 binds the object into a third-party service using a third-party JNDI provider. The object is stored as a URL reference to the actual object in the TIBCO Enterprise Message Service server.
4. Client 2 requests a lookup of an object in the third-party service using the third-party JNDI provider.
5. The URL reference is returned by the third-party JNDI provider.
6. JNDI realizes that this is a reference, and further that it is a URL reference to the TIBCO Enterprise Message Service server. Therefore, it invokes the URL context factory of the TIBCO Enterprise Message Service JNDI provider which requests a lookup of the object in the TIBCO Enterprise Message Service server.
7. The TIBCO Enterprise Message Service server returns the object to Client 2.

If however, the object was created locally by the client using the public constructor of the class, then the Reference object that is stored for the object contains whatever information is required to re-create the object locally. When this object is subsequently looked up in the foreign naming/directory service, the JNDI provider uses the information stored in the Reference object to instantiate the original object locally. All of this behavior happens automatically without any special interaction required of the client. Figure 2 illustrates this case.

Figure 2 Object created locally by the client



In Figure 2, the following occurs:

1. Client 1 creates a new administered object using the constructor of the class.
2. Client 1 binds the object into the third-party service using the third-party JNDI provider. The object is stored as a local reference.
3. Client 2 requests a lookup of the object in the third-party service using the third-party JNDI provider
4. The local reference is returned to the third-party JNDI provider.
5. JNDI realizes that this is a local reference, and invokes the TIBCO Enterprise Message Service JNDI object factory associated with the reference, which creates a new instance of the object locally and returns it to Client 2.

This behavior occurs automatically without any special interaction from the client.

Storing objects as a URL reference requires that the TIBCO Enterprise Message Service server be up and running (at the same URL) when the object is looked up in the foreign naming/directory service. Storing objects created locally does not have this requirement, however, because there is no automatic synchronization between the foreign naming/directory service and the TIBCO Enterprise Message Service server. There is no guarantee that the object returned from a lookup is *valid*, that is, that it exists inside the TIBCO Enterprise Message Service server. Storing objects as a URL reference ensures that the returned object is always a valid object.

For an example of storing administered objects in both of these forms, refer to the **tibjmsJNDIStore.java** example included with TIBCO Enterprise Message Service.

All TIBCO Enterprise Message Service administered objects implement the `javax.naming.Referenceable` interface. Therefore, these objects cannot be directly bound, along with a `javaCodebase` attribute, into a directory service that follows the schema defined in RFC 2713, and subsequently looked up with a service provider that uses the `javaCodebase` attribute (such as Sun's LDAP service provider). To successfully look up a TIBCO Enterprise Message Service administered object bound with the `javaCodebase` attribute, the object must first be manually serialized, then the serialized object can be directly bound into the directory service.

Retrieving Administered Objects from a Naming/Directory Service

In order to retrieve (look up) administered objects from a foreign naming/directory service, an initial context must be established for that service. The following example illustrates how to create an initial context using the Sun LDAP JNDI provider for an LDAP server running on the local machine, listening on port 20329, where the root naming context is `myJMSObjects`:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,
        "ldap://localhost:20329/o=myJMSObjects");
Context context = new InitialContext(env);
```

Once the context is established, retrieving (looking up) administered objects from a foreign naming/directory service is accomplished no differently than with the TIBCO Enterprise Message Service server. However, there is one exception — two properties must be added to the context object that inform the JNDI provider where to find the object factories for the TIBCO Enterprise Message Service administered objects. The JNDI provider invokes these factories when any of the objects are retrieved. The properties are responsible for locating the factories that create the appropriate instances of the desired objects for the user.

The following example illustrates setting these properties. If the variable "context" contains the initial context for the foreign naming/directory service, then these properties would be set with the following two lines:

```
context.addToEnvironment(Context.OBJECT_FACTORIES,
        "com.tibco.tibjms.naming.TibjmsObjectFactory");
context.addToEnvironment(Context.URL_PKG_PREFIXES,
        "com.tibco.tibjms.naming");
```

Once these properties are set, then the "lookup" method of the context can be used to retrieve any object stored in that context.

For an example of retrieving administered objects from a foreign naming/directory service, see the `tibjmsJNDIRead.java` example included with TIBCO Enterprise Message Service.

Chapter 2

Overview of Third-Party Application Servers

TIBCO Enterprise Message Service successfully integrates with third-party J2EE EJB (application) servers so that TIBCO Enterprise Message Service can drive Message Driven Beans (MDBs) within the application servers. This chapter gives an overview of this integration.

Topics

- *Third Party Application Servers, page 10*

Third Party Application Servers

Third party servers which have been tested with TIBCO Enterprise Message Service are:

- JBoss 3.2.3 and 3.0.4 from JBoss.org
- Borland Enterprise Server 5.1 from Borland
- WebLogic 8.1, 7.0 and 6.1 with Service Pack 1 from BEA
- IBM WebSphere Application Server Version 5
- Sun Java System Application Server 7

Integration with other application servers is possible, although it has not been tested.

TIBCO Enterprise Message Service successfully integrates with third party J2EE EJB (application) servers so that TIBCO Enterprise Message Service can drive Message Driven Beans (MDBs) within the application servers.

TIBCO Enterprise Message Service implements the `ConnectionFactory` interface of the JMS specification for application servers that follow the JMS specification for JMS integration. The application servers listed above that use this interface are Borland Enterprise Server and JBoss.

TIBCO Enterprise Message Service also implements all interfaces necessary for Java Transaction API (JTA) compliance.

Special TIBCO Enterprise Message Service adapter classes are required for integration with some of the above listed application servers. These classes are contained in a separate JAR file included with this release, `tibjmsapps.jar`.

The following chapters contain more detailed instructions on how to integrate TIBCO Enterprise Message Service with each of the above listed application servers. Each chapter details how to run an example program provided by the application server using TIBCO Enterprise Message Service. When applicable, each chapter also describes how to modify the example to use SSL for communications between TIBCO Enterprise Message Service, the application server, and the example application.

Chapter 3 **Integrating With JBoss 4.0.2**

This chapter describes integrating TIBCO Enterprise Message Service with the JBoss J2EE application server, version 4.0.2. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside JBoss from any EMS client.

Topics

- *Overview of Integrating With JBoss 3.2.3, page 26*
- *Get the Example MDB Working Using JBossMQ, page 27*
- *Get the Example MDB Working Using TIBCO Enterprise Message Service, page 30*
- *Modify the Example to use SSL Communications, page 34*
- *Container-Managed Transactions (XA), page 38*

Overview of Integrating With JBoss 4.0.2

This chapter describes integrating TIBCO Enterprise Message Service with the JBoss J2EE application server, version 4.0.2. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside JBoss from any EMS client.

The techniques described in this chapter assume you are using Windows 2000, and that you have already downloaded and installed JBoss 4.0.2. It further assumes that you are running JBoss, the TIBCO Enterprise Message Service server, and the client program on the same machine.

Throughout the following procedures, environment variables are used to refer to specific directories within the JBoss installation. They are not actually needed by the JBoss server, but merely facilitate the reference to different directories in the JBoss installation.

The following environment variables are used throughout the discussion below:

```
JBOSS_HOME = C:\JBoss-4.0.2
JBOSS_CLIENT = %JBOSS_HOME%\client
JBOSS_DEPLOY = %JBOSS_HOME%\server\default\deploy
JBOSS_CONF = %JBOSS_HOME%\server\default\conf
```



The example in this chapter configures an MDB that uses container-managed transactions. For more information, see Container-Managed Transactions (XA) on page 53.

Get the Example MDB Working Using JBossMQ

1. To build the example MDB, add the following to your CLASSPATH:
`%JBOSS_CLIENT%\jboss-j2ee.jar`
2. Compile the example MDB, `TextMDB.java`. The source code for this example is located in *JBoss Administration and Development*, Chapter 4, Example 2.
3. Create a directory named `META-INF` in the output directory that now contains the `org.jboss.chap4.ex2.TextMDB.class`.
4. Copy the files `ejb-jar.xml` and `jboss.xml` from the source directory associated with Example 2 of Chapter 4 of *JBoss Administration and Development*, to the `META-INF` directory.
5. Create the EJB jar file by changing directories to the output directory and issuing the following command:

```
jar cvf myejb.jar META-INF org\jboss\chap4\ex2\TextMDB.class
```

6. Map Connection Factory Names

In JBoss 4.0.2, the JNDI names `QueueConnectionFactory` and `TopicConnectionFactory` do not come pre-mapped to the internal JBoss connection factory called `ConnectionFactory`. Therefore you must add this mapping in order to use the example MDB without modification. The mapping must be configured by adding the following lines in the file

`%JBOSS_DEPLOY%\jms\jbossmq-service.xml`:

```
<!--===== -->
<!-- JBossMQ -->
<!--===== -->

<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=QueueConnectionFactory">
  <attribute name="ToName">ConnectionFactory</attribute>
  <attribute name="FromName">QueueConnectionFactory</attribute>
</mbean>
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TopicConnectionFactory">
  <attribute name="ToName">ConnectionFactory</attribute>
  <attribute name="FromName">TopicConnectionFactory</attribute>
</mbean>
```

7. Start the JBoss server by changing to the `%JBOSS_HOME%\bin` directory and issuing the following command:
`run`
8. Copy `myejb.jar` to `%JBOSS_DEPLOY%`.
9. In the JBoss window, you should see output similar to the following:

```

19:35:33,343 INFO [org.jboss.deployment.MainDeployer] Starting deployment of
package: file:/C:/jboss-4.0.2/server/default/deploy/myejb.jar
19:35:34,296 INFO [org.jboss.ejb.EjbModule] Deploying TextMDB
19:35:34,875 INFO [org.jboss.ejb.plugins.jms.DLQHandler] Started null
19:35:34,875 INFO [org.jboss.ejb.plugins.jms.JMSContainerInvoker] Started
jboss.j2ee:binding=message-driven-bean,jndiName=local/TextMDB,plugin=invoker,serv
ice=EJB
19:35:34,875 INFO [org.jboss.ejb.plugins.MessageDrivenInstancePool] Started
jboss.j2ee:jndiName=local/TextMDB,plugin=pool,service=EJB
19:35:34,875 INFO [org.jboss.ejb.MessageDrivenContainer] Started
jboss.j2ee:jndiName=local/TextMDB,service=EJB
19:35:34,875 INFO [org.jboss.ejb.EjbModule] Started
jboss.j2ee:module=myejb.jar,service=EjbModule
19:35:34,875 INFO [org.jboss.ejb.EJBDeployer] Deployed:
file:/C:/jboss-4.0.2/server/default/deploy/myejb.jar
19:35:34,968 INFO [org.jboss.deployment.MainDeployer] Deployed package:
file:/C:/jboss-4.0.2/server/default/deploy/myejb.jar

```

10. To build the client program, make sure the following are in your CLASSPATH:

```

%JBOSS_CLIENT%\jboss-j2ee.jar
%JBOSS_CLIENT%\concurrent.jar

```

11. Compile the client program, `org.jboss.chap4.ex2.SendRecvClient.java`. The source to this program is listed in the book *JBoss Administration and Development*.

12. To run the client program, add the following to your CLASSPATH:

```

%JBOSS_CLIENT%\jnp-client.jar
%JBOSS_CLIENT%\jbossmq-client.jar
%JBOSS_CLIENT%\jboss-common-client.jar
%JBOSS_CLIENT%\jnet.jar
%JBOSS_CLIENT%\log4j.jar
%JBOSS_CLIENT%

```

The `%JBOSS_CLIENT%` directory is included so that the file `jndi.properties` in that directory can be found (see the next step).

13. In JBoss 4.0.2, a `jndi.properties` file does not come pre-configured for the client, therefore, you will have to create one. The easiest way is to first copy `jndi.properties` from `%JBOSS_CONF%` to `%JBOSS_CLIENT%`. Then add the following line in the copied file:

```
java.naming.provider.url=localhost
```

14. Run the client program:

```
java org.jboss.chap4.ex2.SendRecvClient
```

You should see output like the following in the client program window:

```

Begin sendRecvAsync
sendRecvAsync, sent text=A text msg#0
sendRecvAsync, sent text=A text msg#1
...
sendRecvAsync, sent text=A text msg#8
sendRecvAsync, sent text=A text msg#9

```

```

End sendRecvAsync
onMessage, recv text=A text msg#0processed by: 3824284
onMessage, recv text=A text msg#3processed by: 32953059
....
onMessage, recv text=A text msg#6processed by: 32420722
onMessage, recv text=A text msg#8processed by: 23916456

```

You should also see output like the following in the JBoss server console:

```

20:03:25,046 INFO [STDOUT] TextMDB.ctor, this=3824284
20:03:25,078 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=3824284
20:03:25,078 INFO [STDOUT] TextMDB.ejbCreate, this=3824284
20:03:25,109 INFO [STDOUT] TextMDB.ctor, this=2003839
20:03:25,109 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=2003839
20:03:25,109 INFO [STDOUT] TextMDB.ejbCreate, this=2003839
20:03:25,109 INFO [STDOUT] TextMDB.ctor, this=30170403
20:03:25,125 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=30170403
20:03:25,125 INFO [STDOUT] TextMDB.ejbCreate, this=30170403
20:03:25,109 INFO [STDOUT] TextMDB.ctor, this=32953059
20:03:25,140 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=32953059
20:03:25,140 INFO [STDOUT] TextMDB.ejbCreate, this=32953059
20:03:25,125 INFO [STDOUT] TextMDB.ctor, this=31834937
20:03:25,156 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=31834937
20:03:25,156 INFO [STDOUT] TextMDB.ejbCreate, this=31834937
20:03:25,187 INFO [STDOUT] TextMDB.onMessage, this=3824284
20:03:25,187 INFO [STDOUT] TextMDB.sendReply, this=3824284, dest=QUEUE.A
20:03:25,125 INFO [STDOUT] TextMDB.ctor, this=13863286
20:03:25,203 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=13863286
20:03:25,203 INFO [STDOUT] TextMDB.ejbCreate, this=13863286
20:03:25,203 INFO [STDOUT] TextMDB.onMessage, this=32953059
20:03:25,218 INFO [STDOUT] TextMDB.sendReply, this=32953059, dest=QUEUE.A
20:03:25,234 INFO [STDOUT] TextMDB.onMessage, this=3824284
20:03:25,234 INFO [STDOUT] TextMDB.sendReply, this=3824284, dest=QUEUE.A
20:03:25,218 INFO [STDOUT] TextMDB.ctor, this=32420722
20:03:25,250 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=32420722
20:03:25,250 INFO [STDOUT] TextMDB.ejbCreate, this=32420722
20:03:25,296 INFO [STDOUT] TextMDB.ctor, this=23916456
20:03:25,296 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=23916456
20:03:25,296 INFO [STDOUT] TextMDB.ejbCreate, this=23916456
20:03:25,312 INFO [STDOUT] TextMDB.onMessage, this=31834937
20:03:25,312 INFO [STDOUT] TextMDB.sendReply, this=31834937, dest=QUEUE.A
20:03:25,328 INFO [STDOUT] TextMDB.onMessage, this=2003839
20:03:25,328 INFO [STDOUT] TextMDB.sendReply, this=2003839, dest=QUEUE.A
20:03:25,296 INFO [STDOUT] TextMDB.onMessage, this=32953059
20:03:25,343 INFO [STDOUT] TextMDB.sendReply, this=32953059, dest=QUEUE.A
20:03:25,343 INFO [STDOUT] TextMDB.onMessage, this=13863286
20:03:25,343 INFO [STDOUT] TextMDB.sendReply, this=13863286, dest=QUEUE.A
20:03:25,328 INFO [STDOUT] TextMDB.onMessage, this=30170403
20:03:25,343 INFO [STDOUT] TextMDB.sendReply, this=30170403, dest=QUEUE.A
20:03:25,375 INFO [STDOUT] TextMDB.onMessage, this=32420722
20:03:25,375 INFO [STDOUT] TextMDB.sendReply, this=32420722, dest=QUEUE.A
20:03:25,421 INFO [STDOUT] TextMDB.onMessage, this=23916456
20:03:25,421 INFO [STDOUT] TextMDB.sendReply, this=23916456, dest=QUEUE.A

```

Get the Example MDB Working Using TIBCO Enterprise Message Service



This example MDB uses container-managed transactions. For more information, see Container-Managed Transactions (XA) on page 53.

Queues and Connection Factories

1. Start the `tibemsd` server and the `tibemsadmin` console.
2. Create three queues (`queue/A`, `queue/B` and `queue/DLQ`) and two XA connection factories (`XAQueueConnectionFactory` and `XATopicConnectionFactory`), by entering the following commands in `tibemsadmin`:

```
> connect
> create queue queue/A
> create queue queue/B
> create queue queue/DLQ
> create factory XAQueueConnectionFactory xaqueue url=tcp://7222
> create factory XATopicConnectionFactory xatopic url=tcp://7222
```

3. Make a backup copy (in a separate directory) of the configuration files that will be changed:

```
%JBOSS_DEPLOY%\jms\jms-ds.xml
%JBOSS_CONF%\jboss-service.xml
%JBOSS_CONF%\standardjboss.xml
<mdb output>\META-INF\jboss.xml
```



You should copy the files in the `%JBOSS_DEPLOY%` directory to another directory, rather than rename the files in place. JBoss attempts to deploy all files in that directory, regardless of name or file extension.

4. Add TIBCO EMS and the TIBCO EMS adapter class for JBoss to the CLASSPATH of the JBoss server by modifying the file `%JBOSS_CONF%\jboss-service.xml` as described below. Substitute an appropriate JAR file CLASSPATH for your installation.

Add the following lines under the `<server>` element in the file `%JBOSS_CONF%\jboss-service.xml`:

```
<!-- TIBCO Enterprise Message Service classpath -->
<classpath codebase="file:/C:\TIBCO\EMS\clients\java"
  archives="tibjms.jar" />
```


5. Reconfigure the JMSProviderLoader mbean to load TIBCO Enterprise Message Service instead of JBoss MQ. To do so, edit the file %JBOSS_DEPLOY%\jms\jms-ds.xml to resemble these lines:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name=":service=JMSProviderLoader,name=TibjmsProvider">
  <attribute name="ProviderName">TIBCOJMSProvider</attribute>
  <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapter</attribute>
  <attribute name="QueueFactoryRef">XAQueueConnectionFactory</attribute>
  <attribute name="TopicFactoryRef">XATopicConnectionFactory</attribute>
  <attribute name="Properties">
    java.naming.security.principal=jbosslookup
    java.naming.security.credentials=jbosslookup
    java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory
    java.naming.factory.url.pkgs=com.tibco.tibjms.naming
    java.naming.provider.url=tibjmsnaming://localhost:7222
  </attribute>
</mbean>
```

6. When the sample MDB looks up the QueueConnectionFactory (in order to create a QueueSender to send a message back to the initiator), it looks it up as java:comp/env/jms/QCF. Because we want the QueueConnectionFactory object that is returned from the lookup to be a TIBCO Enterprise Message Service XAQueueConnectionFactory, we must store a JNDI LinkRef under that name in the JBoss JNDI implementation that points to the XAQueueConnectionFactory in the TIBCO Enterprise Message Service implementation. Adding the following lines in %JBOSS_DEPLOY%\jms\jms-ds.xml accomplishes this.

```
<!-- Redirect QueueConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
  name="DefaultDomain:service=NamingAlias,fromName=QueueConnectionFactory">
  <attribute name="ToName">tibjmsnaming://localhost/XAQueueConnectionFactory
  </attribute>
  <attribute name="FromName">QueueConnectionFactory</attribute>
</mbean>
```

7. When the SendRecvClient test program looks up the ConnectionFactory referenced in the test program, it needs to be redirected to the appropriate QueueConnectionFactory configured in EMS. Add the following lines in %JBOSS_DEPLOY%\jms\jms-ds.xml to accomplish this.

```
<mbean code="org.jboss.naming.NamingAlias"
  name="DefaultDomain:service=NamingAlias,fromName=ConnectionFactory">
  <attribute name="ToName">tibjmsnaming://localhost/QueueConnectionFactory
  </attribute>
  <attribute name="FromName">ConnectionFactory</attribute>
</mbean>
```

8. When the JBoss server invokes JNDI and encounters the tibjmsnaming scheme, the server must be able to find the TIBCO Enterprise Message Service

URLConnectionFactory. Therefore, modify the file
%JBASS_CONF%\jndi.properties as follows:

Change:

```
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

To

```
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces:com.tibco.tibjms.naming
```

9. In the %JBASS_CONF%\standardjboss.xml file, modify the following line.

Change:

```
<JMSProviderAdapterJNDI>DefaultJMSProvider
</JMSProviderAdapterJNDI>
```

To:

```
<JMSProviderAdapterJNDI>TIBCOJMSProvider
</JMSProviderAdapterJNDI>
```

This change sets "TIBCOJMSProvider" as the JMS Provider Adapter JNDI name.

10. In the %JBASS_CONF%\standardjboss.xml file, modify the following line.

Change:

```
<DestinationQueue>queue/DLQ</DestinationQueue>
```

To:

```
<DestinationQueue>
  tibjmsnaming://localhost/queue/DLQ
</DestinationQueue>
```

This change specifies the TIBCO Enterprise Message Service JNDI name for the MDB Dead Letter Queue (DLQ) queue/DLQ.

11. Move the following files out of the %JBASS_DEPLOY% directory. These files are not needed when using TIBCO Enterprise Message Service and therefore must not be deployed:

```
%JBASS_DEPLOY%\jms\jbossmq-service.xml
%JBASS_DEPLOY%\jms\jbossmq-destinations-service.xml
```

12. Stop the JBoss server, then it restart by entering:

```
run
```

13. When the client program invokes JNDI, it should use the TIBCO Enterprise Message Service JNDI server. Modify %JBOSS_CLIENT%\jndi.properties to use TIBCO Enterprise Message Service JNDI by setting the following property:

```
java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory
```

14. Add C:\TIBCO\EMS\clients\java\tibjms.jar to the CLASSPATH of the client program.
15. Run the client program as you did in the previous section. You should see the same output.

Modify the Example to use SSL Communications

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, JBoss, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Adding the SSL JAR Files to the CLASSPATH for the JBoss Server

Add the TIBCO `tibcrypt.jar` file to the CLASSPATH of the JBoss server by modifying the file `%JBOSS_CONF%\jboss-service.xml` as described below. Substitute an appropriate JAR file CLASSPATH for your installation.

Add the following line under the `<server>` element in `%JBOSS_CONF%\jboss-service.xml`:

```
<classpath codebase="file:/C:\TIBCO\EMS\clients\java"
    archives="tibcrypt.jar" />
```

Configuring the TIBCO Enterprise Message Service Server for SSL

1. Start `tibemsd` in the working directory `C:\TIBCO\ems\bin` as follows:

```
tibemsd -config tibemsdssl.conf
```

When `tibemsd` starts you should see messages like the following in the console window, confirming SSL is enabled:

```
17:09:03 Secure Socket Layer is enabled, using OpenSSL 0.9.7c.
17:09:03 Accepting connections on tcp://localhost:7222.
17:09:03 Accepting connections on ssl://localhost:7243.
17:09:03 Server is active.
```

2. Start `tibemsdadmin` (administration tool) and enter the following commands.

First, create a new `XAQueueConnectionFactory` that establishes SSL connections:

```
create factory SSLXAQueueConnectionFactory xaqueue url=ssl://7243
```

Second, disable host verification for connections that this connection factory creates:

```
setprop factory SSLXAQueueConnectionFactory ssl_verify_host=disabled
```

This is the simplest SSL configuration.

Configuring JBoss for SSL-based JMS Communications

There are two aspects to SSL communications between JBoss and the TIBCO EMS server. The first is for messaging between the JBoss and TIBCO servers to occur over SSL. The second is for JNDI lookups from JBoss to the TIBCO JNDI provider to occur over SSL. The following two sections separately describe the required steps for each.

JMS Messaging over SSL

1. Modify the line you added to %JBOSS_DEPLOY%\jms\jms-ds.xml in the previous section (which specifies the QueueFactoryRef attribute of the JMS ProviderLoader) to be the new connection factory you just created (which establishes SSL connections):

```
<attribute name="QueueFactoryRef">
    SSLXAQueueConnectionFactory
</attribute>
```

2. Modify the line you added to %JBOSS_DEPLOY%\jms\jms-ds.xml in the previous section so it creates the JNDI LinkRef ToName to the new SSL connection factory—namely SSLXAQueueConnectionFactory (which you created above):

```
<attribute name="ToName">
    tibjmsnaming://localhost/SSLXAQueueConnectionFactory
</attribute>
```

JNDI Lookups over SSL

1. In the file %JBOSS_CONF%\jndi.properties, add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```

These properties specify the SSL protocol for JNDI lookups, and disable host verification.

2. Add the following line in the JMSProviderLoader mbean in %JBOSS_DEPLOY%\jms\jms-ds.xml:

```
<attribute name="ProviderUrl">
    tibjmsnaming://localhost:7243</attribute>
```

The new line creates an additional attribute ProviderUrl, that explicitly states the JNDI provider URL (rather than using the default built into the TIBCO Enterprise Message Service JBoss adapter class) with a port number of 7243 for SSL. Note that attribute names are case sensitive and must be entered exactly as shown above.

3. Modify the line you previously added to %JBOSS_DEPLOY%\jms\jms-ds.xml to explicitly specify the SSL port of 7243 in the JNDI LinkRef ToName:

```
<attribute name="ToName">
  tibjmsnaming://localhost:7243/SSLXAQueueConnectionFactory
</attribute>
```

4. Modify the line in file `standardjboss.xml` where you specified the TIBCO Enterprise Message Service JNDI name of the DLQ to explicitly specify the SSL port of 7243:

```
<DestinationQueue>tibjmsnaming://localhost:7243/queue/DLQ</DestinationQueue>
```

Stop and restart the JBoss server

You should see the same messages in the JBoss console during startup that you saw in the previous section.

Adding the SSL JAR Files to the CLASSPATH for the Client Program

The following JAR files, distributed with TIBCO Enterprise Message Service, must be added to the CLASSPATH of the client program, in the same manner that you added the non-SSL jar files to the CLASSPATH in the previous example:

```
jcirt.jar
jnet.jar
jsse.jar
tibcrypt.jar
```

Adding the SSL JNDI Properties for the Client Program

The following changes must be made to the file `%JBoss_CLIENT%\jndi.properties` that you modified in the previous section for the client:

1. Modify the provider url property to specify the SSL port number, as follows:

```
java.naming.provider.url=tibjmsnaming://localhost:7243
```

2. Add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```



Be sure there are no trailing spaces on either line above (particularly after `security_protocol=ssl`).

These properties specify that the "SSL" protocol should be used for JNDI lookups, and that host verification is turned off (the client will trust any host).

Modify and Rebuild the Client

Modify the client program (`SendRecvClient`) to look up `SSLXAQueueConnectionFactory` instead of `QueueConnectionFactory`. Rebuild the program.

Re-Run the Client Program

Run the client program as you did in the previous section. You should see the same output.

To prove that SSL communications are occurring, stop the EMS server, then restart it without SSL:

```
tibemsd -config tibemsd.conf
```

Then stop JBoss, then restart it. You should see the following exception in the JBoss console:

```
javax.jms.JMSEException: Failed to connect to the server at
ssl://localhost:7243
```

If you now run the test program again, you should see that it throws the same exception. This shows that when the TIBCO Enterprise Message Service server was set up to accept SSL connections, both clients successfully connected and communicated using SSL.

Alternatively, you could start the TIBCO Enterprise Message Service server from a command prompt window and turn SSL debug tracing on, as follows:

```
> tibemsd -ssl_debug_trace
```

Then when you restart JBoss and re-run the client program, you will see SSL debugging output on the `tibemsd` console window.

Container-Managed Transactions (XA)

The steps we have outlined in this chapter assume that the MDB uses container-managed transactions. This section highlights configuration details related to this feature.

Developer The MDB developer did not code transaction logic into the MDB, and specified this fact to application server in the file `ejb-jar.xml`. Namely, the `<transaction-type>` attribute has the value `Container` (see `ejb-jar.xml` on page 41). This fact has two consequences during deployment:

- XA Connection**
- JBoss 4.0.2 interprets this attribute value to indicate that the application requires an XA connection.
 - If your application indeed requires container-managed XA transactions, then this interpretation is correct, and no changes are required.
 - However, if your application does not use transactions at all, add this line to the `<message-driven>` element (see `jboss.xml` on page 41):

```
<xa-connection> false </xa-connection>
```

- XA Connection Factory**
- When we configured the connection factories in the EMS server, we created XA connection factories (see `Queues and Connection Factories` on page 45).
 - If your application indeed requires container-managed XA transactions, then this interpretation is correct, and no changes are required.
 - However, if your application does not use transactions at all, you can instead create connection factories that do not support XA:


```
> create factory QueueConnectionFactory queue
> create factory TopicConnectionFactory topic
```


Chapter 4 **Integrating With JBoss 3.2.3**

This chapter describes integrating TIBCO Enterprise Message Service with the JBoss J2EE application server, version 3.2.3. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside JBoss from any EMS client.

Topics

- *Overview of Integrating With JBoss 3.2.3, page 26*
- *Get the Example MDB Working Using JBossMQ, page 27*
- *Get the Example MDB Working Using TIBCO Enterprise Message Service, page 30*
- *Modify the Example to use SSL Communications, page 34*
- *Container-Managed Transactions (XA), page 38*

Overview of Integrating With JBoss 3.2.3

This chapter describes integrating TIBCO Enterprise Message Service with the JBoss J2EE application server, version 3.2.3. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside JBoss from any EMS client.

The techniques described in this chapter assume you are using Windows 2000, and that you have already downloaded and installed JBoss 3.2.3. It further assumes that you are running JBoss, the TIBCO Enterprise Message Service server, and the client program on the same machine.

Throughout the following procedures, environment variables are used to refer to specific directories within the JBoss installation. They are not actually needed by the JBoss server, but merely facilitate the reference to different directories in the JBoss installation.

The following environment variables are used throughout the discussion below:

```
JBOSS_HOME = C:\JBoss-3.2.3
JBOSS_CLIENT = %JBOSS_HOME%\client
JBOSS_DEPLOY = %JBOSS_HOME%\server\default\deploy
JBOSS_CONF = %JBOSS_HOME%\server\default\conf
```



The example in this chapter configures an MDB that uses container-managed transactions. For more information, see Container-Managed Transactions (XA) on page 53.

Get the Example MDB Working Using JBossMQ

1. To build the example MDB, add the following to your CLASSPATH:
`%JBOSS_CLIENT%\jboss-j2ee.jar`
2. Compile the example MDB, `TextMDB.java`. The source code for this example is located in *JBoss Administration and Development*, Chapter 4, Example 2.
3. Create a directory named `META-INF` in the output directory that now contains the `org.jboss.chap4.ex2.TextMDB.class`.
4. Copy the files `ejb-jar.xml` and `jboss.xml` from the source directory associated with Example 2 of Chapter 4 of *JBoss Administration and Development*, to the `META-INF` directory.
5. Create the EJB jar file by changing directories to the output directory and issuing the following command:

```
jar cvf myejb.jar META-INF org\jboss\chap4\ex2\TextMDB.class
```

6. Map Connection Factory Names

In JBoss 3.2.3, the JNDI names `QueueConnectionFactory` and `TopicConnectionFactory` do not come pre-mapped to the internal JBoss connection factory called `ConnectionFactory`. Therefore you must add this mapping in order to use the example MDB without modification. The mapping must be configured by adding the following lines in the file

`%JBOSS_DEPLOY%\jms\jbossmq-service.xml`:

```
<!--===== -->
<!-- JBossMQ -->
<!--===== -->

<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=QueueConnectionFactory">
  <attribute name="ToName">ConnectionFactory</attribute>
  <attribute name="FromName">QueueConnectionFactory</attribute>
</mbean>
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TopicConnectionFactory">
  <attribute name="ToName">ConnectionFactory</attribute>
  <attribute name="FromName">TopicConnectionFactory</attribute>
</mbean>
```

7. Start the JBoss server by changing to the `%JBOSS_HOME%\bin` directory and issuing the following command:
`run`
8. Copy `myejb.jar` to `%JBOSS_DEPLOY%`.
9. In the JBoss window, you should see output similar to the following:

```

19:35:33,343 INFO [org.jboss.deployment.MainDeployer] Starting deployment of
package: file:/C:/jboss-3.2.3/server/default/deploy/myejb.jar
19:35:34,296 INFO [org.jboss.ejb.EjbModule] Deploying TextMDB
19:35:34,875 INFO [org.jboss.ejb.plugins.jms.DLQHandler] Started null
19:35:34,875 INFO [org.jboss.ejb.plugins.jms.JMSContainerInvoker] Started
jboss.j2ee:binding=message-driven-bean,jndiName=local/TextMDB,plugin=invoker,service=EJB
19:35:34,875 INFO [org.jboss.ejb.plugins.MessageDrivenInstancePool] Started
jboss.j2ee:jndiName=local/TextMDB,plugin=pool,service=EJB
19:35:34,875 INFO [org.jboss.ejb.MessageDrivenContainer] Started
jboss.j2ee:jndiName=local/TextMDB,service=EJB
19:35:34,875 INFO [org.jboss.ejb.EjbModule] Started
jboss.j2ee:module=myejb.jar,service=EjbModule
19:35:34,875 INFO [org.jboss.ejb.EJBDeployer] Deployed:
file:/C:/jboss-3.2.3/server/default/deploy/myejb.jar
19:35:34,968 INFO [org.jboss.deployment.MainDeployer] Deployed package:
file:/C:/jboss-3.2.3/server/default/deploy/myejb.jar

```

10. To build the client program, make sure the following are in your CLASSPATH:

```

%JBOSS_CLIENT%\jboss-j2ee.jar
%JBOSS_CLIENT%\concurrent.jar

```

11. Compile the client program, `org.jboss.chap4.ex2.SendRecvClient.java`. The source to this program is listed in the book *JBoss Administration and Development*.

12. To run the client program, add the following to your CLASSPATH:

```

%JBOSS_CLIENT%\jnp-client.jar
%JBOSS_CLIENT%\jbossmq-client.jar
%JBOSS_CLIENT%\jboss-common-client.jar
%JBOSS_CLIENT%\jnet.jar
%JBOSS_CLIENT%\log4j.jar
%JBOSS_CLIENT%

```

The `%JBOSS_CLIENT%` directory is included so that the file `jndi.properties` in that directory can be found (see the next step).

13. In JBoss 3.2.3, a `jndi.properties` file does not come pre-configured for the client, therefore, you will have to create one. The easiest way is to first copy `jndi.properties` from `%JBOSS_CONF%` to `%JBOSS_CLIENT%`. Then add the following line in the copied file:

```
java.naming.provider.url=localhost
```

14. Run the client program:

```
java org.jboss.chap4.ex2.SendRecvClient
```

You should see output like the following in the client program window:

```

Begin sendRecvAsync
sendRecvAsync, sent text=A text msg#0
sendRecvAsync, sent text=A text msg#1
...
sendRecvAsync, sent text=A text msg#8
sendRecvAsync, sent text=A text msg#9

```

```

End sendRecvAsync
onMessage, recv text=A text msg#0processed by: 3824284
onMessage, recv text=A text msg#3processed by: 32953059
....
onMessage, recv text=A text msg#6processed by: 32420722
onMessage, recv text=A text msg#8processed by: 23916456

```

You should also see output like the following in the JBoss server console:

```

20:03:25,046 INFO [STDOUT] TextMDB.ctor, this=3824284
20:03:25,078 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=3824284
20:03:25,078 INFO [STDOUT] TextMDB.ejbCreate, this=3824284
20:03:25,109 INFO [STDOUT] TextMDB.ctor, this=2003839
20:03:25,109 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=2003839
20:03:25,109 INFO [STDOUT] TextMDB.ejbCreate, this=2003839
20:03:25,109 INFO [STDOUT] TextMDB.ctor, this=30170403
20:03:25,125 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=30170403
20:03:25,125 INFO [STDOUT] TextMDB.ejbCreate, this=30170403
20:03:25,109 INFO [STDOUT] TextMDB.ctor, this=32953059
20:03:25,140 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=32953059
20:03:25,140 INFO [STDOUT] TextMDB.ejbCreate, this=32953059
20:03:25,125 INFO [STDOUT] TextMDB.ctor, this=31834937
20:03:25,156 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=31834937
20:03:25,156 INFO [STDOUT] TextMDB.ejbCreate, this=31834937
20:03:25,187 INFO [STDOUT] TextMDB.onMessage, this=3824284
20:03:25,187 INFO [STDOUT] TextMDB.sendReply, this=3824284, dest=QUEUE.A
20:03:25,125 INFO [STDOUT] TextMDB.ctor, this=13863286
20:03:25,203 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=13863286
20:03:25,203 INFO [STDOUT] TextMDB.ejbCreate, this=13863286
20:03:25,203 INFO [STDOUT] TextMDB.onMessage, this=32953059
20:03:25,218 INFO [STDOUT] TextMDB.sendReply, this=32953059, dest=QUEUE.A
20:03:25,234 INFO [STDOUT] TextMDB.onMessage, this=3824284
20:03:25,234 INFO [STDOUT] TextMDB.sendReply, this=3824284, dest=QUEUE.A
20:03:25,218 INFO [STDOUT] TextMDB.ctor, this=32420722
20:03:25,250 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=32420722
20:03:25,250 INFO [STDOUT] TextMDB.ejbCreate, this=32420722
20:03:25,296 INFO [STDOUT] TextMDB.ctor, this=23916456
20:03:25,296 INFO [STDOUT] TextMDB.setMessageDrivenContext, this=23916456
20:03:25,296 INFO [STDOUT] TextMDB.ejbCreate, this=23916456
20:03:25,312 INFO [STDOUT] TextMDB.onMessage, this=31834937
20:03:25,312 INFO [STDOUT] TextMDB.sendReply, this=31834937, dest=QUEUE.A
20:03:25,328 INFO [STDOUT] TextMDB.onMessage, this=2003839
20:03:25,328 INFO [STDOUT] TextMDB.sendReply, this=2003839, dest=QUEUE.A
20:03:25,296 INFO [STDOUT] TextMDB.onMessage, this=32953059
20:03:25,343 INFO [STDOUT] TextMDB.sendReply, this=32953059, dest=QUEUE.A
20:03:25,343 INFO [STDOUT] TextMDB.onMessage, this=13863286
20:03:25,343 INFO [STDOUT] TextMDB.sendReply, this=13863286, dest=QUEUE.A
20:03:25,328 INFO [STDOUT] TextMDB.onMessage, this=30170403
20:03:25,343 INFO [STDOUT] TextMDB.sendReply, this=30170403, dest=QUEUE.A
20:03:25,375 INFO [STDOUT] TextMDB.onMessage, this=32420722
20:03:25,375 INFO [STDOUT] TextMDB.sendReply, this=32420722, dest=QUEUE.A
20:03:25,421 INFO [STDOUT] TextMDB.onMessage, this=23916456
20:03:25,421 INFO [STDOUT] TextMDB.sendReply, this=23916456, dest=QUEUE.A

```

Get the Example MDB Working Using TIBCO Enterprise Message Service



This example MDB uses container-managed transactions. For more information, see Container-Managed Transactions (XA) on page 53.

Queues and Connection Factories

1. Start the `tibemsd` server and the `tibemsadmin` console.
2. Create three queues (`queue/A`, `queue/B` and `queue/DLQ`) and two XA connection factories (`XAQueueConnectionFactory` and `XATopicConnectionFactory`), by entering the following commands in `tibemsadmin`:

```
> connect
> create queue queue/A
> create queue queue/B
> create queue queue/DLQ
> create factory XAQueueConnectionFactory xaqueue url=tcp://7222
> create factory XATopicConnectionFactory xatopic url=tcp://7222
```

3. Make a backup copy (in a separate directory) of the configuration files that will be changed:

```
%JBoss_DEPLOY%\jms\jms-ds.xml
%JBoss_CONF%\jboss-service.xml
%JBoss_CONF%\jndi.properties
%JBoss_CONF%\standardjboss.xml
```



You should copy the files in the `%JBoss_DEPLOY%` directory to another directory, rather than rename the files in place. JBoss attempts to deploy all files in that directory, regardless of name or file extension.

4. Add TIBCO EMS and the TIBCO EMS adapter class for JBoss to the CLASSPATH of the JBoss server by modifying the file `%JBoss_CONF%\jboss-service.xml` as described below. Substitute an appropriate JAR file CLASSPATH for your installation.

Add the following lines under the `<server>` element in the file `%JBoss_CONF%\jboss-service.xml`:

```
<!-- TIBCO Enterprise Message Service classpath -->
<classpath codebase="file:/C:/TIBCO/EMS/clients/java"
  archives="tibjms.jar, tibjmsapps.jar" />
```

5. Reconfigure the `JMSProviderLoader` mbean to load TIBCO Enterprise Message Service instead of JBoss MQ.

Remove the following lines from the file %JBASS_DEPLOY%\jms\jms-ds.xml:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.mq:service=JMSProviderLoader,name=JBossMQProvider">
  <attribute name="ProviderName">DefaultJMSProvider</attribute>
  <attribute name="ProviderAdapterClass">
    org.jboss.jms.jndi.JBossMQProvider
  </attribute>
  <attribute name="QueueFactoryRef">java:/XAConnectionFactory</attribute>
  <attribute name="TopicFactoryRef">java:/XAConnectionFactory</attribute>
</mbean>
```

Replace those removed lines with the following lines to cause JMSProviderLoader mbean to load TIBCO Enterprise Message Service:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.mq:service=JMSProviderLoader,name=TibjmsProvider">
  <attribute name="ProviderName">TIBCOJMSProvider</attribute>
  <attribute name="ProviderAdapterClass">
    com.tibco.tibjms.appserver.jboss.JBossAdapter
  </attribute>
  <attribute name="QueueFactoryRef">XAQueueConnectionFactory</attribute>
  <attribute name="TopicFactoryRef">XATopicConnectionFactory</attribute>
</mbean>
```

- When the sample MDB looks up the QueueConnectionFactory (in order to create a QueueSender to send a message back to the initiator), it looks it up as java:comp/env/jms/QCF. Because we want the QueueConnectionFactory object that is returned from the lookup to be a TIBCO Enterprise Message Service XAQueueConnectionFactory, we must store a JNDI LinkRef under that name in the JBoss JNDI implementation that points to the XAQueueConnectionFactory in the TIBCO Enterprise Message Service implementation. Adding the following lines in %JBASS_DEPLOY%\jms\jms-ds.xml accomplishes this.

```
<!-- Redirect QueueConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=QueueConnectionFactory">
  <attribute name="ToName">tibjmsnaming://localhost/XAQueueConnectionFactory
  </attribute>
  <attribute name="FromName">QueueConnectionFactory</attribute>
</mbean>
```

- When the SendRecvClient test program looks up the ConnectionFactory referenced in the test program, it needs to be redirected to the appropriate QueueConnectionFactory configured in EMS. Add the following lines in %JBASS_DEPLOY%\jms\jms-ds.xml to accomplish this.

```
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=ConnectionFactory">
```

```

    <attribute name="ToName">tibjmsnaming://localhost/QueueConnectionFactory
  </attribute>
  <attribute name="FromName">ConnectionFactory</attribute>
</mbean>

```

8. When the JBoss server invokes JNDI and encounters the `tibjmsnaming` scheme, the server must be able to find the TIBCO Enterprise Message Service `URLConnectionFactory`. Therefore, modify the file `%JBOSS_CONF%\jndi.properties` as follows:

Change:

```
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

To

```
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces:com.tibco.tibjms.naming
```

9. In the `%JBOSS_CONF%\standardjboss.xml` file, modify the following line.

Change:

```

<JMSProviderAdapterJNDI>DefaultJMSProvider
</JMSProviderAdapterJNDI>

```

To:

```

<JMSProviderAdapterJNDI>TIBCOJMSProvider
</JMSProviderAdapterJNDI>

```

This change sets "TIBCOJMSProvider" as the JMS Provider Adapter JNDI name.

10. In the `%JBOSS_CONF%\standardjboss.xml` file, modify the following line.

Change:

```
<DestinationQueue>queue/DLQ</DestinationQueue>
```

To:

```

<DestinationQueue>
  tibjmsnaming://localhost/queue/DLQ
</DestinationQueue>

```

This change specifies the TIBCO Enterprise Message Service JNDI name for the MDB Dead Letter Queue (DLQ) `queue/DLQ`.

11. Move the following files out of the `%JBOSS_DEPLOY%` directory. These files are not needed when using TIBCO Enterprise Message Service and therefore must not be deployed:


```
%JBoss_DEPLOY%\jms\jbossmq-service.xml  
%JBoss_DEPLOY%\jms\jbossmq-destinations-service.xml
```

12. Stop the JBoss server, then it restart by entering:

```
run
```

13. When the client program invokes JNDI, it should use the TIBCO Enterprise Message Service JNDI server. Modify %JBoss_CLIENT%\jndi.properties to use TIBCO Enterprise Message Service JNDI by setting the following property:

```
java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory
```

14. Add C:\TIBCO\EMS\clients\java\tibjms.jar to the CLASSPATH of the client program.
15. Run the client program as you did in the previous section. You should see the same output.

Modify the Example to use SSL Communications

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, JBoss, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Adding the SSL JAR Files to the CLASSPATH for the JBoss Server

Add the TIBCO `tibcrypt.jar` file to the CLASSPATH of the JBoss server by modifying the file `%JBOSS_CONF%\jboss-service.xml` as described below. Substitute an appropriate JAR file CLASSPATH for your installation.

Add the following line under the `<server>` element in `%JBOSS_CONF%\jboss-service.xml`:

```
<classpath codebase="file:/C:\TIBCO\EMS\clients\java"
    archives="tibcrypt.jar" />
```

Configuring the TIBCO Enterprise Message Service Server for SSL

1. Start `tibemspd` in the working directory `C:\TIBCO\ems\bin` as follows:

```
tibemspd -config tibemspdssl.conf
```

When `tibemspd` starts you should see messages like the following in the console window, confirming SSL is enabled:

```
17:09:03 Secure Socket Layer is enabled, using OpenSSL 0.9.7c.
17:09:03 Accepting connections on tcp://localhost:7222.
17:09:03 Accepting connections on ssl://localhost:7243.
17:09:03 Server is active.
```

2. Start `tibemspadmin` (administration tool) and enter the following commands.

First, create a new `XAQueueConnectionFactory` that establishes SSL connections:

```
create factory SSLXAQueueConnectionFactory xaqueue url=ssl://7243
```

Second, disable host verification for connections that this connection factory creates:

```
setprop factory SSLXAQueueConnectionFactory ssl_verify_host=disabled
```

This is the simplest SSL configuration.

Configuring JBoss for SSL-based JMS Communications

There are two aspects to SSL communications between JBoss and the TIBCO EMS server. The first is for messaging between the JBoss and TIBCO servers to occur over SSL. The second is for JNDI lookups from JBoss to the TIBCO JNDI provider to occur over SSL. The following two sections separately describe the required steps for each.

JMS Messaging over SSL

1. Modify the line you added to %JBOSS_DEPLOY%\jms\jms-ds.xml in the previous section (which specifies the QueueFactoryRef attribute of the JMS ProviderLoader) to be the new connection factory you just created (which establishes SSL connections):

```
<attribute name="QueueFactoryRef">
    SSLXAQueueConnectionFactory
</attribute>
```

2. Modify the line you added to %JBOSS_DEPLOY%\jms\jms-ds.xml in the previous section so it creates the JNDI LinkRef ToName to the new SSL connection factory—namely SSLXAQueueConnectionFactory (which you created above):

```
<attribute name="ToName">
    tibjmsnaming://localhost/SSLXAQueueConnectionFactory
</attribute>
```

JNDI Lookups over SSL

1. In the file %JBOSS_CONF%\jndi.properties, add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```

These properties specify the SSL protocol for JNDI lookups, and disable host verification.

2. Add the following line in the JMSProviderLoader mbean in %JBOSS_DEPLOY%\jms\jms-ds.xml:

```
<attribute name="ProviderUrl">
    tibjmsnaming://localhost:7243</attribute>
```

The new line creates an additional attribute ProviderUrl, that explicitly states the JNDI provider URL (rather than using the default built into the TIBCO Enterprise Message Service JBoss adapter class) with a port number of 7243 for SSL. Note that attribute names are case sensitive and must be entered exactly as shown above.

3. Modify the line you previously added to %JBOSS_DEPLOY%\jms\jms-ds.xml to explicitly specify the SSL port of 7243 in the JNDI LinkRef ToName:

```
<attribute name="ToName">
  tibjmsnaming://localhost:7243/SSLXAQueueConnectionFactory
</attribute>
```

4. Modify the line in file `standardjboss.xml` where you specified the TIBCO Enterprise Message Service JNDI name of the DLQ to explicitly specify the SSL port of 7243:

```
<DestinationQueue>tibjmsnaming://localhost:7243/queue/DLQ</DestinationQueue>
```

Stop and restart the JBoss server

You should see the same messages in the JBoss console during startup that you saw in the previous section.

Adding the SSL JAR Files to the CLASSPATH for the Client Program

The following JAR files, distributed with TIBCO Enterprise Message Service, must be added to the CLASSPATH of the client program, in the same manner that you added the non-SSL jar files to the CLASSPATH in the previous example:

```
jcirt.jar
jnet.jar
jsse.jar
tibcrypt.jar
```

Adding the SSL JNDI Properties for the Client Program

The following changes must be made to the file `%JBoss_CLIENT%\jndi.properties` that you modified in the previous section for the client:

1. Modify the provider url property to specify the SSL port number, as follows:

```
java.naming.provider.url=tibjmsnaming://localhost:7243
```

2. Add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```



Be sure there are no trailing spaces on either line above (particularly after `security_protocol=ssl`).

These properties specify that the "SSL" protocol should be used for JNDI lookups, and that host verification is turned off (the client will trust any host).

Modify and Rebuild the Client

Modify the client program (`SendRecvClient`) to look up `SSLXAQueueConnectionFactory` instead of `QueueConnectionFactory`. Rebuild the program.

Re-Run the Client Program

Run the client program as you did in the previous section. You should see the same output.

To prove that SSL communications are occurring, stop the EMS server, then restart it without SSL:

```
tibemsd -config tibemsd.conf
```

Then stop JBoss, then restart it. You should see the following exception in the JBoss console:

```
javax.jms.JMSEException: Failed to connect to the server at  
ssl://localhost:7243
```

If you now run the test program again, you should see that it throws the same exception. This shows that when the TIBCO Enterprise Message Service server was set up to accept SSL connections, both clients successfully connected and communicated using SSL.

Alternatively, you could start the TIBCO Enterprise Message Service server from a command prompt window and turn SSL debug tracing on, as follows:

```
> tibemsd -ssl_debug_trace
```

Then when you restart JBoss and re-run the client program, you will see SSL debugging output on the `tibemsd` console window.

Container-Managed Transactions (XA)

The steps we have outlined in this chapter assume that the MDB uses container-managed transactions. This section highlights configuration details related to this feature.

Developer The MDB developer did not code transaction logic into the MDB, and specified this fact to application server in the file `ejb-jar.xml`. Namely, the `<transaction-type>` attribute has the value `Container` (see `ejb-jar.xml` on page 41). This fact has two consequences during deployment:

- XA Connection**
- JBoss 3.2.3 interprets this attribute value to indicate that the application requires an XA connection.
 - If your application indeed requires container-managed XA transactions, then this interpretation is correct, and no changes are required.
 - However, if your application does not use transactions at all, add this line to the `<message-driven>` element (see `jboss.xml` on page 41):

```
<xa-connection> false </xa-connection>
```

- XA Connection Factory**
- When we configured the connection factories in the EMS server, we created XA connection factories (see *Queues and Connection Factories* on page 45).
 - If your application indeed requires container-managed XA transactions, then this interpretation is correct, and no changes are required.
 - However, if your application does not use transactions at all, you can instead create connection factories that do not support XA:


```
> create factory QueueConnectionFactory queue
> create factory TopicConnectionFactory topic
```

Chapter 5

Integrating With JBoss 3.0.4

This chapter describes integrating TIBCO Enterprise Message Service with the JBoss J2EE application server, version 3.0.4. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside JBoss from any EMS client.

Topics

- *Overview of Integrating With JBoss 3.0.4, page 40*
- *Get the Example MDB Working Using JBossMQ, page 41*
- *Get the Example MDB Working Using TIBCO Enterprise Message Service, page 45*
- *Modify the Example to use SSL Communications, page 49*
- *Container-Managed Transactions (XA), page 53*

Overview of Integrating With JBoss 3.0.4

This chapter describes integrating TIBCO Enterprise Message Service with the JBoss J2EE application server, version 3.0.4. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside JBoss from any EMS client.

The techniques described in this chapter assume you are using Windows 2000, and that you have already downloaded and installed JBoss 3.0.4. It further assumes that you are running JBoss, the TIBCO Enterprise Message Service server, and the client program on the same machine.

Throughout the following procedures, environment variables are used to refer to specific directories within the JBoss installation. They are not actually needed by the JBoss server, but merely facilitate the reference to different directories in the JBoss installation.

The following environment variables are used throughout the discussion below:

```
JBOSS_HOME = C:\JBoss-3.0.4
JBOSS_CLIENT = %JBOSS_HOME%\client
JBOSS_DEPLOY = %JBOSS_HOME%\server\default\deploy
JBOSS_CONF = %JBOSS_HOME%\server\default\conf
```



The example in this chapter configures an MDB that uses container-managed transactions. For more information, see Container-Managed Transactions (XA) on page 53.

Get the Example MDB Working Using JBossMQ

1. To build the example MDB, add the following to your CLASSPATH:
`%JBOSS_CLIENT%\jboss-j2ee.jar`
2. Compile the example MDB, `TextMDB.java`. The source code for this example is located in *JBoss Administration and Development*, Chapter 4, Example 2.
3. Create a directory named `META-INF` in the output directory that now contains the `org.jboss.chap4.ex2.TextMDB.class`.
4. Create the `ejb-jar.xml` and `jboss.xml` files in the `META-INF` directory. The code for these files is listed in *JBoss Administration and Development*, but the code below has slight modifications.

ejb-jar.xml

```
<?xml version="1.0"?>
<!-- The ejb-jar.xml descriptor -->
<!DOCTYPE ejb-jar
    PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
        JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd" >

<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>TextMDB</ejb-name>
      <ejb-class>org.jboss.chap4.ex2.TextMDB</ejb-class>
      <transaction-type>Container</transaction-type>
      <acknowledge-mode>AUTO_ACKNOWLEDGE</acknowledge-mode>
      <message-driven-destination>
        <destination-type>javax.jms.Queue</destination-type>
      </message-driven-destination>
      <resource-ref>
        <res-ref-name>jms/QCF</res-ref-name>
        <res-type>javax.jms.QueueConnectionFactory</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

jboss.xml

```
<?xml version="1.0"?>
<!-- The jboss.xml descriptor -->
<jboss>
  <enterprise-beans>
    <message-driven>
      <ejb-name>TextMDB</ejb-name>
      <destination-jndi-name>queue/B</destination-jndi-name>
      <resource-ref>
```

```

        <res-ref-name>jms/QCF</res-ref-name>
        <jndi-name>QueueConnectionFactory</jndi-name>
    </resource-ref>
</message-driven>
</enterprise-beans>
</jboss>

```

5. Create the EJB jar file by changing directories to the output directory and issuing the following command:

```
jar cvf myejb.jar META-INF org\jboss\chap4\ex2\TextMDB.class
```

6. Map Connection Factory Names

In JBoss 3.0.4, the JNDI names `QueueConnectionFactory` and `TopicConnectionFactory` do not come pre-mapped to the internal JBoss connection factory called `ConnectionFactory`. Therefore you must add this mapping in order to use the example MDB without modification. The mapping must be configured by adding the following lines in the file `%JBOSS_DEPLOY%\jbossmq-service.xml`:

```

<!--===== -->
<!-- JBossMQ -->
<!--===== -->

<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=QueueConnectionFactory">
    <attribute name="ToName">ConnectionFactory</attribute>
    <attribute name="FromName">QueueConnectionFactory</attribute>
</mbean>
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TopicConnectionFactory">
    <attribute name="ToName">ConnectionFactory</attribute>
    <attribute name="FromName">TopicConnectionFactory</attribute>
</mbean>

```

7. Start the JBoss server by changing to the `%JBOSS_HOME%\bin` directory and issuing the following command:

```
run
```

8. Copy `myejb.jar` to `%JBOSS_DEPLOY%`.
9. In the JBoss window, you should see output similar to the following:

```

12:04:09,172 INFO  [EjbModule] Creating
12:04:09,192 INFO  [EjbModule] Deploying TextMDB
12:04:10,214 WARN  [SecurityManager] No SecurityMetadadata was
available for B adding default security conf
12:04:10,284 INFO  [EjbModule] Created
12:04:10,294 INFO  [EjbModule] Starting
12:04:10,304 INFO  [EjbModule] Started
12:04:10,304 INFO  [MainDeployer]
deployment of package:
file:/C:/jboss-3.0.0/server/default/deploy/myejb.jar

```

10. To build the client program, make sure the following are in your CLASSPATH:

```
%JBoss_CLIENT%\jboss-j2ee.jar
%JBoss_CLIENT%\concurrent.jar
```

11. Compile the client program, `org.jboss.chap4.ex2.SendRecvClient.java`. The source to this program is listed in *JBoss Administration and Development*.
12. To run the client program, add the following to your CLASSPATH:

```
%JBoss_CLIENT%\jnp-client.jar
%JBoss_CLIENT%\jbossmq-client.jar
%JBoss_CLIENT%\jboss-common-client.jar
%JBoss_CLIENT%\jnet.jar
%JBoss_CLIENT%\log4j.jar
%JBoss_HOME%\client
```

The `%JBoss_HOME%\client` directory is included so that the file `jndi.properties` in that directory can be found.

In JBoss 3.0.4, a `jndi.properties` does not come pre-configured for the client, therefore, you will have to create one. The easiest way is to first copy `jndi.properties` from `%JBoss_CONF%` to `%JBoss_CLIENT%`. Then, un-comment the following line in the copied file:

```
java.naming.provider.url=localhost
```

13. Run the client program:

```
java org.jboss.chap4.ex2.SendRecvClient
```

You should see output like the following in the client program window:

```
Begin sendRecvAsync
sendRecvAsync, sent text=A text msg#0
sendRecvAsync, sent text=A text msg#1
sendRecvAsync, sent text=A text msg#2
sendRecvAsync, sent text=A text msg#3
...
End sendRecvAsync
onMessage, recv text=A text msg#0processed by: 7438914
onMessage, recv text=A text msg#1processed by: 1639412
onMessage, recv text=A text msg#2processed by: 10668
onMessage, recv text=A text msg#3processed by: 1611150
...
```

You should also see output like the following in the JBoss server console:

```
[INFO,Default] TextMDB.ctor, this=7438914
[INFO,Default] TextMDB.setMessageDrivenContext, this=7438914
[INFO,Default] TextMDB.ejbCreate, this=7438914
[INFO,Default] TextMDB.ctor, this=1639412
[INFO,Default] TextMDB.setMessageDrivenContext, this=1639412
[INFO,Default] TextMDB.ejbCreate, this=1639412
[INFO,Default] TextMDB.onMessage, this=7438914
[INFO,Default] TextMDB.sendReply, this=7438914, dest=QUEUE.A
[INFO,Default] TextMDB.ctor, this=10668
[INFO,Default] TextMDB.setMessageDrivenContext, this=10668
[INFO,Default] TextMDB.ejbCreate, this=10668
[INFO,Default] TextMDB.onMessage, this=1639412
```

```

[INFO,Default] TextMDB.sendReply, this=1639412, dest=QUEUE.A
[INFO,Default] TextMDB.ctor, this=1611150
[INFO,Default] TextMDB.setMessageDrivenContext, this=1611150
[INFO,Default] TextMDB.ejbCreate, this=1611150
[INFO,Default] TextMDB.onMessage, this=10668
[INFO,Default] TextMDB.sendReply, this=10668, dest=QUEUE.A
[INFO,Default] TextMDB.ctor, this=6808485
[INFO,Default] TextMDB.setMessageDrivenContext, this=6808485
[INFO,Default] TextMDB.ejbCreate, this=6808485
[INFO,Default] TextMDB.onMessage, this=1611150
[INFO,Default] TextMDB.sendReply, this=1611150, dest=QUEUE.A
[INFO,Default] TextMDB.onMessage, this=6808485
[INFO,Default] TextMDB.sendReply, this=6808485, dest=QUEUE.A
[INFO,Default] TextMDB.ctor, this=3277650
[INFO,Default] TextMDB.setMessageDrivenContext, this=3277650
[INFO,Default] TextMDB.ejbCreate, this=3277650
[INFO,Default] TextMDB.ctor, this=5224450
[INFO,Default] TextMDB.setMessageDrivenContext, this=5224450
[INFO,Default] TextMDB.ejbCreate, this=5224450
[INFO,Default] TextMDB.onMessage, this=3277650
[INFO,Default] TextMDB.sendReply, this=3277650, dest=QUEUE.A
[INFO,Default] TextMDB.ctor, this=4280406
[INFO,Default] TextMDB.setMessageDrivenContext, this=4280406
[INFO,Default] TextMDB.ejbCreate, this=4280406
[INFO,Default] TextMDB.onMessage, this=5224450
[INFO,Default] TextMDB.sendReply, this=5224450, dest=QUEUE.A
[INFO,Default] TextMDB.ctor, this=4977982
[INFO,Default] TextMDB.onMessage, this=4280406
[INFO,Default] TextMDB.sendReply, this=4280406, dest=QUEUE.A
[INFO,Default] TextMDB.setMessageDrivenContext, this=4977982
[INFO,Default] TextMDB.ejbCreate, this=4977982
[INFO,Default] TextMDB.ctor, this=6805499
[INFO,Default] TextMDB.setMessageDrivenContext, this=6805499
[INFO,Default] TextMDB.ejbCreate, this=6805499
[INFO,Default] TextMDB.onMessage, this=4977982
[INFO,Default] TextMDB.sendReply, this=4977982, dest=QUEUE.A
[INFO,Default] TextMDB.onMessage, this=6805499
[INFO,Default] TextMDB.sendReply, this=6805499, dest=QUEUE.A

```

Get the Example MDB Working Using TIBCO Enterprise Message Service



This example MDB uses container-managed transactions. For more information, see Container-Managed Transactions (XA) on page 53.

Queues and Connection Factories

1. Start the `tibemsd` server and the `tibemsadmin` console.
2. Create three queues (`queue/A`, `queue/B` and `queue/DLQ`) and two XA connection factories (`XAQueueConnectionFactory` and `XATopicConnectionFactory`), by entering the following commands in `tibemsadmin`:

```
> connect
> create queue queue/A
> create queue queue/B
> create queue queue/DLQ
> create factory XAQueueConnectionFactory xaqueue
> create factory XATopicConnectionFactory xatopic
```

3. Make a backup copy of the configuration files that will be changed:

```
%JBOSS_DEPLOY%\jms-service.xml
%JBOSS_CONF%\jboss-service.xml
%JBOSS_CONF%\jndi.properties
%JBOSS_CONF%\standardjboss.xml
```



You should copy the files in the `%JBOSS_DEPLOY%` directory to another directory, rather than rename the files in place. JBoss attempts to deploy all files in that directory, regardless of name or file extension.

4. Add TIBCO Enterprise Message Service and the TIBCO Enterprise Message Service adapter class for JBoss to the `CLASSPATH` of the JBoss server by modifying the file described below. The value of the jar file paths should be modified for your installation.

Add the following line under the `<server>` element in `jms-service.xml`:

```
<!-- TIBCO Enterprise Message Service classpath -->
<classpath codebase="file://C:\TIBCO\EMS\clients\java"
           archives="tibjms.jar, tibjmsapps.jar" />
```

5. Remove the configuration that causes the `JMSProviderLoader` mbean to load JBoss MQ.

In the file `jms-service.xml`, remove the following lines:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.mq:service=JMSProviderLoader,name=JBossMQProvider">
  <attribute name="ProviderName">DefaultJMSProvider</attribute>
  <attribute name="ProviderAdapterClass">
    org.jboss.jms.jndi.JBossMQProvider
  </attribute>
  <attribute name="QueueFactoryRef">java:/XAConnectionFactory</attribute>
  <attribute name="TopicFactoryRef">java:/XAConnectionFactory</attribute>
</mbean>
```

Replace the removed lines with the following lines to cause JMSProviderLoader mbean to load TIBCO Enterprise Message Service:

In the file `jms-service.xml` add the following lines:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.mq:service=JMSProviderLoader,name=TibjmsProvider">
  <attribute name="ProviderName">TIBCOJMSProvider</attribute>
  <attribute name="ProviderAdapterClass">
    com.tibco.tibjms.appserver.jboss.JBossAdapter
  </attribute>
  <attribute name="QueueFactoryRef">XAQueueConnectionFactory</attribute>
  <attribute name="TopicFactoryRef">XATopicConnectionFactory</attribute>
</mbean>
```

- When the sample MDB looks up the QueueConnectionFactory (in order to create a QueueSender to send a message back to the initiator), it looks it up as `java:comp/env/jms/QCF`. Because we want the QueueConnectionFactory object that is returned from the lookup to be a TIBCO Enterprise Message Service XAQueueConnectionFactory, we must store a JNDI LinkRef under that name in the JBoss JNDI implementation that points to the XAQueueConnectionFactory in the TIBCO Enterprise Message Service implementation. Adding the following lines in `jms-service.xml` accomplishes this.

```
<!-- Redirect QueueConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=QueueConnectionFactory">
  <attribute name="ToName">tibjmsnaming://localhost/XAQueueConnectionFactory
  </attribute>
  <attribute name="FromName">QueueConnectionFactory</attribute>
</mbean>
```

- In `jboss-service.xml`, change the dependency of the EJB Deployer from JBossMQProvider to TibjmsProvider.

Change:

```
<depends>jboss.mq:service=JMSProviderLoader,name=JBossMQProvider</depends>
```

To:

```
<depends>jboss.mq:service=JMSProviderLoader,name=TibjmsProvider</depends>
```

8. When the JBoss server invokes JNDI and encounters the `tibjmsnaming` scheme, the server must be able to find the TIBCO Enterprise Message Service `URLConnectionFactory`. Therefore, modify the file `%JBOSS_CONF%\jndi.properties` as follows:

Change:

```
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

To

```
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces:com.tibco.tibjms.naming
```

9. In the `%JBOSS_CONF%\standardjboss.xml` file, modify the following line. This sets "TIBCOJMSProvider" as the JMS Provider Adapter JNDI name:

Change:

```
<JMSProviderAdapterJNDI>DefaultJMSProvider
</JMSProviderAdapterJNDI>
```

To:

```
<JMSProviderAdapterJNDI>TIBCOJMSProvider
</JMSProviderAdapterJNDI>
```

10. In the `%JBOSS_CONF%\standardjboss.xml` file, modify the following line. This specifies the TIBCO Enterprise Message Service JNDI name for the MDB Dead Letter Queue (DLQ) `queue/DLQ`:

Change:

```
<DestinationQueue>queue/DLQ</DestinationQueue>
```

To:

```
<DestinationQueue>
  tibjmsnaming://localhost/queue/DLQ
</DestinationQueue>
```

11. Move the following files out of the `%JBOSS_DEPLOY%` directory. These files are not needed when using TIBCO Enterprise Message Service and therefore must not be deployed:

```
%JBOSS_DEPLOY%\jbossmq-service.xml
%JBOSS_DEPLOY%\jbossmq-destinations-service.xml
```

12. Stop and restart the JBoss server by entering:

run

13. When the client program invokes JNDI, it should use the TIBCO Enterprise Message Service JNDI server. Modify %JBoss_CLIENT%\jndi.properties to use TIBCO Enterprise Message Service JNDI by setting the following property:

```
java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory
```

14. Add C:\TIBCO\EMS\clients\java\tibjms.jar to the CLASSPATH of the client program.
15. Run the client program as you did in the previous section. You should see the same output.

Modify the Example to use SSL Communications

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, JBoss, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Adding the SSL JAR Files to the CLASSPATH for the JBoss Server

The following JAR files, distributed with TIBCO Enterprise Message Service, must be added to the CLASSPATH of the JBoss server, in the same manner that you added the non-SSL jar files to the JBoss CLASSPATH in the previous example:

```
jcirt.jar
jnet.jar
jsse.jar
tibcrypt.jar
```

Add the following line under the <server> element in `jms-service.xml`:

```
<classpath codebase="file://C:\TIBCO\EMS\clients\java"
    archives="jcirt.jar, jnet.jar, jsse.jar, tibcrypt.jar" />
```

Configuring the TIBCO Enterprise Message Service Server for SSL

1. Enter the following commands in `tibemsadmin`:

```
>create factory SSLXAQueueConnectionFactory xaqueue url=ssl://7243
```

This creates a new `XAQueueConnectionFactory` that establishes SSL connections.

```
>setprop factory SSLXAQueueConnectionFactory
ssl_verify_host=disabled
```

This turns off host verification for connections created by this connection factory. This is the simplest SSL configuration.

2. In `C:\Tibco\EMS\bin\tibemsd.conf`, add the following lines:

```
listen = ssl://localhost:7243

ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password = password

listen = tcp://localhost:7222
```

These lines explicitly set the `tcp` and `ssl` listen ports and specify the three required server-side SSL parameters: identity, private key, and password.

Save the file, stop and restart the TIBCO Enterprise Message Service server. When it restarts you should see messages like the following in the console window confirming SSL is enabled:

```
2002-03-19 13:48:34 Secure Socket Layer is enabled.
2002-03-19 13:48:34 Accepting connections on
ssl://localhost:7243.
2002-03-19 13:48:34 Accepting connections on
tcp://localhost:7222.
```

Configuring JBoss for SSL-based JMS Communications

There are two aspects to SSL communications between JBoss and the TIBCO EMS server. The first is for messaging between the JBoss and TIBCO servers to occur over SSL. The second is for JNDI lookups from JBoss to the TIBCO JNDI provider to occur over SSL. The following two sections separately describe the required steps for each.

JMS Messaging over SSL

1. Modify the line you added to `jms-service.xml` in the previous section which specifies the `QueueFactoryRef` attribute of the JMS Provider Loader to be the be the new connection factory you just created that establishes SSL connections:

```
<attribute name="QueueFactoryRef">
    SSLXAQueueConnectionFactory</attribute>
```

2. Modify the line you added to `jms-service.xml` in the previous section to create the JNDI LinkRef ToName to the new connection factory:

```
<attribute name="ToName">
    tibjmsnaming://localhost/SSLXAQueueConnectionFactory
</attribute>
```

and the name is changed to `SSLXAQueueConnectionFactory`, the SSL-based `XAQueueConnectionFactory` that you just created.

JNDI Lookups over SSL



The following steps arrange for the JBoss server to do JNDI lookup using SSL. However, a defect in JBoss 3.0.0 (and later releases) requires TCP for JNDI lookup—SSL is not currently available. In the meantime, we retain these instructions, as we expect a future JBoss release to correct this defect.

1. In the file `%JBOSS_CONF%\jndi.properties`, add the following line:
`com.tibco.tibjms.naming.security_protocol=ssl`

This property specifies that the "SSL" protocol should be used for JNDI lookups.

2. Add the following line in the `JMSProviderLoader` mbean in `jms-service.xml`:

```
<attribute name="ProviderUrl">
    tibjmsnaming://localhost:7243</attribute>
```

The new line creates an additional attribute `ProviderUrl`, that explicitly states the JNDI provider URL (rather than using the default built into the TIBCO Enterprise Message Service JBoss adapter class) with a port number of 7243 for SSL. Note that attribute names are case sensitive and must be entered exactly as shown above.

3. Modify the line you previously added to `jms-service.xml` to explicitly specify the SSL port of 7243 in the JNDI LinkRef `ToName`:

```
<attribute name="ToName">
    tibjmsnaming://localhost:7243/SSLXAQueueConnectionFactory
</attribute>
```

4. Modify the line in file `standardjboss.xml` where you specified the TIBCO Enterprise Message Service JNDI name of the DLQ to explicitly specify the SSL port of 7243:

```
<DestinationQueue>tibjmsnaming://localhost:7243/queue/DLQ</DestinationQueue>
```

Stop and restart the JBoss server

You should see the same messages in the JBoss console during startup that you saw in the previous section.

Adding the SSL JAR Files to the CLASSPATH for the Client Program

The following JAR files, distributed with TIBCO Enterprise Message Service, must be added to the `CLASSPATH` of the client program, in the same manner that you added the non-SSL jar files to the `CLASSPATH` in the previous example:

```
jcert.jar
jnet.jar
jsse.jar
tibcrypt.jar
```

Adding the SSL JNDI Properties for the Client Program

The following changes must be made to the file `%JBoss_CLIENT%\jndi.properties` that you modified in the previous section for the client:

1. Modify the provider url property to specify the SSL port number, as follows:

```
java.naming.provider.url=tibjmsnaming://localhost:7243
```

2. Add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```



Be sure there are no trailing spaces on either line above (particularly after `security_protocol=ssl`).

These properties specify that the "SSL" protocol should be used for JNDI lookups, and that host verification is turned off (the client will trust any host).

Modify and Rebuild the Client

Modify the client program (`SendRecvClient`) to look up `SSLXAQueueConnectionFactory` instead of `QueueConnectionFactory`. Rebuild the program.

Re-Run the Client Program

Run the client program as you did in the previous section. You should see the same output.

To prove that SSL communications are occurring, you could remove the SSL settings you added to `tibemsd.conf` in *Configuring the TIBCO Enterprise Message Service Server for SSL* on page 49, and restart the TIBCO Enterprise Message Service server. Then stop and restart JBoss. You should see the following exception in the JBoss console:

```
javax.jms.JMSEException: Failed to connect to the server at
ssl://localhost:7243
```

If you now run the test program again, you should see that it throws the same exception. This shows that when the TIBCO Enterprise Message Service server was set up to accept SSL connections, both clients successfully connected and communicated using SSL.

Alternatively, you could start the TIBCO Enterprise Message Service server from a command prompt window and turn SSL debug tracing on, as follows:

```
> tibemsd -ssl_debug_trace
```

Then when you restart JBoss and re-run the client program, you will see SSL debugging output on the `tibemsd` console window.

Container-Managed Transactions (XA)

The steps we have outlined in this chapter assume that the MDB uses container-managed transactions. This section highlights configuration details related to this feature.

- | | |
|--------------------------|---|
| Developer | <p>The MDB developer did not code transaction logic into the MDB, and specified this fact to application server in the file <code>ejb-jar.xml</code>. Namely, the <code><transaction-type></code> attribute has the value <code>Container</code> (see <code>ejb-jar.xml</code> on page 41). This fact has two consequences during deployment:</p> |
| XA Connection | <ul style="list-style-type: none"> • JBoss 3.0.4 interprets this attribute value to indicate that the application requires an XA connection. <ul style="list-style-type: none"> — If your application indeed requires container-managed XA transactions, then this interpretation is correct, and no changes are required. — However, if your application does not use transactions at all, add this line to the <code><message-driven></code> element (see <code>jboss.xml</code> on page 41): <pre style="margin-left: 40px;"><xa-connection> false </xa-connection></pre> |
| XA Connection
Factory | <ul style="list-style-type: none"> • When we configured the connection factories in the EMS server, we created XA connection factories (see <code>Queues and Connection Factories</code> on page 45). <ul style="list-style-type: none"> — If your application indeed requires container-managed XA transactions, then this interpretation is correct, and no changes are required. — However, if your application does not use transactions at all, you can instead create connection factories that do not support XA: <pre style="margin-left: 40px;">> create factory QueueConnectionFactory queue > create factory TopicConnectionFactory topic</pre> |

Integrating With Borland Enterprise Server

5.1

This chapter describes integrating TIBCO Enterprise Message Service with Borland Enterprise Server (BES) 5.1. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside Borland Enterprise Server with a J2EE application client.

Borland Enterprise Server has an example MDB named “Hello Message-Driven Beans Example”. The example includes a simple MDB and J2EE application client program. The example illustrates how to trigger the MDB within Borland Enterprise Server using the external client program (using SonicMQ as the JMS provider).

This chapter demonstrates using that same example with TIBCO Enterprise Message Service as the JMS provider. Also, instructions are given on how to convert the example to support container-managed XA transactions in which TIBCO Enterprise Message Service can participate as an XA resource. Also, this section details how the example can be modified to use the SSL communication protocol between the TIBCO Enterprise Message Service server and both Borland Enterprise Server and the J2EE application client.

Topics

- *Configure Borland Enterprise Server to use TIBCO Enterprise Message Service, page 56*
- *Configure TIBCO Enterprise Message Service for the Example Program, page 59*
- *Building and Deploying the Example MDB and the Example Client, page 64*
- *Running This Example, page 65*
- *Modifying This Example to use SSL Communications, page 66*

Configure Borland Enterprise Server to use TIBCO Enterprise Message Service

Borland Enterprise Server (BES) 5.1 uses a definitions archive (DAR) module for deployment of administered objects. Administered JMS objects such as queues, topics, and their respective connection factories are defined in the `jndi-definitions.xml` file. This file is used to build the DAR module that defines objects to be loaded into the Borland Partition's Naming Service. You must build a DAR module that specifies TIBCO Enterprise Message Service objects.

The Borland Enterprise Server installation contains several copies of the `jndi-definitions.xml` file. Modify the file located in `C:\<BES-install-dir>\examples\ejb\mdb` and use the modified file to build the DAR module.

In this file, there are several XML elements named `<jndi-object>` that define the SonicMQ JMS classes. These classes must be replaced with TIBCO Enterprise Message Service classes.

The following illustrates the replacements to make in bold:

```
<jndi-definitions>
  <jndi-object>
    <jndi-name>serial://jms/tibqcf</jndi-name>
    <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandQueueConnectionFactory
    </class-name>
    <property>
      <prop-name>serverUrl</prop-name>
      <prop-type>String</prop-type>
      <prop-value>localhost:7222</prop-value>
    </property>
  </jndi-object>
  <jndi-object>
    <jndi-name>serial://jms/tibtcf</jndi-name>
    <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandTopicConnectionFactory
    </class-name>
    <property>
      <prop-name>serverUrl</prop-name>
      <prop-type>String</prop-type>
      <prop-value>localhost:7222</prop-value>
    </property>
  </jndi-object>
  <jndi-object>
    <jndi-name>serial://jms/tibxaqcf</jndi-name>
    <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandXAQueueConnectionFactory
    </class-name>
```



```

    <property>
      <prop-name>serverUrl</prop-name>
      <prop-type>String</prop-type>
      <prop-value>localhost:7222</prop-value>
    </property>
  </jndi-object>
</jndi-object>
  <jndi-name>serial://jms/tibxatcf</jndi-name>
  <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandXAConnectionFactory
  </class-name>
  <property>
    <prop-name>serverUrl</prop-name>
    <prop-type>String</prop-type>
    <prop-value>localhost:7222</prop-value>
  </property>
</jndi-object>
</jndi-object>
  <jndi-name>serial://jms/tibq</jndi-name>
  <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandQueue</class-name>
  <property>
    <prop-name>address</prop-name>
    <prop-type>String</prop-type>
    <prop-value>TibQ1</prop-value>
  </property>
</jndi-object>
</jndi-object>
  <jndi-name>serial://jms/tibt</jndi-name>
  <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandTopic</class-name>
  <property>
    <prop-name>address</prop-name>
    <prop-type>String</prop-type>
    <prop-value>TibT1</prop-value>
  </property>
</jndi-object>
</jndi-definitions>

```

Save the changes to the `jndi-definitions.xml` file and build a new DAR module using the following command:

```
> jar cvMf ems-resources.dar META-INF/jndi-definitions.xml
```



If you are running the TIBCO Enterprise Message Service server in secure mode, you can specify default username and password attributes in the connection factories.

The default username and password are used by the connection factories for every connection created where a username and password is not explicitly provided by the application server. An example definition of these connection factory properties is shown below:

```
<property>
  <prop-name>userName</prop-name>
  <prop-type>String</prop-type>
  <prop-value>user1</prop-value>
</property>
<property>
  <prop-name>userPassword</prop-name>
  <prop-type>String</prop-type>
  <prop-value>secret</prop-value>
</property>
```

Deploy the TIBCO Enterprise Message Service JAR files, `tibjms.jar` and `tibjmsapps.jar`, then deploy the `ems-resources.dar` file in the target partition using the Borland Enterprise Server Console. The deployment steps are similar to an EJB JAR file. Refer to the *Borland Enterprise Server 5.1 User's Guide* for details.

Configure TIBCO Enterprise Message Service for the Example Program

You must create the topics and queues for the example program using the TIBCO Enterprise Message Service administration tool. To accomplish this, perform the following procedure:

1. Start the TIBCO Enterprise Message Service server (`tibemsd`).
2. Start the administration tool (`tibemsadmin`).
3. Enter the following commands at the administration tool prompt:

```
> connect
> create queue TibQ1
> create topic TibT1
> commit
```

Configure Borland Enterprise Server for the Example Message Driven Bean

Borland Enterprise Server contains an example MDB in the `C:\<BES-install-dir>\examples\ejb\mdb` directory. The example consists of the MDB `HelloBean.java` and the client `MdbClient.java`. The same bean can be used to consume messages from both queues and topics.

The MDB is defined in the standard `ejb-jar.xml` deployment descriptor file. This file defines two EJBs, one named `HelloEJBQueue` and another named `HelloEJBTopic`. Both beans are implemented as the same class, `com.borland.examples.ejb.mdb>HelloBean`. This class can be used for this example without modification.

Using Container-Managed XA Transactions

If you want to use container managed XA transactions with the `HelloEJBQueue` MDB, make the following changes to the deployment descriptors, `ejb-jar.xml` and `ejb-borland.xml`.

In `ejb-jar.xml`, make the changes in bold:

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>HelloEJBQueue</ejb-name>
      <ejb-class>
        com.borland.examples.ejb.mdb>HelloBean</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
        <destination-type>javax.jms.Queue</destination-type>
      </message-driven-destination>
      <env-entry>
        <env-entry-name>
          messageAcknowledgement</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>&lt;HelloEJBQueue&gt; Got a message
          from queue TibQ1:</env-entry-value>
        </env-entry>
      </message-driven>
    <message-driven>
      <ejb-name>HelloEJBTopic</ejb-name>
      <ejb-class>
        com.borland.examples.ejb.mdb>HelloBean</ejb-class>
      <transaction-type>Bean</transaction-type>
      <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
      <message-driven-destination>
        <destination-type>javax.jms.Topic</destination-type>
```

```

        <subscription-durability>
            Durable</subscription-durability>
    </message-driven-destination>
    <env-entry>
        <env-entry-name>
            messageAcknowledgement</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>&lt;HelloEJBTopic&gt; Got a message
            from topic TibT1:</env-entry-value>
    </env-entry>
</message-driven>
</enterprise-beans>
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>HelloEJBQueue</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
</ejb-jar>

```

In **ejb-borland.xml**, make the changes in bold:

```

<ejb-jar>
    <enterprise-beans>
        <message-driven>
            <ejb-name>HelloEJBQueue</ejb-name>
            <message-driven-destination-name>
                serial://jms/tibq</message-driven-destination-name>
            <connection-factory-name>
                serial://jms/tibxaqcf</connection-factory-name>
            <pool>
                <max-size>20</max-size>
                <init-size>2</init-size>
            </pool>
        </message-driven>
        <message-driven>
            <ejb-name>HelloEJBTopic</ejb-name>
            <message-driven-destination-name>
                serial://jms/tibt</message-driven-destination-name>
            <connection-factory-name>
                serial://jms/tibtcf</connection-factory-name>
            <pool>
                <max-size>20</max-size>
                <init-size>2</init-size>
            </pool>
        </message-driven>
    </enterprise-beans>
</ejb-jar>

```

Using XA Transactions That Are Not Container-Managed

If you do not want to use container managed XA transactions with the HelloEJBQueue MDB, make the following changes to the deployment descriptors, `ejb-jar.xml` and `ejb-borland.xml`.

In `ejb-jar.xml`, make the changes in bold:

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>HelloEJBQueue</ejb-name>
      <ejb-class>
        com.borland.examples.ejb.mdb.HelloBean</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
        <destination-type>javax.jms.Queue</destination-type>
      </message-driven-destination>
      <env-entry>
        <env-entry-name>
          messageAcknowledgement</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>&lt;<b>HelloEJBQueue&gt;<b> Got a message
          from queue TibQ1:</env-entry-value>
        </env-entry>
      </message-driven>
    <message-driven>
      <ejb-name>HelloEJBTopic</ejb-name>
      <ejb-class>
        com.borland.examples.ejb.mdb.HelloBean</ejb-class>
      <transaction-type>Bean</transaction-type>
      <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
      <message-driven-destination>
        <destination-type>javax.jms.Topic</destination-type>
        <subscription-durability>
          Durable</subscription-durability>
        </message-driven-destination>
      <env-entry>
        <env-entry-name>
          messageAcknowledgement</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>&lt;<b>HelloEJBTopic&gt;<b> Got a message
          from topic TibT1:</env-entry-value>
        </env-entry>
      </message-driven>
    </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>HelloEJBQueue</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute><b>NotSupported</b></trans-attribute>
    </container-transaction>
  </assembly-descriptor>
```

```
</ejb-jar>
```

In `ejb-borland.xml`, make the changes in bold:

```
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>HelloEJBQueue</ejb-name>
      <message-driven-destination-name>
        serial://jms/tibq</message-driven-destination-name>
      <connection-factory-name>
        serial://jms/tibqcf</connection-factory-name>
      <pool>
        <max-size>20</max-size>
        <init-size>2</init-size>
      </pool>
    </message-driven>
    <message-driven>
      <ejb-name>HelloEJBTopic</ejb-name>
      <message-driven-destination-name>
        serial://jms/tibt</message-driven-destination-name>
      <connection-factory-name>
        serial://jms/tibtcf</connection-factory-name>
      <pool>
        <max-size>20</max-size>
        <init-size>2</init-size>
      </pool>
    </message-driven>
  </enterprise-beans>
</ejb-jar>
```

Building and Deploying the Example MDB and the Example Client

The `application-client.xml` file does not need to be changed for this example. However, the application client deployment descriptor `application-client-borland.xml` file must be changed. The following highlights the changes to make in bold:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application-client PUBLIC "-//Borland Software
Corporation//DTD J2EE Application Client //EN"
"http://www.borland.com/devsupport/appserver/dtds/application-clie
nt_1_3-borland.dtd">
<application-client>
  <resource-ref>
    <res-ref-name>jms/qcf</res-ref-name>
    <jndi-name>serial://jms/tibqcf</jndi-name>
  </resource-ref>
  <resource-ref>
    <res-ref-name>jms/tcf</res-ref-name>
    <jndi-name>serial://jms/tibtcf</jndi-name>
  </resource-ref>
  <resource-env-ref>
    <resource-env-ref-name>jms/q</resource-env-ref-name>
    <jndi-name>serial://jms/tibq</jndi-name>
  </resource-env-ref>
  <resource-env-ref>
    <resource-env-ref-name>jms/t</resource-env-ref-name>
    <jndi-name>serial://jms/tibt</jndi-name>
  </resource-env-ref>
</application-client>
```

After changing `application-client-borland.xml`, build the example MDB and example, by changing to the directory `C:\<BES-install-dir>\examples\ejb\mdb` and issuing the `make_all` command. This results in two files in that directory: `message_beans.jar` and `message_beans_client.jar`.

Deploy the `message_beans.jar` module to the target partition using the Borland Enterprise Server Console. See the *Borland Enterprise Server 5.1 User's Guide* for more information about deploying modules.

Running This Example

Before running the example, ensure that the CLASSPATH includes the following JAR files:

```
message_beans_client.jar  
tibjms.jar  
tibjmsapps.jar  
tibcrypt.jar
```

To run the example client, navigate to the directory

C:\<BES-install-dir>\examples\ejb\mdb and enter the following command:

```
>appclient message_beans_client.jar
```

The client prints the following messages in the window:

```
Sending a message to queue TibQ1.  
Publishing a message to topic TibT1.  
Done.
```

The output of the MDB appears in the event log for the partition where you deployed the MDB. You can view the event log output from the Borland Enterprise Console.

The following messages are displayed in the event log:

```
<HelloEJBQueue> Got a message from queue TibQ1:  
Hello MDB, this is a message from the client...  
<HelloEJBTopic> Got a message from topic TibT1:  
Hello MDB, this is another message from the client...
```

Modifying This Example to use SSL Communications

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, Borland Enterprise Server, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Configuring the TIBCO Enterprise Message Service Server for SSL

In `C:\Tibco\EMS\bin\tibemsd.conf`, add the following lines:

```
listen = ssl://localhost:7223

ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password = password

listen = tcp://localhost:7222
```

These lines explicitly set the tcp and ssl listen ports, and specify the three required server-side SSL parameters: identity, private key, and password.

Save the file, then stop and restart the TIBCO Enterprise Message Service server.

Configuring Borland Enterprise Server and the Application Client for SSL-Based Communication

You must configure the JMS ConnectionFactories that Borland Enterprise Server and the application client retrieve from JNDI to use SSL-based communication. Borland Enterprise Server reads definitions for JMS administered objects from the `jndi-definitions.xml` file, deployed as part of a DAR module. Borland Enterprise Server instantiates and stores the objects into its own JNDI provider for subsequent lookup by all J2EE clients. Therefore, modify the definitions of the ConnectionFactories in the `jndi-definitions.xml` as described in the following paragraphs. After the modifications are complete, build and deploy a new DAR module using the updated `jndi-definitions.xml` file.

Change the value of the `serverUrl` property for both the `QueueConnectionFactory` and the `TopicConnectionFactory` to specify "ssl" as the protocol and "7223" as the port. The following section of code illustrates this change.

```
<property>
  <prop-name>serverUrl</prop-name>
  <prop-type>String</prop-type>
  <prop-value>ssl://localhost:7223</prop-value>
</property>
```

Add definitions for two additional properties to both the `QueueConnectionFactory` and the `TopicConnectionFactory`. These properties turn on SSL tracing so that output is generated indicating that SSL is being used. The properties also turn off host verification so that specifying a trusted certificate is not required for this example (refer to the Borland Enterprise Server documentation for a complete list of all the parameters that can be set for the Connection Factories). The following section of code illustrates this change:

```
<property>
  <prop-name>SSLTrace</prop-name>
  <prop-type>Boolean</prop-type>
  <prop-value>true</prop-value>
</property>
<property>
  <prop-name>SSLEnableVerifyHost</prop-name>
  <prop-type>Boolean</prop-type>
  <prop-value>>false</prop-value>
</property>
```

Save the changes to the `jndi-definitions.xml` file and build a new DAR module using the following command:

```
> jar cvMf ems-resources.dar META-INF/jndi-definitions.xml
```

Deploy the JAR file `jcrt.jar`, `jnet.jar`, `jsse.jar`, and `tibcrypt.jar` from the TIBCO Enterprise Message Service installation to the target partition using the Borland Enterprise Server Console. Redeploy the `ems-resources.dar` file to the target partition (refer to the *Borland Enterprise Server 5.1 User's Guide* for details).

Stop and restart Borland Enterprise Server to make these changes take effect.

When Borland Enterprise Server starts, it uses the new SSL-based ConnectionFactories to establish SSL-based topic and queue connections to invoke the example MDB. This can be verified by examining the SSL tracing output in the error log of the target partition. The error log can be viewed using the Borland Enterprise Server Console.

When Borland Enterprise Server completes its startup sequence, you should see output similar to the following:

```
[Mon Oct 18 18:32:03 PST 2002] stderr: [TibjmsSSL]: using security vendor 'j2se'
[Mon Oct 18 18:32:03 PST 2002] stderr: [TibjmsSSL]: WARNING: server verification is
disabled, will trust any server.
[Mon Oct 18 18:32:03 PST 2002] stderr: [TibjmsSSL]: client identity not set, using
empty identity.
[Mon Oct 18 18:32:07 PST 2002] stderr: [TibjmsSSL]: selected cipher:
SSL_RSA_WITH_RC4_128_SHA
[Mon Feb 18 18:32:08 PST 2002] stderr: [TibjmsSSL]: WARNING: server verification is
disabled, will trust any server.
[Mon Oct 18 18:32:08 PST 2002] stderr: [TibjmsSSL]: client identity not set, using
empty identity.
[Mon Oct 18 18:32:08 PST 2002] stderr: [TibjmsSSL]: selected cipher:
SSL_RSA_WITH_RC4_128_SHA
```

Running This Example

To run the example client, navigate to the directory
C:\<BES-install-dir>\examples\ejb\mdb and enter the following command:

```
> appclient message_beans_client.jar
```

The client prints the same messages in the window as before, but the SSL trace messages described in the previous section are also output. For example:

```
[TibjmsSSL]: using security vendor 'j2se'
[TibjmsSSL]: WARNING: server verification is disabled, will trust any server.
[TibjmsSSL]: client identity not set, using empty identity.
[TibjmsSSL]: selected cipher: SSL_RSA_WITH_RC4_128_SHA
Sending a message to queue TibQ1.
[TibjmsSSL]: WARNING: server verification is disabled, will trust any server.
[TibjmsSSL]: client identity not set, using empty identity.
[TibjmsSSL]: selected cipher: SSL_RSA_WITH_RC4_128_SHA
Publishing a message to topic TibT1.
Done.
```

Integrating With Borland Enterprise Server 5.0

This chapter describes integrating TIBCO Enterprise Message Service with Borland Enterprise Server (BES) 5.0. Specifically, you can use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside Borland Enterprise Server with a J2EE application client.

Borland Enterprise Server has an example MDB named "Hello Message-Driven Beans Example". The example includes a simple MDB and J2EE application client program. The example illustrates how to trigger the MDB within Borland Enterprise Server using the external client program (using SonicMQ as the JMS provider).

This chapter demonstrates using that same example with TIBCO Enterprise Message Service as the JMS provider. Also, instructions are given on how to convert the example to use SSL as the communication protocol between the TIBCO Enterprise Message Service server and both Borland Enterprise Server and the J2EE application client.

Topics

- *Configure Borland Enterprise Server to use TIBCO Enterprise Message Service, page 70*
- *Configure TIBCO Enterprise Message Service for the Example Program, page 73*
- *Configure Borland Enterprise Server for the Example Message Driven Bean, page 74*
- *Building and Deploying the Example MDB and the Example Client, page 75*
- *Running This Example, page 76*
- *Modifying This Example to use SSL Communications, page 77*

Configure Borland Enterprise Server to use TIBCO Enterprise Message Service

Administered JMS objects such as queues, topics, and their respective connection factories are defined in the `jndi-definitions.xml` file. This file defines objects that are loaded into the Borland file-based naming service. This file is pre-configured to load the SonicMQ objects. You must modify this file to specify TIBCO Enterprise Message Service objects instead.

The Borland Enterprise Server installation contains several copies of the `jndi-definitions.xml` file, and you should modify the file in the correct location. If Borland Enterprise Server was installed in the default location (`C:\BorlandEnterpriseServer`), then there is a subdirectory within the `C:\BorlandEnterpriseServer\var\servers` directory named for the server you have installed. The default name of the subdirectory is the name of the machine, but you can specify a different server name. The `jndi-definitions.xml` file you wish to edit should be located in the following directory:

`C:\BorlandEnterpriseServer\var\servers\<server-name>\partitions\stand`
`ard`

Near the end of this file are several XML elements named `<jndi-object>` that define the SonicMQ JMS classes. These must be replaced with TIBCO Enterprise Message Service classes. The following illustrates the replacements to make in bold:

```
<jndi-object>
  <jndi-name>serial://jms/qcf</jndi-name>
  <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandQueueConnectionFactory
  </class-name>
  <property>
    <prop-name>serverUrl</prop-name>
    <prop-type>String</prop-type>
    <prop-value>tcp://localhost:7222</prop-value>
  </property>
</jndi-object>

<jndi-object>
  <jndi-name>serial://jms/tcf</jndi-name>
  <class-name>
com.tibco.tibjms.appserver.borland.TibjmsBorlandTopicConnectionFactory
  </class-name>
  <property>
    <prop-name>serverUrl</prop-name>
    <prop-type>String</prop-type>
    <prop-value>tcp://localhost:7222</prop-value>
  </property>
```

```

</jndi-object>

<jndi-object>
  <jndi-name>serial://jms/q</jndi-name>
  <class-name>
    com.tibco.tibjms.appserver.borland.TibjmsBorlandQueue
  </class-name>
  <property>
    <prop-name>address</prop-name>
    <prop-type>String</prop-type>
    <prop-value>SampleQ1</prop-value>
  </property>
</jndi-object>

<jndi-object>
  <jndi-name>serial://jms/t</jndi-name>
  <class-name>
    com.tibco.tibjms.appserver.borland.TibjmsBorlandTopic
  </class-name>
  <property>
    <prop-name>address</prop-name>
    <prop-type>String</prop-type>
    <prop-value>SampleT1</prop-value>
  </property>
</jndi-object>

```



If you are running the TIBCO Enterprise Message Service server in secure mode, you can specify default username and password attributes in the connection factories.

The default username and password are used by the connection factories for every connection created where a username and password is not explicitly provided by the application server. An example definition of these connection factory properties is shown below:

```

<property>
  <prop-name>userName</prop-name>
  <prop-type>String</prop-type>
  <prop-value>user1</prop-value>
</property>
<property>
  <prop-name>userPassword</prop-name>
  <prop-type>String</prop-type>
  <prop-value>secret</prop-value>
</property>

```

You must also place the TIBCO Enterprise Message Service JAR files where Borland Enterprise Server can locate them. There are various ways to accomplish this, but the simplest method is to create a directory named patches under C:\BorlandEnterpriseServer\lib and place the JAR files there. Borland

Enterprise Server prepends any jar files in that location to its CLASSPATH automatically. The following jar files from the TIBCO Enterprise Message Service installation should be placed under

C:\BorlandEnterpriseServer\lib\patches:

```
jms.jar  
jndi.jar  
tibjms.jar  
tibjmsapps.jar
```


Configure TIBCO Enterprise Message Service for the Example Program

You must create the topics and queues for the example program using the TIBCO Enterprise Message Service administration tool. To accomplish this, perform the following procedure:

1. Start the TIBCO Enterprise Message Service server (`tibemsd`).
2. Start the administration tool (`tibemsadmin`).
3. Enter the following commands at the administration tool prompt:

```
> connect
> create queue SampleQ1
> create topic SampleT1
> commit
```

Configure Borland Enterprise Server for the Example Message Driven Bean

Borland Enterprise Server contains an example MDB in the `C:\BorlandEnterpriseServer\examples\ejb\mdb` directory. The example consists of the MDB `HelloBean.java` and the client `MdbClient.java`. The same bean can be used to consume messages from both queues and topics.

The MDB is defined in the standard `ejb-jar.xml` deployment descriptor file. This file defines two EJBs, one named `HelloEJBQueue` and another named `HelloEJBTopic`. Both beans are implemented as the same class, `com.borland.examples.ejb.mdb>HelloBean`. This class can be used for this example without modification.

This example will not use XA transactions, therefore the deployment descriptor for the example MDB must be modified. The following two files must be modified to remove XA from the example:

```
C:\BorlandEnterpriseServer\examples\ejb\mdb\META-INF\ejb-jar.xml
C:\BorlandEnterpriseServer\examples\ejb\mdb\META-INF\ejb-borland.xml
```

In `ejb-jar.xml`, change the `<trans-attribute>` element of the deployment descriptor for the `HelloEJBQueue` bean. The attribute should be changed from `"Required"` to `"NotSupported"`.

In `ejb-borland.xml`, the JMS Destination and Connection Factory JNDI names are pre-configured to use the XA Queue Connection Factory. Change the `<connection-factory-name>` element from `serial://jms/xaqcf` to `serial://jms/qcf`.

Building and Deploying the Example MDB and the Example Client

The application client deployment descriptors (`application-client.xml` and `application-client-borland.xml`) do not need modification for this example. To build the example MDB and example, change to the directory `C:\BorlandEnterpriseServer\examples\ejb\mdb` and issue the `make_all` command. This results in two files in that directory: `message_beans.jar` and `message_beans_client.jar`.

To deploy the example MDB, use the Borland Enterprise Server console. Use the following procedure:

1. Start Borland Enterprise Server by selecting **Start > Programs > Borland Enterprise Server > Server** from the Windows Start menu.
2. Start the console by selecting **Start > Programs > Borland Enterprise Server > Console** from the Windows Start menu.
3. Enter the administrator password when the login box appears.
4. When the console window appears, highlight the server name of the server you just installed. You can find the server under **Management Domain > Enterprise Servers**.
5. On the main menu, select **Tasks > Deployment > Deploy J2EE modules to a partition**.
6. When the J2EE Deployment Wizard appears, click **"Add..."**.
7. Navigate to
`C:\BorlandEnterpriseServer\examples\ejb\mdb\message_beans.jar`
and click **"OK"**.
8. Ensure that the target partition is `bes://<server name>/standard`, then click **"OK"**. There should be no errors displayed in the dialog box.
9. To verify the MDB deployed successfully, navigate to **Management Domain > Enterprise Servers > <server-name> > Partition > standard > Deployed Modules** in the left hand pane of the console. You should see `message_beans.jar` in the list with a green check box next to it.

Running This Example

To run the example client, navigate to the directory
C:\BorlandEnterpriseServer\examples\ejb\mdb and enter the following
command:

```
>appclient message_beans_client.jar
```

The client prints the following messages in the window:

```
Sending a message to queue SampleQ1.  
Publishing a message to topic SampleT1.  
Done.
```

The output of the MDB appears in the event log in the following location:
C:\BorlandEnterpriseServer\var\servers\<server-name>\adm\logs\
partitions\standard\event.log

The following messages are contained in that file:

```
<HelloEJBQueue> Got a message from queue SampleQ1:  
Hello MDB, this is a message from the client...  
<HelloEJBTopic> Got a message from topic SampleT1:  
Hello MDB, this is another message from the client...
```

You can also view the event log output from the Borland Console, by clicking on "standard" in the left pane, clicking on the "Logs" tab in the lower right pane, and choosing "event" in the drop down menu in the upper left corner of the right pane.

Modifying This Example to use SSL Communications

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, Borland Enterprise Server, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Configuring the TIBCO Enterprise Message Service Server for SSL

In `C:\Tibco\EMS\bin\tibemsd.conf`, add the following lines:

```
listen = ssl://localhost:7223

ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password      = password

listen = tcp://localhost:7222
```

These lines explicitly set the tcp and ssl listen ports, and specify the three required server-side SSL parameters: identity, private key, and password.

Save the file, then stop and restart the TIBCO Enterprise Message Service server.

Configuring Borland Enterprise Server and the Application Client for SSL-Based Communication

You must configure the JMS ConnectionFactories that Borland Enterprise Server and the application client retrieve from JNDI to use SSL-based communication. Borland Enterprise Server reads definitions for JMS administered objects from the `jndi-definitions.xml` file then instantiates and stores the objects into its own JNDI provider for subsequent lookup by all J2EE clients. Therefore, modify the definitions of the ConnectionFactories in the `jndi-definitions.xml` as described in the following paragraphs.

Change the value of the `serverUrl` property for both the `QueueConnectionFactory` and the `TopicConnectionFactory` to specify "ssl" as the protocol and "7223" as the port. The following section of code illustrates this change.

```
<property>
  <prop-name>serverUrl</prop-name>
  <prop-type>String</prop-type>
  <prop-value>ssl://localhost:7223</prop-value>
</property>
```

Add definitions for two additional properties to both the `QueueConnectionFactory` and the `TopicConnectionFactory`. These properties turn on SSL tracing so that output is generated indicating that SSL is being used. The properties also turn off host verification so that specifying a trusted certificate is not required for this example (refer to the Borland Enterprise Server documentation for a complete list of all the parameters that can be set for the Connection Factories). The following section of code illustrates this change:

```
<property>
  <prop-name>SSLTrace</prop-name>
  <prop-type>Boolean</prop-type>
  <prop-value>true</prop-value>
</property>
<property>
  <prop-name>SSLEnableVerifyHost</prop-name>
  <prop-type>Boolean</prop-type>
  <prop-value>>false</prop-value>
</property>
```

Save the changes to the `jndi-definitions.xml` file.

You must also add the required SSL JAR files to the Borland Enterprise Server patches directory. The following additional JAR files from the TIBCO Enterprise Message Service installation must be placed in the same patches directory described in the section *Configure Borland Enterprise Server to use TIBCO Enterprise Message Service* on page 70:

```
jcert.jar
jnet.jar
jsse.jar
tibcrypt.jar
```

Stop and restart Borland Enterprise Server to make these changes take effect.

When Borland Enterprise Server starts, it uses the new SSL-based ConnectionFactories to establish SSL-based topic and queue connections to invoke the example MDB. This can be verified by examining the SSL tracing output. This output is written to Standard Error, and can be found in the file:

```
C:\BorlandEnterpriseServer\var\servers\<server-name>\adm\logs\
partitions\standard\error.log
```

When Borland Enterprise Server completes its startup sequence, you should see output similar to the following:

```
[Mon Feb 18 18:32:03 PST 2002] stderr: [TibjmsSSL]: using security vendor 'j2se'
[Mon Feb 18 18:32:03 PST 2002] stderr: [TibjmsSSL]: WARNING: server verification is
disabled, will trust any server.
[Mon Feb 18 18:32:03 PST 2002] stderr: [TibjmsSSL]: client identity not set, using
empty identity.
[Mon Feb 18 18:32:07 PST 2002] stderr: [TibjmsSSL]: selected cipher:
SSL_RSA_WITH_RC4_128_SHA
```

```
[Mon Feb 18 18:32:08 PST 2002] stderr: [TibjmsSSL]: WARNING: server verification is
disabled, will trust any server.
[Mon Feb 18 18:32:08 PST 2002] stderr: [TibjmsSSL]: client identity not set, using
empty identity.
[Mon Feb 18 18:32:08 PST 2002] stderr: [TibjmsSSL]: selected cipher:
SSL_RSA_WITH_RC4_128_SHA
```

Running This Example

To run the example client, navigate to the directory

C:\BorlandEnterpriseServer\examples\ejb\mdb and enter the following command:

```
> appclient message_beans_client.jar
```

The client prints the same messages in the window as before, but the SSL trace messages described in the previous section are also output. For example:

```
[TibjmsSSL]: using security vendor 'j2se'
[TibjmsSSL]: WARNING: server verification is disabled, will trust any server.
[TibjmsSSL]: client identity not set, using empty identity.
[TibjmsSSL]: selected cipher: SSL_RSA_WITH_RC4_128_SHA
Sending a message to queue SampleQ1.
[TibjmsSSL]: WARNING: server verification is disabled, will trust any server.
[TibjmsSSL]: client identity not set, using empty identity.
[TibjmsSSL]: selected cipher: SSL_RSA_WITH_RC4_128_SHA
Publishing a message to topic SampleT1.
Done.
```


Chapter 8

Integrating With WebLogic Server 8.1

This chapter describes how to use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside WebLogic Server 8.1. The examples in this chapter use the preconfigured Examples server and the example MDB that comes bundled as a sample with WebLogic Server 8.1.

The examples in this chapter assume you have installed WebLogic Server 8.1 on a Windows platform in the directory C:\bea.

Topics

- *Running the Example MDB with WebLogic Server, page 82*
- *Configuring the Example MDB, page 83*
- *Rebuilding and Redeploying the Example MDB, page 87*
- *Running the Example MDB Client, page 88*
- *Modifying this Example to Use SSL Communication, page 89*
- *Modifying this Example to use Container Managed Transactions and XA, page 92*

Running the Example MDB with WebLogic Server

You should run the example MDB using WebLogic Server to ensure the MDB is configured and deployed properly. The source code for the example MDB is in:

```
C:\bea\WebLogic81\samples\server\examples\src\examples\ejb20\message
```

The file `package-summary.html` in the directory above contains instructions for building and running the example MDB inside the Examples server. Follow the instructions detailed in that file for running the example MDB. One thing the instructions fail to explicitly state is that you must navigate to the MDB source code directory, given above, when you execute the command to run the MDB client (that is, the command `ant run`).

Configuring the Example MDB

You must make the following configuration changes to the WebLogic Server 8.1 to drive the example MDB using TIBCO Enterprise Message Service instead of WebLogic Server.

- Add the TIBCO Enterprise Message Service JAR file to the CLASSPATH of WebLogic Server.
- Create the appropriate foreign JMSConnectionFactory and foreign JMSDestination for the foreign JMS Server, TIBCO Enterprise Message Service, in WebLogic. This is necessary to allow the WebLogic server to redirect lookups of ConnectionFactory and Destinations in its JNDI to TIBCO Enterprise Message Service's JNDI.
- Create the appropriate JMS Destination object inside the TIBCO Enterprise Message Service server using its administration tool.
- Modify the `weblogic-ejb-jar.xml` file for the MDB to use appropriate JMSConnectionFactory and JMSDestination.
- Modify the client program to look up its administered objects from the built-in JNDI provider in TIBCO Enterprise Message Service.

These steps are described in the following sections.

Adding TIBCO Enterprise Message Service to the WebLogic CLASSPATH

In the directory `C:\bea\weblogic81\samples\domains\examples`, modify the CLASSPATH environment variable in `setExamplesEnv.cmd` (the examples setup script.) and `startExamplesServer.cmd` (the start script).



On Windows platforms the extension for both of these files is `.cmd`; on UNIX platforms the extension is `.sh`.

Modify the CLASSPATH by adding this path to the end of its value list:

```
C:\Tibco\ems\clients\java\tibjms.jar
```

Creating Foreign JMS Server, JMSConnectionFactory, and JMSDestination in WebLogic

To create a foreign JMS Server, JMSConnectionFactory, and JMSDestination:

1. Open a new command prompt window and change directory to:
`C:\bea\weblogic81\samples\domains\examples`.

2. Run the script `startExamplesServer.cmd`.
3. When the WebLogic server completes startup, it will automatically point your default browser to the examples page. If it does not, start a web browser and load the page, `http://<machineName>:7001/examplesWebApp/index.jsp`.
Click on the Administration Console link. Enter `weblogic` as the Username and Password and click Sign In.
4. In the left pane, select **examples->services->JMS->Foreign JMS Servers**.
5. In the right pane click the "Configure a new Foreign JMSServer..." link.
6. Enter `TIBCO JMSServer` in the Name box, `com.tibco.tibjms.naming.TibjmsInitialContextFactory` in the JNDI Initial Context Factory box, and `tibjmsnaming://localhost:7222` in the JNDI Connection URL box.
Click **Create** and then click **Apply**.
7. In the left pane, select **examples->services->JMS->Foreign JMS Servers->TIBCO JMSServer->Foreign JMSConnectionFactory**.
8. In the right pane, click the "Configure a new Foreign JMSConnectionFactory..." link.
9. Enter `TIBCO JMSTopicConnectionFactory` in the Name box, `TIBCO.tcf` in the Local JNDI Name box, and `TopicConnectionFactory` in the Remote JNDI Name box.
Click **Create** and then click **Apply**.
10. In the left pane, select **examples->services->JMS->Foreign JMS Servers->TIBCO JMSServer->Foreign JMSDestinations**.
11. In the right pane, click the "Configure a new Foreign JMSDestination..." link.
12. Enter `TIBCO JMSTopic quotes` in the Name box, `TIBCO.quotes` in the Local JNDI Name box, and `quotes` in the Remote JNDI Name box.
Click **Create** and then click **Apply**.

Creating the Example MDB Destination Object Inside TIBCO EMS

To create the example MDB destination objects, perform the following:

1. Start the TIBCO Enterprise Message Service server by selecting **Start->Programs->TIBCO Enterprise Message Service 4.3->Start JMS Server** from the Windows Start menu.

2. Start the TIBCO Enterprise Message Service administration tool by selecting **Start->Programs->TIBCO Enterprise Message Service 4.3->Start EMS Administration Tool** from the Windows Start menu.
3. Enter the following commands:


```
> connect
> create topic quotes
```

Modifying the weblogic-ejb-jar.xml file for MDB

To use the appropriate JMSConnectionFactory and JMSDestination modify the file `C:\bea\weblogic81\samples\server\examples\src\examples\ejb20\message\weblogic-ejb-jar.xml` as follows:

1. Replace the two instances of `quotes` in the `<destination-jndi-name>` element with `TIBCO.quotes`.
2. Within the `<message-driven-descriptor>` element and immediately after both instances of the `<destination-jndi-name>` element, add the following element:

```
<connection-factory-jndi-name>
    TIBCO.tcf
</connection-factory-jndi-name>
```

Modifying the Client Program to Use TIBCO Enterprise Message Service JNDI

To use the JNDI provided by TIBCO Enterprise Message Service, the example MDB client program must be modified in three areas:

- the source code
- the build script
- the runtime environment i.e. the CLASSPATH

To modify the client source code:

The source file for the MDB client program is `Client.java` in the directory:

`C:\bea\weblogic81\samples\server\examples\src\examples\ejb20\message`

Find and replace the following strings in the source file:

Find	Replace With...
<code>weblogic.jms.ConnectionFactory</code>	<code>TopicConnectionFactory</code>
<code>weblogic.jndi.WLInitialContextFactory</code>	<code>com.tibco.tibjms.naming.TibjmsInitialContextFactory</code>

There should be one occurrence of each of the above strings. When you are finished, save your changes.

To modify the build script to run the client:

The client program is run by executing the ant build script with a target of run. The build script passes the JNDI provider URL to the client program, and therefore it must be modified to pass the URL of TIBCO Enterprise Message Service JNDI. The file `build.xml` in the example MDB source directory contains the build script. Near the bottom of that file is the following line:

```
<arg value="t3://localhost:${PORT}"/>
```

Modify that line as follows:

```
<arg value="tibjmsnaming://localhost:7222" />
```

To set the environment:

You have already added the `tibjms.jar` file to the CLASSPATH in a previous section. To set the environment, perform the following:

1. Open a new command prompt window.
2. Change directory to:

```
C:\bea\weblogic81\samples\domains\examples>
```

3. Enter the following command:

```
> setExamplesEnv
```

Verify that `tibjms.jar` is present when the script echoes the CLASSPATH.

Rebuilding and Redeploying the Example MDB

If WebLogic Server 8.1 server is still running, restart it. This causes TIBCO Enterprise Message Service to be added to its environment. Using the window created in the section To set the environment: on page 86, change directory to the example MDB source directory:

```
C:\bea\weblogic81\samples\server\examples\src\examples\ejb20\message
```

Enter the following command to rebuild and redeploy the MDB:

```
> ant
```

As the build completes, you should see messages in the WebLogic Server Examples Server window indicating that it is activating the "message" application.

Running the Example MDB Client

In the window used to rebuild and redeploy the Example MDB, enter:

```
> ant run
```

You should see the same results in the WebLogic Server Examples Server window as when you ran the example with WebLogic Server JMS.

To show that TIBCO Enterprise Message Service is driving the MDB, you could start another command prompt window and run the TIBCO Enterprise Message Service `tibjmsTopicSubscriber` sample as follows:

```
> java tibjmsTopicSubscriber -topic quotes
```

When you run the example MDB client, you should see that the `tibjmsTopicSubscriber` program receives the messages published by the example MDB client, along with the WebLogic Server.

Modifying this Example to Use SSL Communication

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, the WebLogic Server 8.1, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH

Modify the WebLogic Server 8.1 environment setup script, `setExamplesEnv.cmd` in the directory `C:\bea\weblogic81\samples\domains\examples`. This script is used to setup the environment to run the examples and the examples WebLogic server.

To add SSL JAR Files and New JNDI Properties File to the WLS 8.1 CLASSPATH:

1. Open the file named `setExamplesEnv.cmd`.
2. Add the following jar files to the end of the CLASSPATH and save the file.
`C:\Tibco\ems\clients\java\jcert.jar;C:\Tibco\ems\clients\java\jnet.jar;C:\Tibco\ems\clients\java\jsse.jar;C:\Tibco\ems\clients\java\tibcrypt.jar; C:\Tibco\ems\clients\java`
3. Create a new file named `jndi.properties`, add the following lines and save it to the directory `C:\Tibco\EMS\clients\java`.

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```

These properties specify that the "SSL" protocol should be used for JNDI lookups and that host verification is turned off (the client will trust any host). JNDI reads this file automatically and adds the properties to the environment of the initial JNDI context.

Configure the TIBCO Enterprise Message Service Server for SSL

In `C:\Tibco\EMS\bin\tibemsd.conf`, add the following lines:

```
listen = ssl://localhost:7243
ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password = password
listen = tcp://localhost:7222
```

These lines explicitly set the tcp and ssl listen ports and specify the three required server-side SSL parameters identity, private key, and password.

Save the file, then stop and restart the TIBCO Enterprise Message Service server. When the server restarts, you should see messages like the following in the console window confirming SSL is enabled:

```
2003-06-14 10:00:05 Secure Socket Layer is enabled, using openssl <version>
2003-06-14 10:00:05 Accepting connections on ssl://<machineName>:7243.
2003-06-14 10:00:05 Accepting connections on tcp://<machineName>:7222.
```

Modify the foreign JMSConnectionFactory in WebLogic to point to an SSLConnectionFactory

In the web browser load the page,
<http://<machineName>:7001/examplesWebApp/index.jsp>. Click on the Administration Console link. Enter weblogic as the Username and Password and click **Sign In**.

1. In the left pane, select **examples->services->JMS->Foreign JMS Servers->TIBCO JMSServer**.
2. In the right pane, replace the contents of the JNDI Connection URL box with `tibjmsnaming://localhost:7243`. Click **Apply**.
3. In the left pane, select **examples->services->JMS->Foreign JMS Servers->TIBCO JMSServer->Foreign JMSConnectionFactory->TIBCO JMSTopicConnectionFactory**.
4. In the right pane, replace the contents of the Local JNDI Name box with `TIBCO.tcf` and the contents of the Remote JNDI Name box with `SSLTopicConnectionFactory`. Click **Apply**.

Modify the Example Client Program for SSL-Based Communication

The modifications necessary for the example client program are similar to those that were necessary for MDB:

1. In `Client.java`, change the string `TopicConnectionFactory` to `SSLTopicConnectionFactory`.
2. In `build.xml`, change the port number 7222 to 7243 for the URL.

Rebuilding and Redeploying the Example MDB

Restart the WebLogic Server Examples Server so that it picks up the SSL related changes to the environment.

From the example MDB source directory, enter the command:

```
> ant
```

As the build completes, you should see messages in the WebLogic Server Examples Server window indicating that it is activating the "message" application.

Running the Example MDB Client with SSL

Create a new command prompt window and run the examples setup script, `setExamplesEnv.cmd`, so that the SSL related changes to the environment are picked up.

From the example MDB source directory, enter the command:

```
> ant run
```

You should see the same messages sent by the client and received by the MDB in the WebLogic server window. You may notice that this example runs slightly slower than the non-SSL version. This is because of the SSL handshake that occurs before the messages are displayed.

To show that SSL communications are in fact occurring, you could remove the SSL settings you added to `tibemsd.conf`. Then restart the TIBCO Enterprise Message Service server and the WebLogic Server. If you check the WebLogic Server logs, you should see exceptions thrown indicating that it could not connect. If you now run the test program again, you should see that it throws an exception indicating that it could not connect to the server using the SSL protocol. Alternatively (or additionally), you could start the TIBCO Enterprise Message Service server from a command prompt window and turn SSL debug tracing on, as follows:

```
>tibemsd -ssl_debug_trace
```

Then, if you re-start WebLogic Server and re-run the test program, you will see SSL debugging output on the `tibemsd` console window.

Modifying this Example to use Container Managed Transactions and XA

This section describes how to modify the above example to support container-managed transactions. In this modified example, TIBCO Enterprise Message Service server participates in a distributed transaction started by WebLogic server.

Modify the foreign JMSConnectionFactory in WebLogic to point to a XAConnectionFactory.

In the web browser, load the page,
<http://<machineName>:7001/examplesWebApp/index.jsp>. Click on the Administration Console link. Enter `weblogic` as the Username and Password and click **Sign In**.

1. In the left pane, select **examples->services->JMS->Foreign JMS Servers->TIBCO JMSServer->Foreign JMSConnectionFactory->TIBCO JMSTopicConnectionFactory**.
2. In the right pane, replace the contents of the Local JNDI Name box with `TIBCO.xatcf` and the contents of the Remote JNDI Name box with `XATopicConnectionFactory`. Click **Apply**.

Create a JMS Connection factory that supports XA

To create the JMS Connection factory that supports XA, perform the following:

1. Start the TIBCO Enterprise Message Service administration tool by selecting **Start->Programs->TIBCO Enterprise Message Service 4.3->Start EMS Administration Tool** from the Windows Start menu.
2. Enter the following commands:


```
> connect
> create factory XATopicConnectionFactory xatopic
```

Modifying the WebLogic Deployment files to make MDB to use transactions

1. Add the following lines to `ejb-jar.xml`, within the `<ejb-jar>` descriptor, after the `<enterprise-beans>` descriptor:


```
<assembly-descriptor>
```

```

<container-transaction>
  <method>
    <ejb-name>exampleMessageDriven1</ejb-name>
    <method-name>*</method-name>
  </method>
  <method>
    <ejb-name>exampleMessageDriven2</ejb-name>
    <method-name>*</method-name>
  </method>
  <trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>

```

2. Change the element `<connection-factory-jndi-name>` in the file `weblogic-ejb-jar.xml`. To use a message driven bean with container managed transactions, it should use a JMS connection factory that supports XA. Change the value of the element `<connection-factory-jndi-name>` to `TIBCO.xatcf`.
3. Rebuild, redeploy, and run the example MDB in the same manner as described in the previous sections.

Integrating With WebLogic Server 7.0

This chapter describes how to use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside WebLogic Server 7.0. The examples in this chapter use the preconfigured Examples server and the example MDB that comes bundled as a sample with WebLogic Server 7.0.

The examples in this chapter assume you have installed WebLogic Server 7.0 on a Windows platform.

Topics

- *Running the Example MDB with WebLogic Server, page 96*
- *Configuring the Example MDB, page 97*
- *Rebuilding and Redeploying the Example MDB, page 100*
- *Running the Example MDB Client, page 101*
- *Modifying this Example to Use SSL Communication, page 102*
- *Modifying this Example to use Container Managed Transactions and XA, page 105*

Running the Example MDB with WebLogic Server

You should run the example MDB using WebLogic Server to ensure the MDB is configured and deployed properly.

The source code for the example MDB is in:

```
C:\bea\weblogic700\samples\server\src\examples\ejb20\message
```

The file `package-summary.html` in that directory contains instructions for building and running the example MDB inside the Examples server. Follow the instructions detailed in that file for running the example MDB.



One thing the instructions fail to explicitly state is that you need to navigate to the MDB source code directory, given above, when you execute the command to run the MDB client (that is, the command `ant run`).

Configuring the Example MDB

You must make the following configuration changes to the WebLogic Server 7.0 to drive the example MDB using TIBCO Enterprise Message Service instead of WebLogic Server.

- Add the TIBCO Enterprise Message Service JAR file to the CLASSPATH of WebLogic Server.
- Modify the deployment descriptor of the example MDB to specify the TIBCO Enterprise Message Service initial context factory, provider URL, and connection factory JNDI name.
- Modify the client program to look up its administered objects from the built-in JNDI provider in TIBCO Enterprise Message Service.
- Create the appropriate JMS Destination object inside the TIBCO Enterprise Message Service server using the administration tool.

These steps are described below.

Adding TIBCO Enterprise Message Service to the WebLogic Server CLASSPATH

The WebLogic Server 7.0 environment setup and startup scripts check for the existence of the environment variable `EXT_PRE_CLASSPATH` and a file named `extEnv.cmd`. If the variable exists, WebLogic Server prepends its value to the CLASSPATH. If the `extEnv.cmd` file exists, it is called when WebLogic Server starts. To add TIBCO Enterprise Message Service to the WLS 7.0 CLASSPATH:

1. Create a file named `extEnv.cmd` that contains the following line:

```
set EXT_PRE_CLASSPATH=C:\Tibco\ems\clients\java\tibjms.jar
```

2. Save the file to the following location:

```
C:\bea\weblogic700\samples\server\config\examples
```

Modifying the MDB Deployment Descriptor for TIBCO Enterprise Message Service

The source files for the example MDB deployment descriptor are found in the example MDB source directory. Modify the file named `weblogic-ejb-jar.xml`. The following lines must be inserted (in two separate places) inside the `<message-driven-descriptor>` tag:

```
<initial-context-factory>
```

```
        com.tibco.tibjms.naming.TibjmsInitialContextFactory
    </initial-context-factory>
    <provider-url>
        tibjmsnaming://localhost:7222
    </provider-url>
    <connection-factory-jndi-name>
        TopicConnectionFactory
    </connection-factory-jndi-name>
```

Modifying the Client Program to Use TIBCO Enterprise Message Service JNDI

To use the JNDI provided by TIBCO Enterprise Message Service, the example MDB client program must be modified in three areas: the source code, the build script, and the runtime environment (that is, the CLASSPATH).

To modify the client source code:

The source file for the MDB client program is `Client.java`. Find and replace the following strings in the source file:

Find:	Replace With:
<code>weblogic.jms.ConnectionFactory</code>	<code>TopicConnectionFactory</code>
<code>weblogic.jndi.WLInitialContextFactory</code>	<code>com.tibco.tibjms.naming.TibjmsInitialContextFactory</code>

There should be one occurrence of each of the above strings. When you are finished, save your changes.

To modify the build script to run the client:

The client program is run by executing the ant build script with a target of `run`. The build script passes the JNDI provider URL to the client program, and therefore it must be modified to pass the provider URL of TIBCO Enterprise Message Service.

The file `build.xml` in the example MDB source directory contains the build script. Near the bottom of that file is the following line:

```
<arg value="t3://localhost:${PORT}"/>
```

Modify that line as follows:

```
<arg value="tibjmsnaming://localhost:7222" />
```

To set the environment:

You have already specified that `tibjms.jar` should be added to the `CLASSPATH` by creating a new command file in a previous section. To set the environment, perform the following:

1. Open a new command prompt window
2. Change directory to:

```
C:\bea\weblogic700\samples\server\config\examples>
```

3. Enter the following command:

```
> setExamplesEnv
```

Verify that `tibjms.jar` is present when the script echoes the `CLASSPATH`.

Creating the Example MDB Destination Object Inside TIBCO EMS

To create the example MDB destination objects, perform the following:

1. Start the TIBCO Enterprise Message Service server by selecting **Start->Programs->TIBCO Enterprise Message Service 4.3->Start JMS Server** from the Windows Start menu.
2. Start the TIBCO Enterprise Message Service administration tool by selecting **Start->Programs->TIBCO Enterprise Message Service 4.3->Start EMS Administration Tool** from the Windows Start menu.
3. Enter the following commands:

```
> connect
> create topic quotes
```

Rebuilding and Redeploying the Example MDB

If WebLogic Server 7.0 server is still running, restart it. This causes TIBCO Enterprise Message Service to be added to its environment.

Using the window created in To set the environment: on page 99, change directory to the example MDB source directory:

```
C:\bea\weblogic700\samples\server\src\examples\ejb20\message
```

Enter the following command to rebuild and redeploy the MDB:

```
> ant
```

As the build completes, you should see messages in the WebLogic Server Examples Server window indicating that it is activating the "message" application.

Running the Example MDB Client

From the window used to rebuild and redeploy the Example MDB, navigate to the example MDB source directory, then enter:

```
> ant run
```

You should see the same results in the WebLogic Server Examples Server window as when you ran the example with WebLogic Server JMS. To show that TIBCO Enterprise Message Service is driving the MDB, you could start another command prompt window and run the TIBCO Enterprise Message Service `tibjmsTopicSubscriber` sample as follows:

```
> java tibjmsTopicSubscriber -topic quotes
```

When you run the example MDB client, you should see that the `tibjmsTopicSubscriber` program receives the messages published by the example MDB client, along with the WebLogic Server.

Modifying this Example to Use SSL Communication

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, the WebLogic 7.0 Server, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH

The following JAR files distributed with TIBCO Enterprise Message Service must be added to the CLASSPATH:

```

jcert.jar
jnet.jar
jsse.jar
tibcrypt.jar

```

You can add them to the `extEnv.cmd` file that you created in Adding TIBCO Enterprise Message Service to the WebLogic Server CLASSPATH on page 97.

Next, create a new file named `jndi.properties` containing the following lines:

```

com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false

```

Save the file to directory `C:\Tibco\EMS\clients\java`. This directory must then also be added to the CLASSPATH in `extEnv.cmd`.

These properties specify that the "SSL" protocol should be used for JNDI lookups and that host verification is turned off (the client will trust any host). JNDI reads this file automatically and adds the properties to the environment of the initial JNDI context.

Configure the TIBCO Enterprise Message Service Server for SSL

In `C:\Tibco\EMS\bin\tibemspd.conf`, add the following lines:

```

listen                = ssl://localhost:7243
ssl_server_identity    = certs/server.cert.pem
ssl_server_key         = certs/server.key.pem
ssl_password          = password
listen                = tcp://localhost:7222

```

These lines explicitly set the `tcp` and `ssl` listen ports and specify the three required server-side SSL parameters: identity, private key, and password.

Save the file, then stop and restart the TIBCO Enterprise Message Service server. When the server restarts, you should see messages like the following in the console window confirming SSL is enabled:

```
2002-03-19 13:48:34 Secure Socket Layer is enabled.
2002-03-19 13:48:34 Accepting connections on ssl://localhost:7243.
2002-03-19 13:48:34 Accepting connections on tcp://localhost:7222.
```

Configure Example MDB for SSL-Based Communication

Modify the file `weblogic-ejb-jar.xml` to change the values of the JNDI provider URL and the connection factory JNDI name, as follows:

```
<provider-url>
    tibjmsnaming://localhost:7243
</provider-url>
<connection-factory-jndi-name>
    SSLTopicConnectionFactory
</connection-factory-jndi-name>
```

The provider URL is changed to connect to port 7243 (instead of 7222), and the connection factory JNDI name is changed to specify the SSL-based topic connection factory that comes preconfigured in TIBCO Enterprise Message Service.

Modify the Example Client Program for SSL-Based Communication

The modifications necessary for the example client program are similar to those that were necessary for MDB:

- In `Client.java`, change the string `"TopicConnectionFactory"` to `"SSLTopicConnectionFactory"`.
- In `build.xml`, change the port number `"7222"` to `"7243"` for the URL.

Rebuilding and Redeploying the Example MDB

Restart the WebLogic Server Examples Server so that it picks up the changes to the environment.

From the example MDB source directory, enter the command:

```
> ant
```

As the build completes, you should see messages in the WebLogic Server Examples Server window indicating that it is activating the "message" application.

Running the Example MDB Client with SSL

Create a new command prompt window and run the examples setup script so that the changes to the environment are picked up.

From the example MDB source directory, enter the command:

```
> ant run
```

You should see the same messages sent by the client and received by the MDB in the WebLogic server window. You may notice that this example runs slightly slower than the non-SSL version. This is because of the SSL handshake that occurs before the messages are displayed.

To show that SSL communications are in fact occurring, you could remove the SSL settings you added to `tibemsd.conf`. Then restart the TIBCO Enterprise Message Service server and the WebLogic Server. If you check the WebLogic Server logs, you should see exceptions thrown indicating that it could not connect. If you now run the test program again, you should see that it throws an exception indicating that it could not connect to the server using the SSL protocol.

Alternatively (or additionally), you could start the TIBCO Enterprise Message Service server from a command prompt window and turn SSL debug tracing on, as follows:

```
>tibemsd -ssl_debug_trace
```

Then, if you re-start WebLogic Server and re-run the test program, you will see SSL debugging output on the `tibemsd` console window.

Modifying this Example to use Container Managed Transactions and XA

This section describes how to modify the above example to support container-managed transactions. In this modified example, TIBCO Enterprise Message Service server participates in a distributed transaction started by WebLogic server.

Create a JMS Connection factory that supports XA.

To create the JMS Connection factory that supports XA, perform the following:

1. Start the TIBCO Enterprise Message Service administration tool by running command

```
tibemsadmin
```

2. Enter the following commands

```
connect
create factory XATopicConnectionFactory xatopic
```

Modifying the Weblogic Deployment files to make MDB to use transactions

1. Add the following lines to `ejb-jar.xml`:

```
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>exampleMessageDriven1</ejb-name>
      <method-name>*</method-name>
    </method>
    <method>
      <ejb-name>exampleMessageDriven2</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

2. Change `connection-factory-jndi-name` in the file `weblogic-ejb-jar.xml`. To use a message driven bean with container manager transactions, it should use a JMS connection factory that supports XA. Change `connection-factory-jndi-name` to `XATopicConnectionFactory`.

Rebuild, redeploy, and run the example MDB in the same manner as described in the previous sections.

Chapter 10 **Integrating With WebLogic Server 6.1**

This chapter describes integrating TIBCO Enterprise Message Service with WebLogic Server 6.1. You can share JMS messages in the same application between TIBCO Enterprise Message Service and WebLogic Server. This is useful if you wish to provide a gateway between the two JMS providers. You can also use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside WebLogic Server with a J2EE application client.

Topics

- *Using TIBCO Enterprise Message Service With WebLogic Server, page 108*
- *Using TIBCO Enterprise Message Service with WebLogic Server Message Driven Beans, page 114*
- *Modifying This Example to use SSL Communication, page 118*

Using TIBCO Enterprise Message Service With WebLogic Server

TIBCO Enterprise Message Service and WebLogic Server implement the same JMS interface, JMS 1.1. Because of this, ConnectionFactories from both TIBCO Enterprise Message Service and WebLogic Server can be used within the same program to create the connections, sessions, queues, producers, consumers for both JMS providers. Messages can then be sent and received between the two JMS providers.

The JMS 1.1 interfaces are defined in the J2EE specification and are provided as part of the J2EE platform in `jms.jar`. Table 1 lists where the implementations of this interface are stored and the unique implementation name for both JMS providers:

Table 1 TIBCO Enterprise Message Service and WebLogic Server JMS implementations

Provider	javax.jms.implementation in:	Implementation Name
TIBCO Enterprise Message Service	java/tibjms.jar	com.tibco.tibjms
WebLogic Server	lib/weblogic.jar	weblogic.jms.*

In this example, only generic calls are used for creation of administrative objects, and the TIBCO Enterprise Message Service server also acts as a JNDI provider.

For both JMS providers, the sample program performs the following:

1. The associated JNDI context is retrieved.
2. The JMS ConnectionFactory is looked up in that context.
3. A connection, session, queue, producer and sender are created.
4. A TIBCO Enterprise Message Service message is created.
5. The message is sent and received using TIBCO Enterprise Message Service.
6. The same message and sent and received using WebLogic Server.
7. The message is then sent and received again using TIBCO Enterprise Message Service.

The Sample Program

Create a file named `t.java` that has the following contents:

```
/*
```

```

* This program defines an object that works for WLS JMS and TIBCO
* JMS connection factories and queues. In the sample main, it
* generates a TIBCO Enterprise Message Service provider message,
* sends and receives it via TIBCO Enterprise Message Service,
* sends and receives it via WebLogic Server, then sends and
* receives it via TIBCO Enterprise Message Service.
*
* Usage: java t
*/

import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;

public class t {
    public static final String TIBCOqcf = "QueueConnectionFactory";
    public static final String WLSqcf =
        "javax.jms.QueueConnectionFactory";

    public static final String TIBCOqname = "myQueue";
    public static final String WLSqname = "jms.queue.TestQueue1";

    public static final String TIBCOurl =
        "tibjmsnaming://localhost:7222";
    public static final String WLSurl = "t3://localhost:7001";

    public static final String TIBCOJNDIfactory =
        "com.tibco.tibjms.naming.TibjmsInitialContextFactory";
    public static final String WLSJNDIfactory =
        "weblogic.jndi.WLInitialContextFactory";

    public static void main(String[] args) {
        JMSObject TIBCOobject = null;
        JMSObject WLSobject = null;
        TextMessage msg;
        Context ctx;

        try {
            // Create the TIBCO Enterprise Message Service connection
            // factory,
            // connection, session, queue
            TIBCOobject = new JMSObject(TIBCOurl, TIBCOJNDIfactory,
                TIBCOqcf, TIBCOqname);

            // Create the WLS connection factory, connection, session,
            // queue
            WLSobject = new JMSObject(WLSurl, WLSJNDIfactory, WLSqcf,
                WLSqname);

            msg = TIBCOobject.JMSMessage("Test String");

            TIBCOobject.JMSSend(msg);
            msg = TIBCOobject.JMSReceive();
            System.out.println("Received message: "+msg);

            WLSobject.JMSSend(msg);

```

```

        msg = WLSObject.JMSReceive();
        System.out.println("Received message: "+msg);

        TIBCOobject.JMSSend(msg);
        msg = TIBCOobject.JMSReceive();
        System.out.println("Received message: "+msg);

    } catch(JMSException je) {
        System.out.println("Caught JMSException: "+je);
        Exception le = je.getLinkedException();
        if (le != null) System.out.println("Linked
            exception: "+le);
        je.printStackTrace();
    } catch(Exception e) {
        e.printStackTrace();
        System.out.println("Caught Exception: "+e);
    } finally {
        try {
            if (TIBCOobject != null)
                TIBCOobject.JMSCleanup();
            if (WLSObject != null)
                WLSObject.JMSCleanup();
        } catch (Exception e) { }
    }
}

class JMSObject {
    private Queue ioQueue;
    private QueueSession session;
    private QueueConnection connection;
    private QueueConnectionFactory factory;
    private QueueSender queueSender;
    private QueueReceiver queueReceiver;
    private InitialContext ctx;

    JMSObject(String url, String jndi, String qcf, String qname)
        throws Exception {

        // Get the initial context
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY, jndi);
        if (url != null)
            env.put(Context.PROVIDER_URL, url);
        env.put(Context.REFERRAL, "throw");
        ctx = new InitialContext(env);

        factory = (QueueConnectionFactory)ctx.lookup(qcf);

        // Create a QueueConnection, QueueSession
        connection = factory.createQueueConnection();
        session = connection.createQueueSession(false,
            Session.AUTO_ACKNOWLEDGE);

        ioQueue = (Queue)ctx.lookup(qname);

        connection.start();
    }
}

```

```

        queueSender = session.createSender(ioQueue);
        queueReceiver = session.createReceiver(ioQueue);
    }

    TextMessage JMSMessage(String text) throws Exception {
        TextMessage msg = session.createTextMessage();
        msg.setText(text);
        return(msg);
    }

    void JMSSend(TextMessage msg) throws Exception {
        System.out.println("Sending the message on queue " +
            ioQueue.getQueueName());

        queueSender.send(msg);
    }

    TextMessage JMSReceive() throws Exception {
        TextMessage msg;
        System.out.println("Receiving the message on queue " +
            ioQueue.getQueueName());
        msg = (TextMessage)queueReceiver.receive(1000);

        if (msg == null)
            throw new JMSEException("Failed to receive message");
        return(msg);
    }

    void JMSCleanup() throws Exception {
        if (session != null) {
            session.close();
            session = null;
        }
        if (connection != null) {
            connection.close();
            connection = null;
        }
    }
}

```

Configure WebLogic Server for the Test Program

1. If you do not already have WebLogic Server, download and install WebLogic Server 6.1 with Service Pack 1. You can retrieve the latest version (and a 30-day trial license, if needed) from http://commerce.bea.com/downloads/weblogic_server.jsp#wls. Select the Microsoft Windows NT/2000 platform (one executable file to download and run). When installing WebLogic Server, choose the default WebLogic Administration Domain Name: mydomain, default server name: myserver, and default port: 7001.
2. Set up the WebLogic Server configuration to run JMS. In C:\bea\wlserver6.1\config\mydomain (modified based on where you

have installed WebLogic Server), modify `config.xml` to add the following JMS objects before the final `</domain>` line:

```
<JMSServer Name="TestJMSServer" Targets="myserver">
  <JMSQueue Name="TestQueue1" JNDIName="jms.queue.TestQueue1"/>
</JMSServer>
```

Configure TIBCO Enterprise Message Service within WebLogic Server

1. In `C:\bea\wlserver6.1\config\mydomain` (modified based on where you have installed WebLogic Server), modify `setEnv.cmd` for TIBCO Enterprise Message Service as follows:

```
@rem next line added for TIBCO Enterprise Message Service
set TIBJMS_ROOT=\tibco\JMS
@rem next line modified for TIBCO Enterprise Message Service
set CLASSPATH=%JAVA_HOME%\lib\tools.jar;
%WL_HOME%\lib\weblogic_sp.jar;%WL_HOME%\lib\weblogic.jar;
%TIBJMS_ROOT%\java\jms.jar;%TIBJMS_ROOT%\java\jndi.jar;
%TIBJMS_ROOT%\java\tibjms.jar; %CLASSPATH%
```

2. In this same directory, modify `startWebLogic.cmd` as follows:

```
@rem next line added for TIBCO Enterprise Message Service
set TIBJMS_ROOT=\tibco\JMS
@rem next line modified for TIBCO Enterprise Message Service
set CLASSPATH=.;.\lib\weblogic_sp.jar;.\lib\weblogic.jar;
%TIBJMS_ROOT%\java\jms.jar;%TIBJMS_ROOT%\java\jndi.jar;
%TIBJMS_ROOT%\java\tibjms.jar
```

Running WebLogic Server, TIBCO Enterprise Message Service, and the Test Program

1. To set the environment, change directory to `C:\bea\wlserver6.1\config\mydomain`. Then run `setEnv.cmd`.
2. Compile the program, `t.java` using: `javac t.java`.

The program is already configured with the URL, default queue connection factory, and JNDI factory of the WebLogic application server. The WebLogic queue name matches the name in the `config.xml` file. The program is also configured with the URL, queue connection factory, and JNDI factory of the TIBCO Enterprise Message Service server. The TIBCO queue name matches the name for the queue that will be created in the steps below.

3. Edit `C:\Tibco\EMS\bin\queues.conf` and add the following line at the end:

```
myQueue
```

This allows queue `myQueue` to be created.

4. Start TIBCO Enterprise Message Service by selecting **Start > Programs > TIBCO Enterprise Message Service 4.3 > Start JMS Server** from the Windows Start menu.
5. Start the TIBCO Enterprise Message Service administration tool by selecting **Start > Programs > TIBCO Enterprise Message Service 4.3 > Start EMS Administration Tool** from the Windows Start menu. Enter the following commands at the prompt:


```
> connect  
> create factory QueueConnectionFactory queue  
> create queue myQueue  
> commit
```
6. Start WebLogic Server by selecting **Start > Programs > BEA WebLogic E-Business Platform > Weblogic Server 6.1 > Start Default Server** from the Windows Start menu, and enter the administrator password.
7. Run the test program by typing `java t` at a command prompt. You should see messages indicating that the program sent and received the message by way of TIBCO Enterprise Message Service, then sent and received the same message via WebLogic Server, then sent and received the message using TIBCO Enterprise Message Service again.

Using TIBCO Enterprise Message Service with WebLogic Server

Message Driven Beans

This section describes how to use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) within WebLogic Server. It assumes you have already set up your environment as described in the previous section.

Configure WebLogic Server for the Sample MDB

This example assumes that the application directory for the MDB is C:\Tibco\EMS\samples\client\bea\MDB. You can change the configuration appropriately for a different directory.

1. Create an `ejb-jar.xml` file in the application directory as follows:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>MDB</ejb-name>
      <ejb-class>MDB</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
        <destination-type>javax.jms.Queue
        </destination-type>
      </message-driven-destination>
      <security-identity>
        <run-as>
          <role-name>everyone</role-name>
        </run-as>
      </security-identity>
    </message-driven>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>everyone</role-name>
    </security-role>
  </assembly-descriptor>
</ejb-jar>
```

This file defines the EJB with a class name of "MDB" (file name `MDB.class`). The EJB container can manage transactions, and it specifies that the MDB reads from a Queue. Security for the EJB and MDB is set so that everyone can run it.

Elements in this file are defined in the EJB 2.0 specification (see the EJB documentation at <http://java.sun.com> for more details).

2. Create a `weblogic-ejb-jar.xml` file in the same directory as follows:

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB//EN"
"http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd">
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>MDB</ejb-name>
    <message-driven-descriptor>
      <pool>
        <max-beans-in-free-pool>200
        </max-beans-in-free-pool>
        <initial-beans-in-free-pool>20
        </initial-beans-in-free-pool>
      </pool>
      <destination-jndi-name>myQueue</destination-jndi-name>
      <initial-context-factory>
        com.tibco.tibjms.naming.TibjmsInitialContextFactory
      </initial-context-factory>
      <provider-url>tibjmsnaming://localhost:7222
      </provider-url>
      <connection-factory-jndi-name>QueueConnectionFactory
      </connection-factory-jndi-name>
    </message-driven-descriptor>
    <jndi-name>MDB</jndi-name>
  </weblogic-enterprise-bean>
  <security-role-assignment>
    <role-name>everyone</role-name>
    <principal-name>system</principal-name>
  </security-role-assignment>
</weblogic-ejb-jar>
```

This file further defines the EJB named MDB to indicate how to get to the connection factory and the queue by specifying the JNDI names for the factory and the destination and also the URL for the provider. This example points to the same JNDI provider that was set up in the prior section, that is the TIBCO Enterprise Message Service server, and it identifies `myQueue` as the JNDI destination name for the MDB. This configuration also indicates that initially 20 beans will be deployed and up to 200 may be run if necessary to handle the message traffic.

The elements in this file are WebLogic Server-specific extensions to the EJB 2.0 specification. This file is more completely described at <http://edocs.bea.com/wls/docs60/ejb/reference.html#1071166>.

3. The source code for the message driven bean, `MDB.java`, is below. Save this source as a file named `MDB.java` to the `MDB` directory.

```
import javax.ejb.CreateException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.jms.Message;
import javax.jms.MessageListener;
```

```

import javax.jms.TextMessage;

public class MDB implements MessageDrivenBean, MessageListener {
    private MessageDrivenContext context;

    // Required - public constructor with no argument
    public MDB () {}

    // Required - ejbActivate
    public void ejbActivate() {}

    // Required - ejbRemove
    public void ejbRemove() {
        context = null;
    }

    // Required - ejbPassivate
    public void ejbPassivate() {}

    public void setMessageDrivenContext(
        MessageDrivenContext mycontext) {
        context = mycontext;
    }

    // Required - ejbCreate() with no arguments
    public void ejbCreate () throws CreateException {}

    // Implementation of MessageListener - throws no exceptions
    public void onMessage(Message msg) {
        try {
            System.out.println("MDB: " + ((TextMessage)msg).getText()
                + " Thread: " + Thread.currentThread().getName());
            Thread.sleep(2000, 0);
        }
        catch(Exception e) { // Catch any exception
            e.printStackTrace();
        }
    }
}

```

4. The following commands build the JAR file for the EJB. You should use the same console window where you invoked `C:\bea\wlserver6.1\config\mydomain\setEnv.cmd` in the last section.

```

mkdir build build\META-INF
copy ejb-jar.xml build\META-INF
copy weblogic-ejb-jar.xml build\META-INF
javac -d build MDB.java
cd build
jar cvf myejb.jar META-INF MDB.class
cd ..
java weblogic.ejbc -compiler javac build\myejb.jar MDB.jar

```

5. Add `MDB.jar` to the `CLASSPATH` in `startWebLogic.cmd`.

6. The configuration file, `../config/mydomain/config.xml`, must be updated to deploy the EJB by adding the following lines before the final `</domain>` line:

```
<Application Name="MDB"
Path="C:\TIBCO\EMS\samples\client\bea\MDB">
  <EJBComponent Name="MDB" URI="MDB.jar" Targets="myserver"/>
</Application>
```

7. The program, `t.java`, must be modified so that it only creates a foreign JMS provider message and sends it (take out the lines after `TIBCOobject.JMSSend(msg);` so you do not receive it — that is what the MDB is for).
8. Recompile the test program using `javac t.java`.

Run This Example

1. The TIBCO Enterprise Message Service server should still be running from the previous section, and therefore the administered objects `QueueConnectionFactory` and `myQueue` have already been created. In this example, the WebLogic Server, rather than the test program, reads from that queue.
2. Restart WebLogic Server so that the modifications for the MDB take effect.



When WebLogic Server starts, it echoes the `CLASSPATH` it is using. Make sure the `CLASSPATH` includes the TIBCO Enterprise Message Service JAR files as well as the `MDB.jar` file.

3. Run the test program.

```
> java t
```

You should see the messages that are sent by the client and received by the MDB printing in the WebLogic Server window.

Modifying This Example to use SSL Communication

This section describes how to modify the above example to use SSL communications between the TIBCO Enterprise Message Service server, WebLogic Server, and the client program (`t.java`). This section assumes you have already set up and run the example detailed in the previous sections.

Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH

1. The following JAR files, distributed with TIBCO Enterprise Message Service, must be added to the CLASSPATH settings in both `setEnv.cmd` and `startWebLogic.cmd`:

```

jcert.jar
jnet.jar
jsse.jar
tibcrypt.jar

```

2. Create a new file named `jndi.properties` containing the following lines:

```

com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false

```

3. Save the file to directory `C:\Tibco\EMS\clients\java`. This directory must then be added to the CLASSPATH in both `setEnv.cmd` and `startWebLogic.cmd`.

These properties specify that the "SSL" protocol should be used for JNDI lookups and that host verification is turned off (the client will trust any host). JNDI reads this file automatically and adds the properties to the environment of the initial JNDI context.

Configure the TIBCO Enterprise Message Service Server for SSL

1. In `C:\Tibco\EMS\bin\tibemsd.conf`, add the following lines:

```

listen = ssl://localhost:7223

ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password      = password

listen = tcp://localhost:7222

```

These lines explicitly set the tcp and ssl listen ports, and specify the three required server-side SSL parameters: identity, private key, and password.

2. Save the file, then stop and restart the TIBCO Enterprise Message Service server. When the server restarts, you should see messages like the following in the console window confirming SSL is enabled:

```
2002-03-19 13:48:34 Secure Socket Layer is enabled.
2002-03-19 13:48:34 Accepting connections on ssl://localhost:7223.
2002-03-19 13:48:34 Accepting connections on tcp://localhost:7222.
```

3. Start the administration tool, `tibemsadmin`. Then create a `QueueConnectionFactory` that establishes SSL connections. To do this, enter the following commands at the `tibemsadmin` prompt:

```
> connect
> create factory sslQCF queue url=ssl://localhost:7223
ssl_verify_host=disabled
```

In the above command, the SSL parameter "url" specifies that connections created by this factory will use the SSL protocol and connect on port 7223. The SSL parameter "ssl_verify_host" is set to disabled so that a server-side certificate property is not required (the client will trust any server). This is the simplest SSL configuration.

Configure WebLogic Server for SSL-Based Communication

1. The file `weblogic-ejb-jar.xml` file must be modified to change the values of the JNDI provider URL and the connection factory JNDI name, as follows:

```
<provider-url>
    tibjmsnaming://localhost:7223
</provider-url>
<connection-factory-jndi-name>sslQCF
</connection-factory-jndi-name>
```

The provider URL is changed to connect to port 7223 (instead of 7222), and the connection factory JNDI name is changed to specify the SSL-based queue connection factory that was created in the previous step.

2. Rebuild the JAR file for the EJB. From the build directory, enter the following commands:

```
jar cvf myejb.jar META-INF MDB.class
cd ..
java weblogic.ejbc -compiler javac build\myejb.jar MDB.jar
```

3. Stop and restart the WebLogic Server server.

Modify the Test Program for SSL-Based Communication

The modifications necessary for the test program are similar to those that were necessary for WebLogic Server:

- Change the value of the `TIBCOqcf` variable to `"sslQCF"`.
- Change the value of the port number in the `TIBCOurl` variable to `"7223"`.

Save and recompile the program.

Re-Run this Example

Run the test program:

```
>java t
```

You should see the same messages sent by the client and received by the MDB in the WebLogic server window. You may notice that this example runs slightly slower than the non-SSL version. This is because of the SSL handshake that occurs before the messages are displayed.

To prove that SSL communications are in fact occurring, you could remove the SSL settings you added to `tibemsd.conf` described in [Configure the TIBCO Enterprise Message Service Server for SSL](#) on page 118. Then restart the TIBCO Enterprise Message Service server and the WebLogic Server. You should find that the message normally printed indicating that the server is running in production mode never gets printed. The reason is that WebLogic Server cannot perform the SSL-based JNDI lookup of the connection factory, and it continues to retry forever. If you now run the test program again, you should see that it throws an exception indicating that it could not connect to the server using the SSL protocol.

Alternatively (or additionally), you could start the TIBCO Enterprise Message Service server from a command prompt window and turn SSL debug tracing on, as follows:

```
>tibemsd -ssl_debug_trace
```

Then if you re-start WebLogic Server and re-run the test program you will see SSL debugging output on the `tibemsd` console window.

Chapter 11 **Integrating With IBM WebSphere Application Server Version 5**

This chapter describes integrating TIBCO Enterprise Message Service with IBM WebSphere Application Server Version 5. Specifically, a J2EE client can use TIBCO Enterprise Message Service to trigger a Message Driven Bean (MDB) inside the WebSphere Application Server and also have the MDB send the received message back to the client.

Topics

- *Overview of Integrating With IBM WebSphere, page 122*
- *Get the sample MDB running with the WebSphere Embedded JMS Provider, page 123*
- *Get the Sample MDB running with TIBCO Enterprise Message Service, page 126*
- *Modify the Samples to Use SSL Communications, page 137*

Overview of Integrating With IBM WebSphere

The IBM WebSphere Application Server has two message-driven bean samples that separately demonstrate publish-and-subscribe (topic-based) and point-to-point (queue based) messaging. Each of these sample applications includes a simple MDB and J2EE application client program. The examples illustrate how to trigger the MDB within the WebSphere Application Server using the external client program by way of the WebSphere embedded JMS provider.

This chapter is divided into the following sections:

- Get the sample MDB running with the WebSphere Embedded JMS Provider — provides step-by-step instructions for running the sample MDBs using the WebSphere embedded JMS provider. This ensures that the MDBs are configured and deployed properly.
- Get the Sample MDB running with TIBCO Enterprise Message Service — demonstrates how to reconfigure and run the same MDBs using TIBCO Enterprise Message Service as the JMS provider within WebSphere.

Porting the sample MDBs from WebSphere embedded JMS to TIBCO Enterprise Message Service does not require changing any of the MDB or application client source code. TIBCO Enterprise Message Service is simply added to WebSphere application server and client container as another JMS provider. Then, the sample applications are reconfigured for the new JMS provider resources and re-deployed.

- Modify the Samples to Use SSL Communications — details how to modify the sample programs to use SSL as the communication protocol with TIBCO Enterprise Message Service.

The instructions in this section assume you have already downloaded and installed WebSphere Application Server Version 5.0 Trial (plus embedded messaging) on a Windows platform. The instructions also assume that TIBCO Enterprise Message Service and WebSphere Application Server are both running on the same machine.

Get the sample MDB running with the WebSphere Embedded JMS Provider



The WebSphere Embedded JMS Provider software is a separate (optional) download from the Application Server 5.0 Trial software. Be sure to download and install it along with the Application Server.

Launch the Samples Gallery from the Windows Start menu.

Get the Publish and Subscribe Sample Working

The publish and subscribe sample consists of an application client that publishes a message on one of three topics and an MDB that is listening on a fourth, wildcard topic, that receives messages published on any of the first three. The MDB prints the message it receives to the standard output.

1. Click on the "Message-driven beans" sample in navigation pane of the Samples Gallery.
2. In the content pane, under "Publish and Subscribe", choose "TechNotes".
The page that is displayed contains the information that is needed to setup the embedded JMS provider for this sample (described next).
3. Start the WebSphere Administrative Console and navigate to *<your server>*->**Resources->WebSphere JMS Provider**.
4. In the content pane, under "Additional Properties" choose **WebSphere Topic Connection Factories**.
5. Click the **New** button.
6. Enter the values given on the TechNotes page under "Defining properties for topic connection factory".
7. Click the **Apply** button then click the **OK** button.
8. Choose **WebSphere JMS Provider** in the content pane.
9. In the content pane, under "Additional Properties" choose **WebSphere Topic Destinations**.
10. Create four new topics with the property values given on the TechNotes page under "Defining properties for topics".
11. In the WebSphere Administrative Console, navigate to *<your server>*->**Servers->Application Servers**.

12. In the content pane, choose on the name of your server, then on **Message Listener Service**, then on **Listener Ports**, then on **SamplePubSubListenerPort**.
13. The values for these properties on the TechNotes are incorrect. Verify that the following property values are set:

Initial State	Started
Connection Factory JNDI name	Sample/JMS/TCF
Destination JNDI name	Sample/JMS/listen

14. Choose the "Message-driven beans" sample in the navigation pane of the Samples Gallery.
15. In the content pane, under "Publish and Subscribe", click **Configure and Run**.
16. Follow the instructions there for running the sample and confirming that the server received the message by examining the standard output log file.



The standard output log file for your server is called `SystemOut.log` and can be found under `install_root/Appserver/logs/<your server>`.

Get the Point-to-Point Sample Working

The point-to-point sample consists of an application client that sends a message to a queue and an MDB that is configured to be triggered by messages on that queue. The MDB takes the message it receives and sends it to a second queue where the client receives it back and compares it to the original message it sent.

1. Choose the "Message-driven beans" sample in the navigation pane of the Samples Gallery.

2. In the content window, under "Point-to-Point", choose "TechNotes".

The page that is displayed contains the information that is needed to setup the embedded JMS provider for this sample (described next).

3. In the WebSphere Administrative Console, navigate to: *<your server>*->**Resources**->**WebSphere JMS Provider**.
4. In the content pane, under "Additional Properties" choose **WebSphere Queue Connection Factories**.
5. Click the **New** button.
6. Enter the values given on the TechNotes page under "Defining properties for queue connection factory".
7. Click the **Apply** button then click the **OK** button.

8. Choose **WebSphere JMS Provider** in the content pane.
9. In the content pane, under "Additional Properties" choose **WebSphere Queue Destinations**.
10. Create two new queues with the property values given on the TechNotes page under "Defining properties for queues".
11. In the WebSphere Administrative Console, navigate to *<your server>->Servers->Application Servers*.
12. In the content pane, choose on the name of your server, then on **Message Listener Service**, then on **Listener Ports**, then on **SamplePtoPListenerPort**.
13. The values for these properties on the TechNotes page are incorrect. Verify that the following property values are set:

Initial State	Started
Connection Factory JNDI name	Sample/JMS/QCF
Destination JNDI name	Sample/JMS/Q1

14. Choose the "Message-driven beans" sample in the navigation pane of the Samples Gallery.
15. In the content pane, under "Point-to-Point", click **Configure and Run**.
16. Follow the instructions there for running the sample and confirming that the MDB received and sent back the message to the client.

Get the Sample MDB running with TIBCO Enterprise Message Service

Create the TIBCO Enterprise Message Service Administered Objects

1. Start the TIBCO Enterprise Message Service server.
2. Start the admin tool and enter the following commands:


```
> create factory sample.TCF topic
> create factory sample.QCF queue
> create topic sample.*
> create jndiname sample.listen topic sample.*
> create topic sample.weather
> create topic sample.sport
> create topic sample.news
> create queue sample.Q1
> create queue sample.Q2
```

Configure WebSphere for the TIBCO Enterprise Message Service JNDI Provider

1. Create a text file called `jndi.properties` in the directory `<install_root>\AppServer\lib\ext`.
2. Add the following line into the file:


```
java.naming.factory.url.pkgs=com.tibco.tibjms.naming
```
3. Save the `jndi.properties` file.

This allows both the WebSphere application server and client container to find the TIBCO Enterprise Message Service URLConnectionFactory when it encounters the `tibjmsnaming` JNDI naming scheme.

Add TIBCO Enterprise Message Service as a JMS Provider to the Application Server

1. Start the WebSphere application server (if you have not already done so).
2. Start the WebSphere Administrative Console.
3. Expand **Resources** and choose **Generic JMS Providers**.
4. Click the **New** button.

5. Enter the following values for the given properties:

Name	TIBCO
Description	TIBCO Enterprise Message Service
Classpath	C:\tibco\EMS\clients\java\tibjms.jar
External Initial Context Factory	com.tibco.tibjms.naming.TibjmsInitialContextFactory
External Provider URL	tibjmsnaming://localhost:7222

6. Click the **OK** button.
7. Click the **Save** button on the task bar at the top of the console window.
8. To have the changes take effect immediately, stop and restart the application server.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Resources > Messaging > Using JMS and messaging in applications > Administering JMS WebSphere support > Installing and configuring a JMS provider > Defining a generic JMS provider.**

Configure JNDI Bindings for TIBCO Enterprise Message Service Connection Factories for the Application Server

1. From the WebSphere Administrative Console, expand **Resources** and choose **Generic JMS Providers**.
2. In the content pane, choose **TIBCO**.
3. Scroll down and choose **JMS Connection Factories**.
4. Click the **New** button.
5. Enter the following values for the given properties:

Name	TIBCOConnectionFactory1
Type	TOPIC
JNDI Name	jms/ConnectionFactory1
Description	Sample Topic ConnectionFactory
External JNDI Name	tibjmsnaming://localhost/sample.TCF

6. Click the **OK** button.
7. Click the **New** button.

- Enter the following values for the given properties:

Name	TIBCOConnectionFactory
Type	QUEUE
JNDI Name	jms/ConnectionFactory
Description	Sample Queue ConnectionFactory
External JNDI Name	tibjmsnaming://localhost/sample.QCF

- Click the **OK** button.
- Click the **Save** button on the task bar of the Administrative Console (and **Save** again to confirm),
- To have the changes take effect immediately, stop and restart the application server.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Resources > Messaging > Using JMS and messaging in applications > Administering JMS WebSphere support > Configuring JMS provider resources > Configuring resources for a generic JMS provider > Configuring a JMS connection factory, generic JMS provider.**

Configure JNDI Bindings for TIBCO Enterprise Message Service Destinations for the Application Server

- From the WebSphere Administrative Console, expand **Resources** and choose **Generic JMS Providers**.
- In the content pane, choose **TIBCO**.
- Scroll down and choose **JMS Destinations**.
- Click the **New** button.
- Enter the following values for the given properties:

Name	Listen
Type	TOPIC
JNDI Name	jms/listen
Description	Sample Listen Topic
External JNDI Name	tibjmsnaming://localhost/sample.listen

- Click the **OK** button.

7. Repeat the previous steps to create the following additional destinations:

Name	Type	JNDI Name	Description	External JNDI Name
News	TOPIC	jms/news	Sample News Topic	tibjmsnaming://localhost/sample.news
Sport	TOPIC	jms/sport	Sample Sport Topic	tibjmsnaming://localhost/sample.sport
Weather	TOPIC	jms/weather	Sample Weather Topic	tibjmsnaming://localhost/sample.weather
Q1	QUEUE	jms/Q1	Sample Q1 Queue	tibjmsnaming://localhost/sample.Q1
Q2	QUEUE	jms/Q2	Sample Q2 Queue	tibjmsnaming://localhost/sample.Q2

8. Click the **Save** button on the task bar of the Administrative Console (and **Save** again to confirm).
9. To have the changes take effect immediately, stop and restart the application server.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Resources > Messaging > Using JMS and messaging in applications > Administering JMS WebSphere support > Configuring JMS provider resources > Configuring resources for a generic JMS provider > Configuring a JMS destination, generic JMS provider.**

Create new Listener Ports for TIBCO Enterprise Message Service

1. From the WebSphere Administrative Console, expand **Servers** and choose **Application Servers**.
2. In the content pane, choose the name of the application server.
3. In the Additional Properties Table, select **Message Listener Service**.
4. In the content pane, select **Listener Ports**.
5. In the content pane, click the **New** button.

6. Enter the following values for the given listener port properties:

Name	TIBCOPTtoPLListenerPort
Initial State	Started
Description	Listener Port for TIBCO Point to Point
ConnectionFactory JNDI Name	jms/ConnectionFactory
Destination JNDI Name	jms/Q1

7. Click the **OK** button.
8. Repeat the previous steps to create another listener port with the following property values:

Name	TIBCOPubSubListenerPort
Initial State	Started
Description	Listener Port for TIBCO PubSub
ConnectionFactory JNDI Name	jms/ConnectionFactory1
Destination JNDI Name	jms/listen

9. Click the **Save** button on the task bar of the Administrative Console (and **Save** again to confirm).
10. Stop and restart the application server to have the changes take effect.
11. After the application server has restarted, use the WebSphere Administrative Console to verify that the new listener ports are in their proper initial state.

To do this, expand **Servers->Application Servers**, then choose your server name in the content pane, then on **Message Listener Service** and then on **Listener Ports**. The new TIBCO listener ports should have a solid green arrow under the status column indicating that they are started.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Applications > EJB modules > Using message-driven beans in applications > Configuring message listener resources for message-driven beans > Adding a new listener port**.

Reassemble the Sample MDBs to Use the New TIBCO Enterprise Message Service Listener Ports

1. Start the WebSphere Application Assembly Tool.
2. Open the `MDBSamples.ear` file located in the `<install_root>/AppServer/samples/lib/MessageDrivenBeans` directory.

3. In the navigation pane, expand **MDBSamples->EJB Modules->PSSampleMDB.jar**.
4. Choose **Message Driven Beans**, then in the content pane, choose **PSSampleMDB**.
5. Click the **Bindings** tab in the property pane.
6. Change the value of the Listener Port Name from SamplePubSubListenerPort to TIBCOPubSubListenerPort.
7. Click the **Apply** button.
8. In the navigation pane, expand **MDBSamples->EJB Modules->PtoPSampleMDB.jar**.
9. Choose **Message Driven Beans**, then in the content pane, choose **PtoPSampleMDB**.
10. Click the **Bindings** tab in the property pane.
11. Change the value of the Listener Port Name from SamplePtoPLListenerPort to TIBCOPtoPLListenerPort.
12. Click the **Apply** button.
13. Choose **File->Save** from the menu.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Applications > EJB modules > Using message-driven beans in applications > Deploying an enterprise application to use message-driven beans > Configuring deployment attributes for a message-driven bean**.

Redefine the Resource Reference and Resource Environment Reference for the Point-to-Point Sample MDB

1. In the navigation pane of the WebSphere Application Assembly Tool, under **MDBSamples**, expand **EJBModules->PtoPSampleMDB.jar->Message Driven Beans->PtoPSampleMDB**.
2. Choose **Resource References**. The name **JMS/SamplePPQCF** should appear in the content pane.
3. Click the **Bindings** tab.
4. Change the value of JNDI Name from Sample/JMS/QCF to tibjmsnaming://localhost/sample.QCF.
5. Click the **Apply** button.

6. In the navigation pane, choose **Resource Environment References**. The name **JMS/SampleOutputQueue** should appear in the content pane.
7. Click the **Bindings** tab.
8. Change the value of JNDI Name from `Sample/JMS/Q2` to `tibjmsnaming://localhost/sample.Q2`.
9. Click the **Apply** button.
10. Choose **File->Save** from the menu.

Redefine the Resource Environment References in the Application Client Samples

1. Expand **MDBSamples->Application Clients->PSSampleClient->Resource Environment References**.
2. In the content pane, choose **jms/news** and then click the **Bindings** tab.
3. Change the value of the JNDI name from `thisNode/servers/server1/Sample/JMS/news` to `tibjmsnaming://localhost/sample.news`.
4. Click the **Apply** button.
5. Choose **jms/sport** and then click the **Bindings** tab.
6. Change the value of the JNDI name from `thisNode/servers/server1/Sample/JMS/sport` to `tibjmsnaming://localhost/sample.sport`.
7. Click the **Apply** button.
8. Choose **jms/weather** and then click the **Bindings** tab.
9. Change the value of the JNDI name from `thisNode/servers/server1/Sample/JMS/weather` to `tibjmsnaming://localhost/sample.weather`.
10. Expand **MDBSamples->Application Clients->PtoPSampleClient->Resource Environment References**.
11. Choose **jms/Q1** and then click the **Bindings** tab.
12. Change the value of the JNDI name from `thisNode/servers/server1/Sample/JMS/Q1` to `tibjmsnaming://localhost/sample.Q1`.
13. Click the **Apply** button.
14. Choose **jms/Q2** and then click the **Bindings** tab.

15. Change the value of the JNDI name from
`thisNode/servers/server1/Sample/JMS/Q2` to
`tibjmsnaming://localhost/sample.Q2`.
16. Click the **Apply** button.
17. Choose **File->Save**, then **File->Close** to save and then close the Application Assembly tool.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Applications > Client Modules > Using application clients > Developing J2EE application client code**.

Add TIBCO Enterprise Message Service as a JMS Provider to the Application Client

1. Start the WebSphere Application Client Resource Configuration Tool from a console window by entering the following command:
`install_root\AppServer\bin>clientConfig`
2. Open the `MDBSamples.ear` file located in the
`<install_root>/AppServer/samples/lib/MessageDrivenBeans` directory.
3. Expand `PSSampleClient.jar`.
4. Right-click on **JMS Providers** and select **New**.
5. Enter the following values for the given properties:

Name	TIBCO
Description	TIBCO Enterprise Message Service
Classpath	C:\Tibco\EMS\clients\java\tibjms.jar
ContextFactory Class	com.tibco.tibjms.naming.TibjmsInitialContextFactory
Provider URL	tibjmsnaming://localhost:7222

6. Click the **OK** button.
7. Repeat the previous three steps for `PtoPSampleClient.jar`.
8. Save the EAR file by choosing **File->Save** from the menu.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Applications > Client Modules > Using application clients > Deploying application clients > Configuring Java messaging client resources > Configuring new JMS providers with the Application Client Resource Configuration Tool**.

Configure the JNDI bindings for TIBCO Enterprise Message Service Connection Factories for the Application Client

1. In the Application Client Resource Configuration Tool for the MDBSamples.ear file, expand PSSampleClient.jar->JMS Providers->TIBCO.
2. Right-click on **JMS Connection Factories** and select **New**.
3. Enter the following values for the given properties:

Name	TIBCOConnectionFactory1
Description	Sample Topic Connection Factory
JNDI Name	jms/ConnectionFactory1
External JNDI Name	tibjmsnaming://localhost/sample.TCF
Connection Type	TOPIC

4. Click the **OK** button.
5. Repeat the previous three steps for PtoPSampleClient.jar using the following values:

Name	TIBCOConnectionFactory
Description	Sample Queue Connection Factory
JNDI Name	jms/ConnectionFactory
External JNDI Name	tibjmsnaming://localhost/sample.QCF
Connection Type	QUEUE

6. Save the EAR file by choosing **File->Save** from the menu.
7. Close the MDBSamples.ear file (**File->Close**).

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Applications > Client Modules > Using application clients > Deploying application clients > Configuring Java messaging client resources > Generic JMS connection factory settings for application clients**.

Update the Deployed Application on the Server

1. From the WebSphere Administrative Console, expand **Applications** and click on **Enterprise Applications**.
2. Check the box in front of **MDBSamples** and click the **Update** button.

3. Click the **Browse** button and locate the `MDBSamples.ear` file. On Windows, by default, it is located in: `C:\Program Files\WebSphere\AppServer\samples\lib\MessageDrivenBeans`.
4. Click the **Next** button.
5. Do not change any of the default settings on this page.
6. Click the **Next** button.
7. The "Step 1, Provide options to perform the installation" page appears. Do not change any of the default settings on this page.
8. Click the **Next** button.
9. The "Step 2, Provide Listener Ports for Messaging Beans" appears. It should already contain the names of the new listener ports previously created.
10. Click the **Next** button.
11. The "Step 3, Map resource references to resources" page appears. It should already contain the binding for the `jms/SamplePPQCF` reference for the `PtoPSampleMDB`.
12. Click the **Next** button.
13. The "Step 4, Map resource env entry references to resources" page appears. It should already contain the binding for the `jms/SampleOutputQueue` reference for the `PtoPSampleMDB`.
14. Click the **Next** button.
15. The "Step 5, Map virtual hosts for web modules" page appears. Do not change any of the default settings on this page.
16. Click the **Next** button.
17. The "Step 6, Map modules to application servers" page appears. Do not change any of the default settings on this page.
18. Click the **Next** button.
19. The "Step 7, Summary" page appears.
20. Click the **Finish** button.
21. The message "Application MDBSamples installed successfully" appears in the content window.
22. Choose **Save to Master Configuration**.
23. Click **Save** again.

More information about this task can be found in the WebSphere Application Server Version 5 Documentation On-line InfoCenter under: **Applications > Deployment > Deploying and managing applications > Updating Applications.**

Run the Samples Application Client

1. From the `<install_root>\samples\bin\MessageDrivenBeans` directory, type: `RunPSClient`. You should see the same results as you saw in part I for the publish/subscribe sample.
2. From the `<install_root>\samples\bin\MessageDrivenBeans` directory, type: `RunPtoPClient`. You should see the same results as you saw in part I for the point-to-point sample.

Modify the Samples to Use SSL Communications

This section describes how to modify the above samples to use SSL communications between the TIBCO Enterprise Message Service server and WebSphere application server and client container. This section assumes you have already set up and run the samples over unencrypted connections detailed in the previous sections.

Enable SSL in the TIBCO Enterprise Message Service Server

In `C:\tibco\EMS\bin\tibemsd.conf`, add the following lines:

```
listen = ssl://localhost:7243

ssl_server_identity = certs/server.cert.pem
ssl_server_key = certs/server.key.pem
ssl_password = password

listen = tcp://localhost:7222
```

These lines explicitly set the tcp and ssl listen ports and specify the three required server-side SSL parameters: identity, private key, and password.

Save the file, stop and restart the TIBCO Enterprise Message Service server. When it restarts you should see messages like the following in the console window confirming SSL is enabled:

```
2003-01-13 13:48:34 Secure Socket Layer is enabled.
2002-01-13 13:48:34 Accepting connections on ssl://localhost:7243.
2002-01-13 13:48:34 Accepting connections on tcp://localhost:7222.
```

Create JNDI Names for the SSL Queue and Topic Connection Factories

TIBCO Enterprise Message Service is pre-configured with a sample SSL queue and topic connection factory. This step will create new JNDI names for the sample connection factories that are then be used throughout the rest of this section.

1. Verify that the SSL connection factories exist by starting the `tibemsadmin` tool and entering the command `show factories`. The names `SSLQueueConnectionFactory` and `SSLTopicConnectionFactory` should be among the names displayed.
2. Create new JNDI names for the existing SSL connection factories by entering the following commands:

```
> create jndiname sample.SSLQCF jndiname SSLQueueConnectionFactory
> create jndiname sample.SSLTCF jndiname SSLTopicConnectionFactory
```

Add the Additional SSL JNDI Properties to WebSphere

Edit the `jndi.properties` file created in Configure WebSphere for the TIBCO Enterprise Message Service JNDI Provider on page 126 and add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
```

These properties specify that the "SSL" protocol should be used for JNDI lookups, and that host verification is turned off (the JMS client will trust any host).



For WebSphere 5.1, add the following line in addition to those above:

```
com.tibco.tibjms.naming.ssl_vendor=j2se-default
```

Configure SSL Communications Between the Application Server and the TIBCO Enterprise Message Service Server

This procedure adds the additional jar files required for SSL to the CLASSPATH. It also modifies the external provider URL and the external JNDI name properties of the TIBCO JMS provider within the application server.

This causes the application server to connect to the SSL port on the TIBCO Enterprise Message Service server for JNDI lookups of administered objects. Additionally, the connection factory external JNDI names are modified to specify SSL connection factories (connection factories that create SSL-based connections).

1. From the WebSphere Administrative Console, expand **Resources->Generic JMS Providers** and choose **TIBCO** in the content pane.
2. Add the following lines to the **Classpath** property value:


```
C:\tibco\ems\clients\java\jcert.jar
C:\tibco\ems\clients\java\jnet.jar
C:\tibco\ems\clients\java\jsse.jar
C:\tibco\ems\clients\java\tibcrypt.jar
```
3. Change the port number of the **External Provider URL** property from 7222 to 7243.
4. Click the **Apply** button.
5. In the content pane under **Additional Properties**, choose **JMS Connection Factories**.
6. Choose **TIBCO Connection Factory**.
7. For the **External JNDI Name** property value, add port 7243 after the host specification and change the name of the factory that is looked up to `sample.SSLQCF`.

That is, change `tibjmsnaming://localhost/sample.QCF` to `tibjmsnaming://localhost:7243/sample.SSLQCF`.

8. Click the **OK** button.
9. Repeat the above steps for **TIBCO Connection Factory1**, changing `tibjmsnaming://localhost/sample.TCF` to `tibjmsnaming://localhost:7243/sample.SSLTCF`.
10. Navigate to **Generic JMS Providers->TIBCO**.
11. Choose **JMS Destinations**.
12. Modify the **External JNDI Name** value for each of the destinations to specify port 7243.
13. Click the **Save** button on the task bar of the Administrative Console (and **Save** again to confirm).
14. Stop and restart the application server to allow the changes to take effect.

Configure SSL Communications between the Point-to-Point Sample MDB and the TIBCO Enterprise Message Service Server

This procedure modifies the resource reference and the resource environment references of the point-to-point sample MDB. This causes the sample point-to-point MDB to connect to the SSL port on the TIBCO Enterprise Message Service server for JNDI lookups of administered objects.

Additionally, the connection factory external JNDI name is modified to specify a SSL connection factory (connection factory that creates SSL-based connections).

1. Start the WebSphere Application Assembly Tool.
2. Open the `MDBSamples.ear` file located in the `<install_root>/AppServer/samples/lib/MessageDrivenBeans` directory.
3. Expand **EJBModules->PtoPSampleMDB.jar->Message Driven Beans->PtoPSampleMDB**.
4. Choose **Resource References**. The name `JMS/SamplePPQCF` should appear in the content pane.
5. Click the **Bindings** tab.
6. Change the value of JNDI Name from `tibjmsnaming://localhost/sample.QCF` to `tibjmsnaming://localhost:7243/sample.SSLQCF`.
7. Click the **Apply** button.

8. In the navigation pane, choose **Resource Environment References**. The name **JMS/SampleOutputQueue** should appear in the content pane.
9. Click the **Bindings** tab.
10. Change the value of JNDI Name from
`tibjmsnaming://localhost/sample.Q2` to
`tibjmsnaming://localhost:7243/sample.Q2`.
11. Click the **Apply** button.
12. Choose **File->Save** from the menu.

Configure SSL Communications between the Application Client and the TIBCO Enterprise Message Service Server

1. In the Application Assembly Tool, expand **MDBSamples->Application Clients->PSSampleClient->Resource Environment References**.
2. In the content pane, choose **jms/news** and then click the **Bindings** tab.
3. Change the value of the JNDI name from
`tibjmsnaming://localhost/sample.news` to
`tibjmsnaming://localhost:7243/sample.news`.
4. Click the **Apply** button.
5. Repeat the above steps for the sport and weather destinations as well.
6. Expand **MDBSamples->Application Clients->PtoPSampleClient->Resource Environment References**.
7. In the content pane, choose **jms/Q1** and click the **Bindings** tab.
8. Change the value of the JNDI name from
`tibjmsnaming://localhost/sample.Q1` to
`tibjmsnaming://localhost:7243/sample.Q1`.
9. Click the **Apply** button.
10. Repeat the above steps for the Q2 destination.
11. Save the MDBSamples.ear file (**File->Save**).
12. Exit the Application Assembly Tool.
13. Start the WebSphere Application Client Resource Configuration Tool from a console window by entering:
`<install_root>\AppServer\bin>clientConfig`
14. Open the MDBSamples.ear file located in the
`<install_root>/AppServer/samples/lib/MessageDrivenBeans` directory.

15. Expand **PSSampleClient.jar->JMS Providers**.
16. Right-click on **TIBCO** and select **Properties**.
17. Append the following line to the end of the value for the **Class Path** property:


```

;C:\tibco\ems\clients\java\jcert.jar;
C:\tibco\ems\clients\java\jnet.jar;
C:\tibco\ems\clients\java\jsse.jar;
C:\tibco\ems\clients\java\tibcrypt.jar
      
```
18. Change the value of the **Provider URL** property from
 tibjmsnaming://localhost:7222 to tibjmsnaming://localhost:7243.
19. Click the **OK** button.
20. Expand **PSSampleClient.jar->JMS Providers->TIBCO->JMS Connection Factories**.
21. Right-click on **TIBCOConnectionFactory1** and select **Properties**.
22. Change the value of the **External JNDI Name** property from
 tibjmsnaming://localhost/sample.TCF to
 tibjmsnaming://localhost:7243/sample.SSLTCF.
23. Click the **OK** button.
24. Repeat the above steps for **PtoPSampleClient.jar**, again appending to the **Class Path**:


```

;C:\tibco\ems\clients\java\jcert.jar;
C:\tibco\ems\clients\java\jnet.jar;
C:\tibco\ems\clients\java\jsse.jar;
C:\tibco\ems\clients\java\tibcrypt.jar
      
```

Change tibjmsnaming://localhost:7222 to
 tibjmsnaming://localhost:7243.

Also change tibjmsnaming://localhost/sample.QCF to
 tibjmsnaming://localhost:7243/sample.SSLQCF.
25. Save the EAR file by choosing **File->Save** from the menu.
26. Close the **MDBSamples.ear** file.
27. Exit the **Application Client Resource Configuration Tool**.

Update the Deployed Application on the Server

Follow the same procedure to update the deployed application on the server as in the previous section.

Run the Samples Application Client

Run the samples application client again. You should see the same results.

Chapter 12 **Integrating With Sun Java System Application Server 7**

This chapter describes integrating TIBCO Enterprise Message Service with Sun Java System Application Server 7.

Topics

- *Run the MDB Sample with Built-In JMS, page 144*
- *Run the MDB Sample with TIBCO EMS, page 145*
- *Run the MDB Sample with TIBCO EMS using SSL, page 147*

Run the MDB Sample with Built-In JMS

These steps establish baseline behavior for the sample message-driven bean (MDB) in JMS.

- Configure
1. Ensure that *install_dir*\bin is in the PATH.
 2. Start the application server.
 3. Change directory to *install_dir*\samples\ejb\mdb\simple\src, and run the following commands:

Build `asant`

Deploy `asant deploy-jms-resource`
 `asant deploy`

The server log should indicate that the MDB is successfully deployed.

- Run
4. Change directory to
install_dir\domains\domain1\server1\applications\j2ee-apps\mdb-simple_1, and run this command:

```
appclient -client mdb-simpleClient.jar -name SimpleMessageClient
-textauth
```

The console should display these lines:

```
Sending message: This is message 1
Sending message: This is message 2
Sending message: This is message 3
```

The server log should display these lines:

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

- Clean Up
5. Change directory to *install_dir*\samples\ejb\mdb\simple\src, and run these commands:

```
asant clean
asant undeploy
```

6. Remove the directory:
install_dir\domains\domain1\server1\applications\j2ee-apps\mdb-simple_1

Run the MDB Sample with TIBCO EMS

This section demonstrates the procedure for using the sample MDB with EMS.

Configure Application Server

In a web browser, access Sun's Java System Administration Tool at `http://host:admin_port`

1. In the left frame, navigate the tree to the folder `AppServer Instances->server1`
2. In the right frame, click the JVM Settings tab, then the Path Settings link.
3. In the Classpath Suffix box, enter the filename `C:\tibco\ems\clients\java\tibjms.jar`
4. Click the Save button.
5. To propagate these modifications to the server, click the General tab, then the Apply Changes button. Then stop and restart the server instance.

Register JMS Resources with Application Server

6. Change directory to `install_dir\bin`, and run these commands:

```
asadmin multimode
```

```
asadmin>export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=password AS_ADMIN_HOST=hostname
AS_ADMIN_PORT=port AS_ADMIN_INSTANCE=server
```

```
asadmin>create-jndi-resource --jndilookupname QueueConnectionFactory --resourcetype
javax.jms.QueueConnectionFactory --factoryclass
com.tibco.tibjms.naming.TibjmsInitialContextFactory --enabled=true --property
java.naming.provider.url=tibjmsnaming\://localhost\:7222 jms/MyQcf
```

```
asadmin>create-jndi-resource --jndilookupname queue.sample --resourcetype
javax.jms.Queue --factoryclass com.tibco.tibjms.naming.TibjmsInitialContextFactory
--enabled=true --property java.naming.provider.url=tibjmsnaming\://localhost\:7222
jms/MyQueue
```

```
asadmin>reconfig server1
```

Run the Sample

7. Ensure that the EMS server is running with the default configuration.

8. Change directory to *install_dir*\bin. Modify *appclient.bat* by adding *C:\tibco\ems\clients\java\tibjms.jar* to *JVM_CLASSPATH*.
9. Change directory to *install_dir*\samples\ejb\mdb\simple\src, then build and deploy the sample using the following commands:

Build *asant*

Deploy *asant deploy*

The server log should indicate that the MDB is successfully deployed.

- Run 10. Change directory to
 install_dir\domains\domain1\server1\applications\j2ee-apps\mdb-simple_1, and run this command:

```
appclient -client mdb-simpleClient.jar -name SimpleMessageClient
-textauth
```

The console should display these lines:

```
Sending message: This is message 1
Sending message: This is message 2
Sending message: This is message 3
```

The server log should display these lines:

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

- Clean Up 11. Clean up the build and undeploy the sample MDB.

Change directory to *install_dir*\samples\ejb\mdb\simple\src, and run these commands:

```
asant clean
asant undeploy
```

12. Remove the directory:
 install_dir\domains\domain1\server1\applications\j2ee-apps\mdb-simple_1

13. Undeploy JNDI resources:

```
asadmin>delete-jndi-resource jms/MyQcf
asadmin>delete-jndi-resource jms/MyQueue
asadmin>reconfig server1
```

Run the MDB Sample with TIBCO EMS using SSL

Configure the EMS Server

1. Ensure that these parameters are set in `tibemsd.conf` *before* starting the EMS server:

```
listen                = ssl://localhost:7243
ssl_server_identity   = certs/server.cert.pem
ssl_server_key        = certs/server.key.pem
ssl_password          = password
```

Java Security Policy

2. If you use the default installation (and depending on the local Java setting), you must grant the following permissions in your J2SDK policy file `/jre/lib/security/java.policy`.

```
permission java.util.PropertyPermission "com.sun.net.ssl.dhKeyExchangeFix",
"write";

permission java.util.PropertyPermission "java.protocol.handler.pkgs", "write";

permission java.security.SecurityPermission "putProviderProperty.SunJSSE";

permission java.security.SecurityPermission "insertProvider.SunJSSE";
```

Configure Application Server

3. In a web browser, access Sun's Java System Administration Tool at `http://host:admin_port`
4. In the left frame, navigate the tree to the folder `AppServer Instances->server1`
5. In the right frame, click the JVM Settings tab, then the Path Settings link.
6. In the Classpath Suffix box, enter the following filenames, and click the Save button:

```
C:\tibco\ems\clients\java\tibjms.jar
C:\tibco\ems\clients\java\jcert.jar
C:\tibco\ems\clients\java\jnet.jar
C:\tibco\ems\clients\java\jsse.jar
C:\tibco\ems\clients\java\tibcrypt.jar
```

7. To propagate these modifications to the server, click the General tab, then the Apply Changes button. Then stop and restart the server instance.
8. If you are using a console configured in the previous section, omit this step and continue to the next step.

If you have started a new console, change directory to *install_dir*\bin, and run these commands:

```
asadmin multimode
```

```
asadmin>export AS_ADMIN_USER=admin AS_ADMIN_PASSWORD=password
AS_ADMIN_HOST=localhost AS_ADMIN_PORT=4848 AS_ADMIN_INSTANCE=server1
```

9. In all cases, run the following commands in the asadmin interface:

```
asadmin>create-jndi-resource --jndilookupname SSLQueueConnectionFactory
--resourcetype javax.jms.QueueConnectionFactory --factoryclass
com.tibco.tibjms.naming.TibjmsInitialContextFactory --enabled=true --property
java.naming.provider.url=tibjmsnaming\://localhost\:7243:com.tibco.tibjms.naming.se
curity_protocol=ssl:com.tibco.tibjms.naming.ssl_enable_verify_host=false jms/MyQcf
```

```
asadmin>create-jndi-resource --jndilookupname queue.sample --resourcetype
javax.jms.Queue --factoryclass com.tibco.tibjms.naming.TibjmsInitialContextFactory
--enabled=true --property
java.naming.provider.url=tibjmsnaming\://localhost\:7243:com.tibco.tibjms.naming.se
curity_protocol=ssl:com.tibco.tibjms.naming.ssl_enable_verify_host=false
jms/MyQueue
```

```
asadmin>reconfig server1
```

10. Change directory to *install_dir*\samples\ejb\mdb\simple\src, then build and deploy the sample using the following commands:

Build	asant
Deploy	asant deploy

The server log should indicate that the MDB is successfully deployed.

11. Add *tibjms.jar*, *jcrt.jar*, *jnet.jar*, *jsse.jar*, and *tibcrypt.jar* to JVM_CLASSPATH in *appclient.bat*.

- | | |
|-----|---|
| Run | 12. Change directory to
<i>install_dir</i> \domains\domain1\server1\applications\j2ee-apps\mdb-simple_1, and run this command: |
|-----|---|

```
appclient -client mdb-simpleClient.jar -name SimpleMessageClient
-textauth
```

The console should display these lines:

```
Sending message: This is message 1
Sending message: This is message 2
Sending message: This is message 3
```

The server log should display these lines:

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

Clean Up 13. Clean up the build and undeploy the sample MDB.

Change directory to *install_dir*\Sample\ejb\mdb\Simple\src, and run these commands:

```
asant clean
asant undeploy
```

14. Remove the directory:

```
install_dir\domains\domain1\server1\applications\j2ee-apps\mdb-simple_1
```

15. Undeploy JNDI resources:

```
asadmin>delete-jndi-resource jms/MyQcf
asadmin>delete-jndi-resource jms/MyQueue
asadmin>reconfig server1
```


Index

A

application servers 10

C

container-managed transactions 12, 16, 26, 30, 40, 45

customer support xvi

I

integrating with third-party application servers 10

S

support, contacting xvi

T

technical support xvi

third-party application servers 10

transactions, container-managed 12, 16, 26, 30, 40, 45

