

TIBCO Enterprise Message Service™

Application Integration Guide

Software Release 5.1
February 2009

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 LICENSE.PDF) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. 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, TIBCO Adapter, Predictive Business, Information Bus, The Power of Now, TIBCO ActiveEnterprise, TIBCO Hawk, TIBCO Rendezvous, TIBCO Enterprise, TIBCO Enterprise Message Service, TIBCO SmartSockets and the TIBCO logo are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, JAVA EE, J2EE, 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.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 1997-2009 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Figures	vii
Tables	ix
Preface	xi
Related Documentation	xii
TIBCO Enterprise Message Service Documentation	xii
Other TIBCO Product Documentation	xii
Third Party Documentation	xiii
Typographical Conventions	xiv
How to Contact TIBCO Customer Support	xvii
 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	21
Adding the SSL JAR File to the CLASSPATH for the Client Program	22
Adding the SSL JNDI Properties for the Client Program	22
Modify and Rebuild the Client	22
Re-Run the Client Program	22
Container-Managed Transactions (XA)	24

Chapter 4 Integrating With Borland Enterprise Server 5.1	25
Configure Borland Enterprise Server to use TIBCO Enterprise Message Service	26
Configure TIBCO Enterprise Message Service for the Example Program	29
Configure Borland Enterprise Server for the Example Message Driven Bean	30
Using Container-Managed XA Transactions	30
Using XA Transactions That Are Not Container-Managed	32
Building and Deploying the Example MDB and the Example Client	34
Running This Example	35
Modifying This Example to use SSL Communications	36
Chapter 5 Integrating With WebLogic Server 10.0	39
Running the Example MDB with WebLogic Server	40
Configuring the Example MDB	41
Adding TIBCO Enterprise Message Service to the WebLogic CLASSPATH	41
Creating the JMS Destination Object Inside TIBCO EMS	41
Modifying the MDB Class File to Use TIBCO Enterprise Message Service Objects	42
Modifying the Client Program to Use TIBCO Enterprise Message Service JNDI	43
Rebuilding and Redeploying the Example MDB	45
Running the Example MDB Client	46
Modifying this Example to Use SSL Communication	47
Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH	47
Configure the TIBCO Enterprise Message Service Server for SSL	47
Modify the Example MDB to Use the SSL Protocol	48
Modify the Example Client Program for SSL-Based Communication	48
Rebuilding and Redeploying the Example MDB	48
Running the Example MDB Client with SSL	49
Modifying this Example to Use Container Managed Transactions and XA	50
Create a JMS Connection Factory That Supports XA	50
Modifying the MDB to Use Transactions	50
Modify the Example Client Program to Use Transactions	51
Rebuilding and Redeploying the Example MDB	51
Running the Example MDB Client with Transactions	51
Chapter 6 Integrating With WebLogic Server 9.2	53
Running the Example MDB with WebLogic Server	54
Configuring the Example MDB	55
Adding TIBCO Enterprise Message Service to the WebLogic CLASSPATH	55
Creating the JMS Destination Object Inside TIBCO EMS	56
Modifying the Sample MDB Class File to Use TIBCO Enterprise Message Service Objects	56
Modifying the Client Program to Use TIBCO Enterprise Message Service JNDI	57

Rebuilding and Redeploying the Example MDB	59
Running the Example MDB Client	60
Modifying this Example to Use SSL Communication	61
Configure the TIBCO Enterprise Message Service Server for SSL	61
Add the SSL JAR Files and New JNDI Properties File to the CLASSPATH	61
Modify the Example MDB to Use the SSL Protocol	62
Modify the Example Client Program for SSL-Based Communication	62
Rebuilding and Redeploying the Example MDB	62
Running the Example MDB Client with SSL	63
Modifying this Example to Use Container Managed Transactions and XA	64
Create a JMS Connection Factory That Supports XA	64
Modifying the MDB to Use Transactions	64
Modify the Example Client Program to Use Transactions	65
Rebuilding and Redeploying the Example MDB	65
Running the Example MDB Client with Transactions	65
Chapter 7 Integrating With IBM WebSphere Application Server Version 6.1	67
Overview of Integrating With IBM WebSphere	68
Create TIBCO Enterprise Message Service Administered Objects	69
Configure WebSphere to Use EMS as the JMS Provider	70
Add TIBCO Enterprise Message Service as a JMS Provider	70
Configure JNDI Bindings for the Connection Factories	71
Configure JNDI Bindings for the Destinations	72
Create new Listener Ports	73
Install the MDB to Use the Topic and Queue Listeners	74
Run the Sample Application Clients	75
Modify the Samples to Use SSL Communications	77
Enable SSL in the TIBCO Enterprise Message Service Server	77
Create JNDI Names for the SSL Queue and Topic Connection Factories	77
Add Additional SSL JNDI Properties to WebSphere	78
Configure SSL Communications Between the Application Server and the TIBCO Enterprise Message Service Server	78
Run the Samples Application Clients	79
Chapter 8 Integrating With IBM WebSphere Application Server Version 5	81
Overview of Integrating With IBM WebSphere	82
Get the sample MDB running with the WebSphere Embedded JMS Provider	83
Get the Publish and Subscribe Sample Working	83
Get the Point-to-Point Sample Working	84
Get the Sample MDB running with TIBCO Enterprise Message Service	86
Create the TIBCO Enterprise Message Service Administered Objects	86
Configure WebSphere for the TIBCO Enterprise Message Service JNDI Provider	86

Add TIBCO Enterprise Message Service as a JMS Provider to the Application Server	86
Configure JNDI Bindings for TIBCO Enterprise Message Service Connection Factories for the Application Server	87
Configure JNDI Bindings for TIBCO Enterprise Message Service Destinations for the Application Server	88
Create new Listener Ports for TIBCO Enterprise Message Service	89
Reassemble the Sample MDBs to Use the New TIBCO Enterprise Message Service Listener Ports	90
Redefine the Resource Reference and Resource Environment Reference for the Point-to-Point Sample MDB	91
Redefine the Resource Environment References in the Application Client Samples	92
Add TIBCO Enterprise Message Service as a JMS Provider to the Application Client	93
Configure the JNDI bindings for TIBCO Enterprise Message Service Connection Factories for the Application Client	94
Update the Deployed Application on the Server	95
Run the Samples Application Client.	96
Modify the Samples to Use SSL Communications	97
Enable SSL in the TIBCO Enterprise Message Service Server	97
Create JNDI Names for the SSL Queue and Topic Connection Factories	97
Add the Additional SSL JNDI Properties to WebSphere	98
Configure SSL Communications Between the Application Server and the TIBCO Enterprise Message Service Server	98
Configure SSL Communications between the Point-to-Point Sample MDB and the TIBCO Enterprise Message Service Server	99
Configure SSL Communications between the Application Client and the TIBCO Enterprise Message Service Server	100
Update the Deployed Application on the Server	101
Run the Samples Application Client.	101
Chapter 9 Integrating With Sun Java System Application Server 7	103
Run the MDB Sample with Built-In JMS	104
Run the MDB Sample with TIBCO EMS	105
Configure Application Server	105
Register JMS Resources with Application Server	105
Run the Sample	106
Run the MDB Sample with TIBCO EMS using SSL	107
Configure the EMS Server	107
Java Security Policy	107
Configure Application Server	107
Index	111

Figures

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

Tables

Table 1 General Typographical Conventions xiv

Table 2 Syntax Typographical Conventions xvi

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 messaging products.

Topics

- *Related Documentation, page xii*
- *Typographical Conventions, page xiv*
- *How to Contact TIBCO Customer Support, page xvii*

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 can be accessed only through the HTML documentation interface.
- *TIBCO Enterprise Message Service .NET API Reference* The .NET API reference can be accessed only through the HTML documentation interface.
- *TIBCO Enterprise Message Service Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release. 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[®]
- TIBCO SmartSockets[®]
- TIBCO Hawk[®]
- TIBCO EMS[®] Client for z/OS (CICS)
- TIBCO EMS[®] Client for z/OS (MVS)
- TIBCO EMS[®] Client for i5/OS

Third Party Documentation

- Java™ Message Service specification, available through <http://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.
- Java™ Authentication and Authorization Service (JAAS) *LoginModule Developer's Guide* and *Reference Guide*, available through <http://java.sun.com/products/jaas/>.

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
<i>TIBCO_HOME</i> <i>ENV_HOME</i> <i>EMS_HOME</i>	<p>Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as <i>TIBCO_HOME</i>. The value of <i>TIBCO_HOME</i> depends on the operating system. For example, on Windows systems, the default value is <code>C:\tibco</code>.</p> <p>Other TIBCO products are installed into an installation environment. Incompatible products and multiple instances of the same product are installed into different installation environments. The directory into which such products are installed is referenced in documentation as <i>ENV_HOME</i>. The value of <i>ENV_HOME</i> depends on the operating system. For example, on Windows systems the default value is <code>C:\tibco</code>.</p> <p>TIBCO Enterprise Message Service installs into a directory within <i>TIBCO_HOME</i>. This directory is referenced in documentation as <i>EMS_HOME</i>. The value of <i>EMS_HOME</i> depends on the operating system. For example on Windows systems, the default value is <code>C:\tibco\ems\5.1</code>.</p>
<code>code font</code>	<p>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:</p> <p>Use <code>MyCommand</code> to start the TIBCO foo process.</p>
bold code font	<p>Bold code font is used in the following ways:</p> <ul style="list-style-type: none">• In procedures, to indicate what a user types. For example: Type the username <code>admin</code>.• In large code samples, to indicate the parts of the sample that are of particular interest.• In command syntax, to indicate the default value. For example, if no parameter is specified, <code>MyCommand</code> is enabled: <p><code>MyCommand [enable disable]</code></p>

Table 1 General Typographical Conventions

Convention	Use
<i>italic font</i>	<p>Italic font is used in the following ways:</p> <ul style="list-style-type: none"> • To indicate a document title. For example: See <i>TIBCO BusinessWorks Concepts</i> for more details. • To introduce new terms. For example: A portal page may contain several <i>portlets</i>. Portlets are mini-applications that run in a portal. • To indicate a variable in a command or code syntax that you must replace. For example: <code>MyCommand <i>pathname</i></code>
Key combinations	<p>Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C.</p> <p>Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.</p>

Table 2 Syntax Typographical Conventions

Convention	Use
[]	<p>An optional item in a command or code syntax.</p> <p>For example:</p> <pre>MyCommand [optional_parameter] required_parameter</pre>
	<p>A logical 'OR' that separates multiple items of which only one may be chosen.</p> <p>For example, you can select only one of the following parameters:</p> <pre>MyCommand param1 param2 param3</pre>
{ }	<p>A logical group of items. Other syntax notations may appear within each logical group.</p> <p>For example, the following command requires two parameters, which can be either <code>param1</code> and <code>param2</code> or <code>param3</code> and <code>param4</code>:</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command requires two parameters. The first parameter can be either <code>param1</code> or <code>param2</code> and the second can be either <code>param3</code> or <code>param4</code>:</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command can accept either two or three parameters. The first parameter must be <code>param1</code>. You can optionally include <code>param2</code> as the second parameter. And the last parameter is either <code>param3</code> or <code>param4</code>.</p> <pre>MyCommand param1 [param2] {param3 param4}</pre>

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>

- 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.



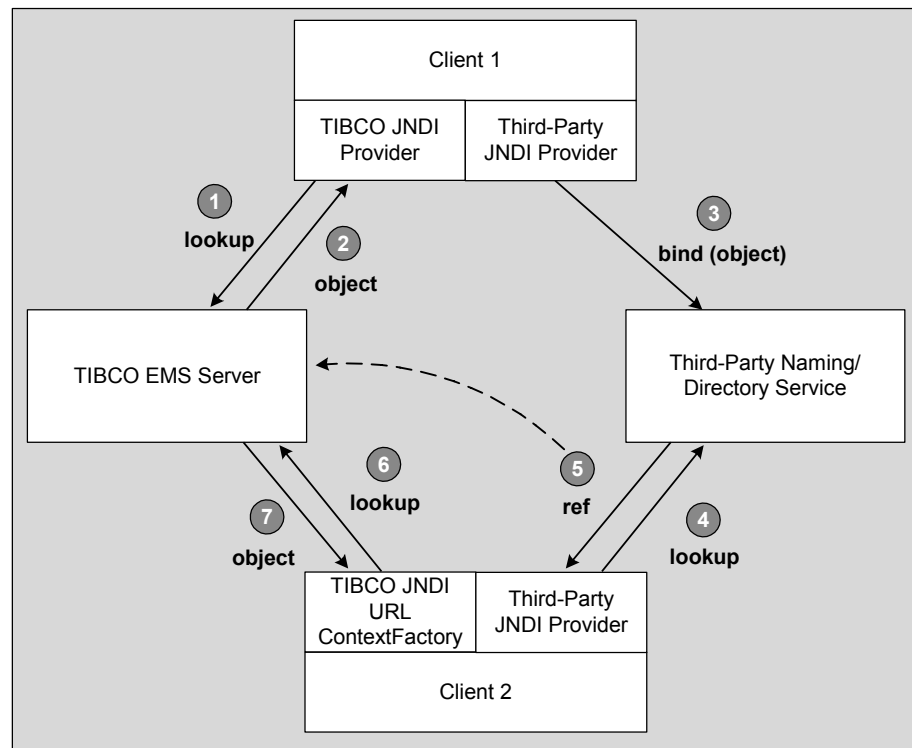
There is no automatic synchronization of administered objects between the TIBCO Enterprise Message Service server and any foreign naming/directory service. Keeping the two synchronized is the responsibility of the system administrator.

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

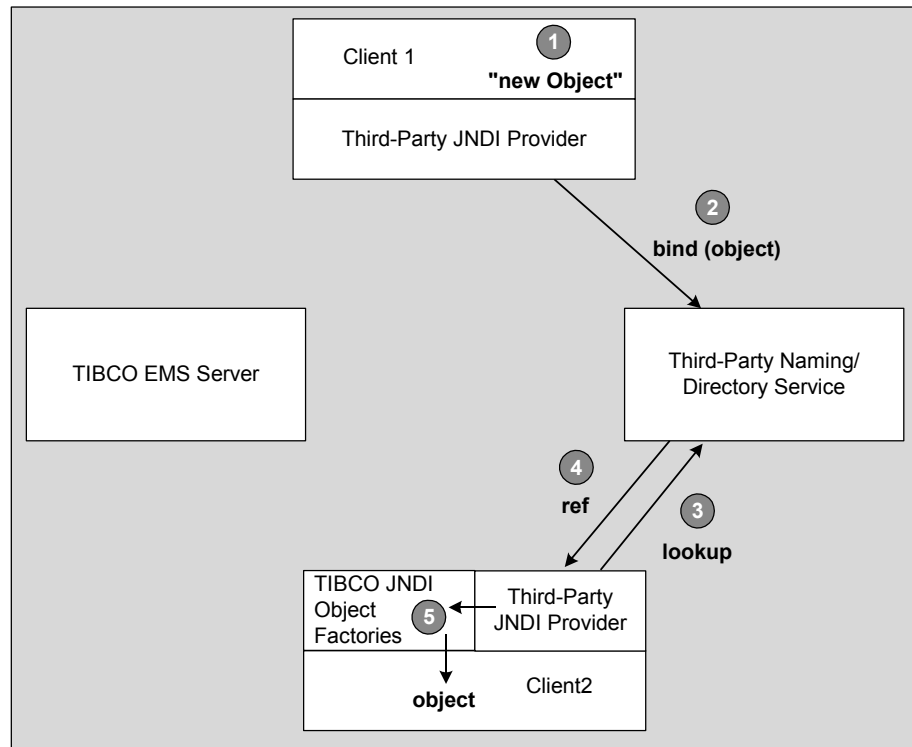


In [Figure 1](#), the following occurs:

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 4.0.2 from JBoss.org
- Borland Enterprise Server 5.1 from Borland
- WebLogic 9.2 and 10.0 with Service Pack 1 from BEA
- IBM WebSphere Application Server Version 5 and 6.1
- 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 `ConnectionConsumer` 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 4.0.2, page 12*](#)
- [*Get the Example MDB Working Using JBossMQ, page 13*](#)
- [*Get the Example MDB Working Using TIBCO Enterprise Message Service, page 16*](#)
- [*Modify the Example to use SSL Communications, page 20*](#)
- [*Container-Managed Transactions \(XA\), page 24*](#)

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 24.

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 24](#).

Queues and Connection Factories

1. Start the `tibemspd` server and the `tibemssadmin` 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 `tibemssadmin`:

```
> 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:/EMS_HOME/lib"
    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. Add the TIBCO Enterprise Message Service as the JMS provider for the connection factory. To do so, edit the file `%JBOSS_DEPLOY%\jms\jms-ds.xml` to change the `JmsProviderAdapterJNDI` property in the JMS XA Resource adapter to `TIBCOJMSProvider`. It should resemble these lines:

```
<!-- JMS XA Resource adapter, use this to get transacted JMS in beans -->
<tx-connection-factory>
  <jndi-name>JmsXA</jndi-name>
  <xa-transaction/>
  <rar-name>jms-ra.rar</rar-name>
  <connection-definition>org.jboss.resource.adapter.jms.JmsConnectionFactory
</connection-definition>
  <config-property name="SessionDefaultType"
    type="java.lang.String">javax.jms.Topic</config-property>
  <config-property name="JmsProviderAdapterJNDI"
    type="java.lang.String">java:/TIBCOJMSProvider</config-property>
  <max-pool-size>20</max-pool-size>
  <security-domain-and-application>JmsXARealm</security-domain-and-application>
</tx-connection-factory>
```

7. Reconfigure the `MDB` to use the queue connection factory reference for TIBCO Enterprise Message Service. To do so, edit the file

`<mdb-output>\META-INF\jboss.xml` as shown:

Change:

```
<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>ConnectionFactory</jndi-name>
```

```

        </resource-ref>
    </message-driven>
</enterprise-beans>
</jboss>

```

To:

```

<jboss>
  <resource-managers>
    <resource-manager>
      <res-name>queuefactoryref</res-name>
      <res-jndi-name>java:/JmsXA</res-jndi-name>
    </resource-manager>
  </resource-managers>

  <enterprise-beans>
    <message-driven>
      <ejb-name>TextMDB</ejb-name>
      <destination-jndi-name>queue/B</destination-jndi-name>
      <mdb-user>MDBUser</mdb-user>
      <mdb-passwd>MDBPassword</mdb-passwd>
      <resource-ref>
        <res-ref-name>jms/QCF</res-ref-name>
        <resource-name>queuefactoryref</resource-name>
      </resource-ref>
    </message-driven>
  </enterprise-beans>
</jboss>

```

This change modifies the MDB to use the queue connection factory for EMS. MDBUser and MDBPassword are used to authenticate the MDB user when it connects to the EMS server. If authorization is disabled in the server, the MDBUser and Password lines can be omitted.

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. 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
```

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

```
run
```

12. 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
```

13. Add `EMS_HOME\lib\tibjms.jar` to the `CLASSPATH` of the client program.
14. 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:/EMS_HOME/lib"
    archives="tibcrypt.jar" />
```

Configuring the TIBCO Enterprise Message Service Server for SSL

1. Start `tibemspd` in the working directory `EMS_HOME\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

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>
```

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.

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 File to the CLASSPATH for the Client Program

The following JAR file, 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:

```
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`. 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:

```
<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 16](#)).
 - 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
```

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 26*](#)
- [*Configure TIBCO Enterprise Message Service for the Example Program, page 29*](#)
- [*Building and Deploying the Example MDB and the Example Client, page 34*](#)
- [*Running This Example, page 35*](#)
- [*Modifying This Example to use SSL Communications, page 36*](#)

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.TibjmsBorlandXATopicConnectionFactory
  </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>
      </message-driven>
    </enterprise-beans>
  </ejb-jar>
```



```

        <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;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>NotSupported</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 `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 2008] stderr: [TibjmsSSL]: using security vendor 'j2se'
[Mon Oct 18 18:32:03 PST 2008] stderr: [TibjmsSSL]: WARNING: server verification is
disabled, will trust any server.
[Mon Oct 18 18:32:03 PST 2008] stderr: [TibjmsSSL]: client identity not set, using
empty identity.
[Mon Oct 18 18:32:07 PST 2008] stderr: [TibjmsSSL]: selected cipher:
SSL_RSA_WITH_RC4_128_SHA
[Mon Feb 18 18:32:08 PST 2008] stderr: [TibjmsSSL]: WARNING: server verification is
disabled, will trust any server.
[Mon Oct 18 18:32:08 PST 2008] stderr: [TibjmsSSL]: client identity not set, using
empty identity.
[Mon Oct 18 18:32:08 PST 2008] 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.
```


Chapter 5

Integrating With WebLogic Server 10.0

This chapter describes how to use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside WebLogic Server 10.0. The examples in this chapter use the preconfigured Examples server and the example MDB that comes bundled as a sample with WebLogic Server 10.0.

The examples in this chapter assume you have installed WebLogic Server 10.0 on a Windows platform in the directory `C:\bea`.



Before running these examples, you must install these patches from BEA: CR326377 and CR333566.

Topics

- [*Running the Example MDB with WebLogic Server, page 40*](#)
- [*Configuring the Example MDB, page 41*](#)
- [*Rebuilding and Redeploying the Example MDB, page 45*](#)
- [*Running the Example MDB Client, page 46*](#)
- [*Modifying this Example to Use SSL Communication, page 47*](#)
- [*Modifying this Example to Use Container Managed Transactions and XA, page 50*](#)

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\WebLogic10\samples\server\examples\src\examples\ejb\ejb20\message
```

The file `instructions.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. There are two things that the instructions fail to explicitly state:

1. You must set up your environment by executing this command in the command window:

```
C:\bea\weblogic10\samples\domains\wl_server\setExamplesEnv.cmd
```

2. Next, you must navigate to the MDB source code directory, given above, before executing 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 10.0 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.
- Create the appropriate JMS Destination and connection factory objects inside the TIBCO Enterprise Message Service server using its administration tool.
- Modify the MDB class file by adding JavaDoc annotations, to use the appropriate `JMSConnectionFactory`, `JMSDestination`, and `JNDI Connection Factory` when rebuilding and redeploying.
- 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\weblogic10\samples\domains\wl_server\examples`, modify the `CLASSPATH` environment variable in `setExamplesEnv.cmd` (the examples setup script.).

In the directory `C:\bea\weblogic10\samples\domains\wl_server\bin`, modify the `CLASSPATH` variable in `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:

```
EMS_HOME\jar\tibjms.jar
```

Creating the JMS Destination Object Inside TIBCO EMS

To create the JMS destination object and connection factory objects:

1. Start the TIBCO Enterprise Message Service server by selecting **Start > Programs > TIBCO EMS 5.1 > Start JMS Server** from the Windows Start menu.

2. Start the TIBCO Enterprise Message Service administration tool by selecting **Start > Programs > TIBCO EMS 5.1 > Start EMS Administration Tool** from the Windows Start menu.
3. Enter the following commands:

```
> connect
> create topic quotes
> create jndiname TIBCO.quotes topic quotes
> create factory TIBCO.tcf topic
> create factory SSLTopicConnectionFactory topic
> setprop factory SSLTopicConnectionFactory
    url=ssl://localhost:7243 ssl_verify_host=disable
```

Modifying the MDB Class File to Use TIBCO Enterprise Message Service Objects

You must modify the `MessageTraderBean.java` file in three ways. The `MessageTraderBean.java` file is located in:

```
C:\bea\weblogic10\samples\server\examples\src\examples\ejb\ejb20\message
```

1. Add this import declaration to the top of the file:

```
import weblogic.ejbgen.ForeignJmsProvider;
```

2. Add these tags to the top of the file

```
@ForeignJmsProvider(providerUrl="tibjmsnaming://localhost:7222",
    initialContextFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory",
    connectionFactoryJndiName="TIBCO.tcf")
```

3. Modify the `MessageDriven` tag, so that the `destinationJndiName` property is `TIBCO.quotes`, and the `transactionType` property is `BEAN` driven. For example:

```
@MessageDriven(maxBeansInFreePool = "200",
    destinationType = "javax.jms.Topic",
    initialBeansInFreePool = "20",
    transTimeoutSeconds = "0",
    defaultTransaction = MessageDriven.DefaultTransaction.NOT_SUPPORTED,
    transactionType = MessageDriven.MessageDrivenTransactionType.BEAN,
    durable = Constants.Bool.FALSE,
    ejbName = "messageDriven",
    destinationJndiName = "TIBCO.quotes",
    acknowledgeMode = MessageDriven.AcknowledgeMode.DUPS_OK_ACKNOWLEDGE)
```

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\weblogic10\samples\server\examples\src\examples\ejb\ejb20\message
```

1. Find and replace the following strings in the source file:

Find	Replace With...
<code>weblogic.jms.ConnectionFactory</code>	<code>TopicConnectionFactory</code> OR <code>TIBCO.tcf</code>
<code>weblogic.jndi.WLInitialContextFactory</code>	<code>com.tibco.tibjms.naming.TibjmsInitialContextFactory</code>
<code>t3://localhost:7001</code>	<code>tibjmsnaming://localhost:7222</code>

There should be one occurrence of each of the above strings.

2. Change the line in constructor method to be (this line is right after `getInitialContext()` call):

```
TopicConnectionFactory cf = (TopicConnectionFactory) m_context.lookup("TIBCO.tcf");
```

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://${wls.home}:${wls.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\weblogic10\samples\domains\wl_server>
```

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 10.0 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 44](#), change directory to the example MDB source directory:

```
C:\bea\weblogic10\samples\server\examples\src\examples\ejb\ejb20\message
```

Enter the following commands to rebuild and redeploy the MDB:

```
> ant clean  
> ant build  
> ant deploy
```

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
```

To verify that the MDB is running, open the TIBCO EMS Administration Tool and issue the command:

```
> show topic quotes
```

The consumer count should be 1.

To show that TIBCO Enterprise Message Service is driving the MDB, you can 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 10.0, 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

In the directory `C:\bea\weblogic10\samples\domains\wl_server`, modify the `CLASSPATH` environment variable in `setExamplesEnv.cmd` (the examples setup script) and `startExamplesServer.cmd` (the start script).

To add SSL JAR Files and New JNDI Properties File to the WLS 10.0 CLASSPATH:

1. Open the `setExamplesEnv.cmd` and `startExamplesServer.cmd` files.
2. In each file, add the following jar files to the end of the `CLASSPATH` and save the files.
`EMS_HOME\jar\tibcrypt.jar;EMS_HOME\jar`
3. Run the `setExamplesEnv.cmd` command to set up the client environment.
4. Create a new file named `jndi.properties`, add the following lines and save it to the directory `EMS_HOME\jar`.

```
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 for 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:

```
2008-06-14 10:00:05 Secure Socket Layer is enabled, using openssl <version>
2008-06-14 10:00:05 Accepting connections on ssl://<machineName>:7243.
2008-06-14 10:00:05 Accepting connections on tcp://<machineName>:7222.
```

Modify the Example MDB to Use the SSL Protocol

Follow these steps to modify the example MDB:

1. Change the `providerUrl` property to `tibjmsnaming://localhost:7243`.
2. Change the `connectionFactoryJndiName` property to `SSLTopicConnectionFactory`.

After completing the modifications, the `foreignJmsProvider` tag should look like this:

```
@ForeignJmsProvider(providerUrl="tibjmsnaming://localhost:7243",
    initialContextFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory",
    connectionFactoryJndiName="SSLTopicConnectionFactory")
```

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` or `TIB CO.tcf` to `SSLTopicConnectionFactory`.
2. In `Client.java`, change the port number from 7222 to 7243 in the URL.
3. In `build.xml`, change the port number from 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 commands:

```
> ant clean
> ant build
> ant deploy
```

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 `tibemspd.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:

```
>tibemspd -ssl_debug_trace
```

Then, if you re-start WebLogic Server and re-run the test program, you will see SSL debugging output on the `tibemspd` 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 selecting **Start > Programs > TIBCO EMS 5.1 > Start EMS Administration Tool** from the Windows Start menu.
2. Enter the following commands:

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

Modifying the MDB to Use Transactions

To modify the example MDB to use transactions, you must update the `ForeignJmsProvider` and `MessageDriven` annotation tags in the `MessageTraderBean.java` file, which is located in:

```
C:\bea\weblogic10\samples\server\examples\src\examples\ejb\ejb20\message
```

Follow these steps to modify the example MDB:

1. For the `ForeignJmsProvider` tag, make these changes:
 - a. Change the `providerUrl` property to `tibjmsnaming://localhost:7222`.
 - b. Change the `connectionFactoryJndiName` property to `XATopicConnectionFactory`.
2. For the `MessageDriven` tag, make these changes:
 - a. Change the `defaultTransaction` property to `MessageDriven.DefaultTransaction.NOT_SUPPORTED`.
 - b. Change the `transactionType` property to `MessageDriven.MessageDrivenTransactionType.BEAN`.

After completing the modifications, the `foreignJmsProvider` and `MessageDriven` tags should look like this:

```
@ForeignJmsProvider(providerUrl="tibjmsnaming://localhost:7222",
    initialContextFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory",
    connectionFactoryJndiName="XATopicConnectionFactory")

@MessageDriven(maxBeansInFreePool = "200",
    destinationType = "javax.jms.Topic",
    initialBeansInFreePool = "20",
    transTimeoutSeconds = "0",
    defaultTransaction = MessageDriven.DefaultTransaction.REQUIRED,
    transactionType=MessageDriven.MessageDrivenTransactionType.CONTAINER,
    durable = Constants.Bool.FALSE,
    ejbName = "messageDriven",
    destinationJndiName = "TIBCO.quotes")
```

Modify the Example Client Program to Use Transactions

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

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

Rebuilding and Redeploying the Example MDB

Restart the WebLogic Server Examples Server as described [Rebuilding and Redeploying the Example MDB on page 48](#), so that it picks up the changes to the environment.

Running the Example MDB Client with Transactions

Create a new command prompt window and run the examples setup script, `setExamplesEnv.cmd`, 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.

Chapter 6

Integrating With WebLogic Server 9.2

This chapter describes how to use TIBCO Enterprise Message Service to drive a Message Driven Bean (MDB) inside WebLogic Server 9.2. The examples in this chapter use the preconfigured Examples server and the example MDB that comes bundled as a sample with WebLogic Server 9.2.

The examples in this chapter assume you have installed WebLogic Server 9.2 on a Windows platform in the directory `C:\bea`.

Topics

- [*Running the Example MDB with WebLogic Server, page 54*](#)
- [*Configuring the Example MDB, page 55*](#)
- [*Rebuilding and Redeploying the Example MDB, page 59*](#)
- [*Running the Example MDB Client, page 60*](#)
- [*Modifying this Example to Use SSL Communication, page 61*](#)
- [*Modifying this Example to Use Container Managed Transactions and XA, page 64*](#)

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\WebLogic92\samples\server\examples\src\examples\ejb\ejb20\message
```

The file `instructions.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. There are two things that the instructions fail to explicitly state:

1. You must set up your environment by executing this command in the command window:

```
C:\bea\weblogic92\samples\domains\wl_server\setExamplesEnv.cmd
```

2. Next, you must navigate to the MDB source code directory, given above, before executing 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 9.2 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 JMS Destination and connection factory objects inside the TIBCO Enterprise Message Service server using its administration tool.
- Modify the MDB class file by adding JavaDoc annotations, to use the appropriate `JMSConnectionFactory`, `JMSDestination`, and `JNDI Connection Factory` when rebuilding and redeploying.
- 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 `<weblogic_installation>\samples\domains\wl_server\examples`, modify the `CLASSPATH` environment variable in `setExamplesEnv.cmd` (the examples setup script.).

In the directory `<weblogic_installation>\samples\domains\wl_server\bin`, modify the `CLASSPATH` variable in `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:

`EMS_HOME\lib\tibjms.jar`

Creating the JMS Destination Object Inside TIBCO EMS

To create the example JMS destination and connection factory objects:

1. Start the TIBCO Enterprise Message Service server by selecting **Start > Programs > TIBCO EMS 5.1 > Start JMS Server** from the Windows Start menu.
2. Start the TIBCO Enterprise Message Service administration tool by selecting **Start > Programs > TIBCO EMS 5.1 > Start EMS Administration Tool** from the Windows Start menu.
3. Enter the following commands:

```
> connect
> create topic quotes
> create jndiname TIBCO.quotes topic quotes
> create factory TIBCO.tcf topic
> create factory SSLTopicConnectionFactory topic
> setprop factory SSLTopicConnectionFactory
    url=ssl://localhost:7243 ssl_verify_host=disable
```

Modifying the Sample MDB Class File to Use TIBCO Enterprise Message Service Objects

You must modify the `MessageTraderBean.java` file in three ways. The `MessageTraderBean.java` file is included with WebLogic, and is located in:

`<weblogic_installation>\samples\server\examples\src\examples\ejb\ejb20\message`

1. Add this import declaration to the top of the file:

```
import weblogic.ejbgen.ForeignJmsProvider;
```

2. Add these tags to the top of the file

```
@ForeignJmsProvider(providerUrl="tibjmsnaming://localhost:7222",
    initialContextFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory",
    connectionFactoryJndiName="TIBCO.tcf")
```

3. Modify the `MessageDriven` tag, so that the `destinationJndiName` property is `TIBCO.quotes`, and the `transactionType` property is `BEAN` driven. For example:

```
@MessageDriven(maxBeansInFreePool = "200",
    destinationType = "javax.jms.Topic",
    initialBeansInFreePool = "20",
    transTimeoutSeconds = "0",
    defaultTransaction = MessageDriven.DefaultTransaction.NOT_SUPPORTED,
    transactionType = MessageDriven.MessageDrivenTransactionType.BEAN,
    durable = Constants.Bool.FALSE,
    ejbName = "messageDriven",
    destinationJndiName = "TIBCO.quotes",
    acknowledgeMode = MessageDriven.AcknowledgeMode.DUPS_OK_ACKNOWLEDGE)
```

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:

`<weblogic_installation>\samples\server\examples\src\examples\ejb\ejb20\message`

1. Find and replace the following strings in the source file:

Find	Replace With...
<code>weblogic.jms.ConnectionFactory</code>	<code>TopicConnectionFactory</code> or <code>TIBCO.tcf</code>
<code>weblogic.jndi.WLInitialContextFactory</code>	<code>com.tibco.tibjms.naming.TibjmsInitialContextFactory</code>
<code>t3://localhost:7001</code>	<code>tibjmsnaming://localhost:7222</code>

There should be one occurrence of each of the above strings.

2. Change the line in constructor method to be (this line is right after `getInitialContext()` call):

```
TopicConnectionFactory cf = (TopicConnectionFactory) m_context.lookup("TIBCO.tcf");
```

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://${wls.home}:${wls.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:

```
<weblogic_installation>\samples\domains\wl_server>
```

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 9.2 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 58](#), change directory to the example MDB source directory:

```
<weblogic_installation>\samples\server\examples\src\examples\ejb\ejb20\message
```

Enter the following commands to rebuild and redeploy the MDB:

```
> ant clean  
> ant build  
> ant deploy
```

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
```

To verify that the MDB is running, open the TIBCO EMS Administration Tool and issue the command:

```
> show topic quotes
```

The consumer count should be 1.

To show that TIBCO Enterprise Message Service is driving the MDB, you can 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 9.2, and the client program. This section assumes you have already set up and run the example detailed in the previous sections.

Configure the TIBCO Enterprise Message Service Server for SSL

In *EMS_HOME\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 for 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:

```
2008-06-14 10:00:05 Secure Socket Layer is enabled, using openssl <version>
2008-06-14 10:00:05 Accepting connections on ssl://<machineName>:7243.
2008-06-14 10:00:05 Accepting connections on tcp://<machineName>:7222.
```

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

In the directory *<weblogic_installation>\samples\domains\wl_server*, modify the *CLASSPATH* environment variable in *setExamplesEnv.cmd* (the examples setup script) and *startExamplesServer.cmd* (the start script).

To add SSL JAR Files and New JNDI Properties File to the WLS 9.2 CLASSPATH:

1. Open the *setExamplesEnv.cmd* and *startExamplesServer.cmd* files, located in the directory *<weblogic_installation>\samples\domains\wl_server*.
2. In each file, add the following jar files to the end of the *CLASSPATH* and save the files.
EMS_HOME\lib\tibcrypt.jar;EMS_HOME\lib
3. Run the *setExamplesEnv.cmd* command to set up the client environment.

4. Create a new file named `jndi.properties`, add the following lines and save it to the directory `EMS_HOME\lib`.

```
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.

Modify the Example MDB to Use the SSL Protocol

Follow these steps to modify the example MDB:

1. Change the `providerUrl` property to `tibjmsnaming://localhost:7243`.
2. Change the `connectionFactoryJndiName` property to `SSLTopicConnectionFactory`.

After completing the modifications, the `foreignJmsProvider` tag should look like this:

```
@ForeignJmsProvider(providerUrl="tibjmsnaming://localhost:7243",
    initialContextFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory",
    connectionFactoryJndiName="SSLTopicConnectionFactory")
```

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` or `TIBCO.tcf` to `SSLTopicConnectionFactory`.
2. In `Client.java`, change the port number from `7222` to `7243` in the URL.
3. In `build.xml`, change the port number from `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 commands:

```
> ant clean
> ant build
> ant deploy
```


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 `tibemspd.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:

```
>tibemspd -ssl_debug_trace
```

Then, if you re-start WebLogic Server and re-run the test program, you will see SSL debugging output on the `tibemspd` 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 selecting **Start > Programs > TIBCO EMS 5.1 > Start EMS Administration Tool** from the Windows Start menu.
2. Enter the following commands:

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

Modifying the MDB to Use Transactions

To modify the example MDB to use transactions, you must update the `ForeignJmsProvider` and `MessageDriven` annotation tags in the `MessageTraderBean.java` file, which is located in:

```
C:\bea\weblogic92\samples\server\examples\src\examples\ejb\ejb20\message
```

Follow these steps to modify the example MDB:

1. For the `ForeignJmsProvider` tag, make these changes:
 - a. Change the `providerUrl` property to `tibjmsnaming://localhost:7222`.
 - b. Change the `connectionFactoryJndiName` property to `XATopicConnectionFactory`.
2. For the `MessageDriven` tag, make these changes:
 - a. Change the `defaultTransaction` property to `MessageDriven.DefaultTransaction.NOT_SUPPORTED`.
 - b. Change the `transactionType` property to `MessageDriven.MessageDrivenTransactionType.BEAN`.

After completing the modifications, the `foreignJmsProvider` and `MessageDriven` tags should look like this:

```
@ForeignJmsProvider(providerUrl="tibjmsnaming://localhost:7222",
    initialContextFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory",
    connectionFactoryJndiName="XATopicConnectionFactory")

@MessageDriven(maxBeansInFreePool = "200",
    destinationType = "javax.jms.Topic",
    initialBeansInFreePool = "20",
    transTimeoutSeconds = "0",
    defaultTransaction = MessageDriven.DefaultTransaction.REQUIRED,
    transactionType=MessageDriven.MessageDrivenTransactionType.CONTAINER,
    durable = Constants.Bool.FALSE,
    ejbName = "messageDriven",
    destinationJndiName = "TIBCO.quotes")
```

Modify the Example Client Program to Use Transactions

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

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

Rebuilding and Redeploying the Example MDB

Restart the WebLogic Server Examples Server as described [Rebuilding and Redeploying the Example MDB on page 62](#), so that it picks up the changes to the environment.

Running the Example MDB Client with Transactions

Create a new command prompt window and run the examples setup script, `setExamplesEnv.cmd`, 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.

Chapter 7

Integrating With IBM WebSphere Application Server Version 6.1

This chapter describes integrating TIBCO Enterprise Message Service with IBM WebSphere Application Server Version 6.1. 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 68*](#)
- [*Create TIBCO Enterprise Message Service Administered Objects, page 69*](#)
- [*Configure WebSphere to Use EMS as the JMS Provider, page 70*](#)
- [*Modify the Samples to Use SSL Communications, page 77*](#)

Overview of Integrating With IBM WebSphere

This chapter is divided into the following sections:

- [Create TIBCO Enterprise Message Service Administered Objects](#) — describes the steps needed to create connection factory, topic, and queue objects in the EMS server, which are then used when configuring the WebSphere application server.
- [Configure WebSphere to Use EMS as the JMS Provider](#) — demonstrates how to reconfigure and run MDBs using TIBCO Enterprise Message Service as the JMS provider within WebSphere.

TIBCO Enterprise Message Service is simply added to WebSphere application server and client container as a JMS provider. Then, the MDB is reconfigured for the new JMS provider (TIBCO EMS) resources and re-deployed.

- [Modify the Samples to Use SSL Communications](#) — details how to modify the program 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 V6.1, trial (plus embedded messaging) on a Windows platform, and have an MDB configured with WebSphere Application Server. The instructions also assume that TIBCO Enterprise Message Service and WebSphere Application Server are both running on the same machine, and that you have complied the sample Java clients included with your EMS installation.

Create TIBCO Enterprise Message Service Administered Objects

This section creates the following administered objects in the `tibemsd`:

- The topic connection factory, `sample.TCF`.
- The queue connection factory, `sample.QCF`.
- Two topics, `sample.weather` and `sample.*`.
- One queue, `sample.Q1`.

In this section, you also associate the topic `sample.*` with the external JNDI name `sample.listen`.

To create the administered objects:

1. Start the TIBCO Enterprise Message Service server.
2. Start the admin tool and connect to the EMS server using the `connect` command:

```
> connect -server server_name
```

where *server_name* is the server you wish to connect to.

3. Enter the following commands to create the connection factories and destinations:

```
> create factory sample.TCF topic
> create factory sample.QCF queue
> create topic sample.*
> create jndiname sample.listen topic sample.*
> create topic sample.weather
> create queue sample.Q1
```

The `create jndiname` command associates the destination with an external JNDI name. This command is needed if you want to use an external JNDI name that is different from the destination name. In the example above, the topic `sample.*` is assigned the external JNDI name `sample.listen`.

Configure WebSphere to Use EMS as the JMS Provider

This section describes the steps required to configure WebSphere Application Server 6.1 to use TIBCO Enterprise Message Service as its JMS provider. The configuration process is described in the following steps:

- [Add TIBCO Enterprise Message Service as a JMS Provider, page 70](#)
- [Configure JNDI Bindings for the Connection Factories, page 71](#)
- [Configure JNDI Bindings for the Destinations, page 72](#)
- [Create new Listener Ports, page 73](#)
- [Install the MDB to Use the Topic and Queue Listeners, page 74](#)

After the application server is configured, the section [Run the Sample Application Clients on page 75](#) gives steps for running the sample Java clients to test the configuration.

This section assumes that an MDB is already configured in the WebSphere Application Server, and that the required administered objects have been created in the EMS server, as described above in [Create TIBCO Enterprise Message Service Administered Objects on page 69](#).

Add TIBCO Enterprise Message Service as a JMS Provider

This section configures WebSphere to uses TIBCO Enterprise Message Service as a JMS provider.

1. Start the WebSphere application server.
2. Start the WebSphere Administrative Console.
3. In the WebSphere navigation pane, choose **Resources > JMS > JMS providers**.
4. In the content pane, click the **New** button.
5. Enter the following values for the required properties:

Name	TIBCO
Description	TIBCO Enterprise Message Service
Classpath	EMS_HOME\jar\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 **Apply** button.

For more information about this task, review the WebSphere Application Server documentation for defining a generic JMS provider.

Configure JNDI Bindings for the Connection Factories

This section describes how to create topic and queue connection factories in the WebSphere server.

Create a Topic
Connection
Factory

1. In the WebSphere navigation pane, choose **Resources > JMS > Topic connection factories**.
2. In the content pane, click the **New** button.
3. Choose **TIBCO** as the resource provider, and click **OK**.
4. Enter the following values for the required properties:

Name	<code>TIBCOConnectionFactory1</code>
Type	<code>TOPIC</code>
JNDI Name	<code>jms/ConnectionFactory1</code>
Description	<code>Sample Topic ConnectionFactory</code>
External JNDI Name	<code>sample.TCF</code>

5. Click the **OK** button.

Create a Queue
Connection
Factory

6. In the WebSphere navigation pane, choose **Resources > JMS > Queue connection factories**.
7. Click the **New** button.
8. Choose **TIBCO** as the resource provider, and click **OK**.
9. Enter the following values for the required properties:

Name	<code>TIBCOConnectionFactory2</code>
Type	<code>QUEUE</code>
JNDI Name	<code>jms/ConnectionFactory2</code>
Description	<code>Sample Queue ConnectionFactory</code>
External JNDI Name	<code>sample.QCF</code>

10. Click the **OK** button.
11. Click the **Save** button.

Configure JNDI Bindings for the Destinations

This section describes how to bind the EMS topics `sample.listen` and `sample.weather` and queue `sample.Q1` in WebSphere.

- Add Topics
1. In the WebSphere navigation pane, choose **Resources > JMS > Topics**.
 2. In the content pane, click the **New** button.
 3. Choose **TIBCO**.
 4. Enter the following values for the required properties:

Name	<code>Listen</code>
Type	<code>TOPIC</code>
JNDI Name	<code>jms/listen</code>
Description	<code>Sample Listen Topic</code>
External JNDI Name	<code>sample.listen</code>

5. Click the **OK** button.
6. Repeat the previous steps to create a topic with these properties:

Name	<code>Weather</code>
Type	<code>TOPIC</code>
JNDI Name	<code>jms/weather</code>
Description	<code>Sample Weather Topic</code>
External JNDI Name	<code>sample.weather</code>

- Add the Queue
7. In the WebSphere navigation pane, choose **Resources > JMS > Queues**.
 8. In the content pane, click the **New** button.
 9. Choose **TIBCO**.
 10. Enter the following values for the required properties:

Name	<code>Q1</code>
Type	<code>QUEUE</code>
JNDI Name	<code>jms/Q1</code>
Description	<code>Sample Q1 Queue</code>
External JNDI Name	<code>sample.Q1</code>

11. Click the **OK** button.
12. Click the **Save** button.

Create new Listener Ports

This section creates the WebSphere listener ports that connect to the `tibemsd`. You create two listener ports:

- `TIBCOPubSubListenerPort` for the configured topics.
- `TIBCOPTtoPLListenerPort` for the configured queue.

To create the listener ports:

- Create a Listener Port for Topics
1. In the WebSphere navigation pane, choose **Servers > Application servers**, and choose the name of the application server.
 2. In the content pane, scroll down to the Communications section and click **Message Listener Service**.
 3. In the content pane, select **Listener Ports**.
 4. Click the **New** button.
 5. Enter the following values for the required listener port properties:

Name	<code>TIBCOPubSubListenerPort</code>
Initial State	<code>Started</code>
Description	<code>Listener Port for TIBCO PubSub</code>
ConnectionFactory JNDI Name	<code>jms/ConnectionFactory1</code>
Destination JNDI Name	<code>jms/listen</code>

6. Click the **OK** button.

- Create a Listener Port for a Queue
7. Repeat the previous steps to create another listener port with the following property values:

Name	<code>TIBCOPTtoPLListenerPort</code>
Initial State	<code>Started</code>
Description	<code>Listener Port for TIBCO Point to Point</code>
ConnectionFactory JNDI Name	<code>jms/ConnectionFactory2</code>
Destination JNDI Name	<code>jms/Q1</code>

8. Click the **OK** button.
9. Click the **Save** button.

For more information about this task, review the WebSphere Application Server documentation for adding a new listener port.

Install the MDB to Use the Topic and Queue Listeners

This section describes the steps needed to install your MDB for the topic and queue listeners created above, in [Create new Listener Ports on page 73](#).

Install the Topic MDB

1. In the navigation pane, choose **Applications > Enterprise Applications**.
2. In the content pane, click the **Install** button.
3. Specify the location of [your message driven bean](#) .ear file.
4. Click the **Next** button.
5. Change the **Application name** property to `test_topic`.

Note that you do not need to change any of the other default installation options.

6. Click the **Next** button.
7. Click the **Next** button to accept the default module to application server mapping.
Note that this mapping can be modified only if you have more than one WebSphere application server configured.
8. Change the value of the Listener Port Name to `TIBCOPubSubListenerPort`.
`TIBCOPubSubListenerPort` is the listener port created above in [step 5 of Create new Listener Ports](#).
9. Click **Next**.
10. Click **Finish** to complete the installation.
11. Save the changes to the Master Configuration by clicking the **Save** button.

Install the Queue MDB

12. In the navigation pane, you should still be in **Applications > Enterprise Applications**.
13. In the content pane, click the **Install** button.
14. Specify the location of [your message driven bean](#) .ear file.
15. Click the **Next** button.
16. Change the Application name to `test_queue`.

Note that you do not need to change any of the other default installation options.

17. Click the **Next** button.
18. Click the **Next** button to accept the default module to application server mapping.

Note that this mapping can be modified only if you have more than one WebSphere application server configured.

19. Change the value of the Listener Port Name to `TIBCOPTOPLISTENERPORT`.

`TIBCOPTOPLISTENERPORT` is the listener port created above in [step 7 of Create new Listener Ports](#).

20. Click **Next**.

21. Click **Finish** to complete the installation.

22. Save the changes to the Master Configuration by clicking the **Save** button.

Restart the
Application
Server

23. Stop and restart the application server to have your changes take effect.

24. 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.

For more information about this task, review the WebSphere Application Server documentation for configuring deployment attributes for a message-driven bean.

Run the Sample Application Clients

In this section, you run the sample Java clients included with your TIBCO Enterprise Message Service installation. If this is the first time you have used these sample clients, you will need to compile them before following the steps described here.

1. From the `EMS_HOME\samples\java` directory, type the command:

```
> java tibjmsTopicPublisher -topic sample.listen "hello world"
```

2. From the `EMS_HOME\samples\java` directory, type the command:

```
> java tibjmsTopicPublisher -topic sample.weather "hello world  
2"
```

3. From the `EMS_HOME\samples\java` directory, type the command:

```
> java tibjmsQueueSender -queue sample.Q1 "hello world 3"
```

4. Open the WebSphere application server log, located in `<installation_directory>\logs\<your server>\SystemOut.log`, and verify that the

WebSphere MDB consumed the sent messages. The log should contain messages sent to the following destinations:

Destination	Message
sample.listen	hello world
sample.weather	hello world 2
sample.Q1	hello world 3

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, as detailed in the previous sections.

Enable SSL in the TIBCO Enterprise Message Service Server

This section describes the steps that enable SSL in the `tibemspd`.

1. In `EMS_HOME\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.

2. Save the file and stop the TIBCO Enterprise Message Service server.
3. Start the TIBCO EMS server with the `-ssl_debug_trace` option:

```
> tibemspd -ssl_debug_trace
```

See [Starting the EMS Server on page 96](#) of the *TIBCO Enterprise Message Service User's Guide* for more information.

When the server restarts you should see messages like the following in the console window confirming SSL is enabled:

```
2008-06-11 13:48:34 Secure Socket Layer is enabled.
2008-06-11 13:48:34 Accepting connections on ssl://localhost:7243.
2008-06-11 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

`SSLTopicConnectionFactory` and `SSLQueueConnectionFactory` 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.SSLTCF jndiname SSLTopicConnectionFactory
> create jndiname sample.SSLQCF jndiname SSLQueueConnectionFactory
```

Add Additional SSL JNDI Properties to WebSphere

Locate the `jndi.properties` file in the WebSphere installation directory:

```
<installation_directory>\properties\jndi.properties
```

If it does not already exist, create a text file called `jndi.properties` in the directory.

Edit the `jndi.properties` file to add the following lines:

```
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_enable_verify_host=false
com.tibco.tibjms.naming.ssl_vendor=j2se-default
```

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).

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 > JMS > JMS providers** and choose **TIBCO** in the content pane.
2. Add the following lines to the **Class path** property value:

```
EMS_HOME\jar\tibjms.jar
EMS_HOME\jar\TIBCrypt.jar
```

3. Change the port number of the **External Provider URL** property from `tibjmsnaming://localhost:7222` to:
`tibjmsnaming://localhost:7243`

Configure the Topic Connection Factory

4. Click the **OK** button.
5. In the WebSphere navigation pane, choose **Resources > JMS > Topic connection factories**.
6. Select **TIBCOConnectionFactory1**.
This is the connection factory created in [Create a Topic Connection Factory on page 71](#).
7. For the External JNDI Name property value, change the name of the factory from `sample.TCF` to `sample.SSLTCF`.
8. Click the **OK** button.

Configure the Queue Connection Factory

9. In the WebSphere navigation pane, choose **Resources > JMS > Queue connection factories**.
10. Select **TIBCOConnectionFactory2**.
This is the connection factory created in [Create a Queue Connection Factory on page 71](#).
11. For the External JNDI Name property value, change the name of the factory from `sample.QCF` to `sample.SSLQCF`.
12. Click the **OK** button.
13. Click **Save**.
14. Stop and restart the WebSphere application server.

Run the Samples Application Clients

Run the samples application client again:

1. From the `EMS_HOME\samples\java` directory, type the command:

```
> java tibjmsTopicPublisher -server ssl://localhost:7243 -topic sample.weather "hello world"
```
2. From the `EMS_HOME\samples\java` directory, type the command:

```
> java tibjmsQueueSender -server ssl://localhost:7243 -queue sample.Q1 "hello world 2"
```
3. Open the WebSphere application server log, located in `<installation_directory>\logs\<your server>\SystemOut.log`, and verify that the WebSphere MDB consumed the sent messages.

Chapter 8

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 82*](#)
- [*Get the sample MDB running with the WebSphere Embedded JMS Provider, page 83*](#)
- [*Get the Sample MDB running with TIBCO Enterprise Message Service, page 86*](#)
- [*Modify the Samples to Use SSL Communications, page 97*](#)

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	<code>Sample/JMS/TCF</code>
Destination JNDI name	<code>Sample/JMS/listen</code>

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	<code>TIBCO</code>
Description	<code>TIBCO Enterprise Message Service</code>
Classpath	<code>EMS_HOME\jar\tibjms.jar</code>
External Initial Context Factory	<code>com.tibco.tibjms.naming.TibjmsInitialContextFactory</code>
External Provider URL	<code>tibjmsnaming://localhost:7222</code>

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	<code>TIBCOConnectionFactory1</code>
Type	<code>TOPIC</code>
JNDI Name	<code>jms/ConnectionFactory1</code>
Description	<code>Sample Topic ConnectionFactory</code>
External JNDI Name	<code>tibjmsnaming://localhost/sample.TCF</code>

6. Click the **OK** button.
7. Click the **New** button.

- Enter the following values for the given properties:

Name	<code>TIBCOConnectionFactory</code>
Type	<code>QUEUE</code>
JNDI Name	<code>jms/ConnectionFactory</code>
Description	<code>Sample Queue ConnectionFactory</code>
External JNDI Name	<code>tibjmsnaming://localhost/sample.QCF</code>

- 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	<code>Listen</code>
Type	<code>TOPIC</code>
JNDI Name	<code>jms/listen</code>
Description	<code>Sample Listen Topic</code>
External JNDI Name	<code>tibjmsnaming://localhost/sample.listen</code>

- 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.

- Enter the following values for the given listener port properties:

Name	<code>TIBCOPTOPListenerPort</code>
Initial State	<code>Started</code>
Description	<code>Listener Port for TIBCO Point to Point</code>
ConnectionFactory JNDI Name	<code>jms/ConnectionFactory</code>
Destination JNDI Name	<code>jms/Q1</code>

- Click the **OK** button.
- Repeat the previous steps to create another listener port with the following property values:

Name	<code>TIBCOPubSubListenerPort</code>
Initial State	<code>Started</code>
Description	<code>Listener Port for TIBCO PubSub</code>
ConnectionFactory JNDI Name	<code>jms/ConnectionFactory1</code>
Destination JNDI Name	<code>jms/listen</code>

- Click the **Save** button on the task bar of the Administrative Console (and **Save** again to confirm).
- Stop and restart the application server to have the changes take effect.
- 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

- Start the WebSphere Application Assembly Tool.
- 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 `TIBCOPTtoPLListenerPort`.
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	<code>EMS_HOME\jar\tibjms.jar</code>
ContextFactory Class	<code>com.tibco.tibjms.naming.TibjmsInitialContextFactory</code>
Provider URL	<code>tibjmsnaming://localhost:7222</code>

6. Click the **OK** button.
7. Repeat the previous three steps for `PtoPSSampleClient.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	<code>TIBCOConnectionFactory1</code>
Description	<code>Sample Topic Connection Factory</code>
JNDI Name	<code>jms/ConnectionFactory1</code>
External JNDI Name	<code>tibjmsnaming://localhost/sample.TCF</code>
Connection Type	<code>TOPIC</code>

4. Click the **OK** button.
5. Repeat the previous three steps for `PtoPSampleClient.jar` using the following values:

Name	<code>TIBCOConnectionFactory</code>
Description	<code>Sample Queue Connection Factory</code>
JNDI Name	<code>jms/ConnectionFactory</code>
External JNDI Name	<code>tibjmsnaming://localhost/sample.QCF</code>
Connection Type	<code>QUEUE</code>

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 86](#) 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 line to the **Classpath** property value:
`EMS_HOME\jar\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:
`EMS_HOME\jar\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**:
`EMS_HOME\jar\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 9

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 104*](#)
- [*Run the MDB Sample with TIBCO EMS, page 105*](#)
- [*Run the MDB Sample with TIBCO EMS using SSL, page 107*](#)

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 *EMS_HOME*\bin is in the `PATH`.
 2. Start the application server.
 3. Change directory to *EMS_HOME*\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 *EMS_HOME*\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 *EMS_HOME*\samples\ejb\mdb\simple\src, and run these commands:

```
asant clean
asant undeploy
```

6. Remove the directory:
EMS_HOME\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 `EMS_HOME\jar\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 `EMS_HOME\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 *EMS_HOME*\bin. Modify *appclient.bat* by adding *EMS_HOME*\jar\tibjms.jar to *JVM_CLASSPATH*.
9. Change directory to *EMS_HOME*\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
 EMS_HOME\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 *EMS_HOME*\samples\ejb\mdb\simple\src, and run these commands:

```
asant clean
asant undeploy
```

12. Remove the directory:

```
EMS_HOME\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 filename, and click the Save button:

```
EMS_HOME\jar\tibjms.jar
EMS_HOME\jar\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 *EMS_HOME*\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 *EMS_HOME*\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* and *tibcrypt.jar* to *JVM_CLASSPATH* in *appclient.bat*.

Run 12. Change directory to
EMS_HOME\domains\domain1\server1\applications\j2ee-apps\mdb-s
 imple_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 *EMS_HOME*\samples\ejb\mdb\Simple\src, and run these commands:

```
asant clean
asant undeploy
```

14. Remove the directory:

EMS_HOME\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](#)

customer support [xvii](#)

I

integrating with third-party application servers [10](#)

S

support, contacting [xvii](#)

T

technical support [xvii](#)

third-party application servers [10](#)

TIBCO_HOME [xiv](#)

transactions, container-managed [12, 16](#)