



BEA

WebLogic Server

WebLogic Tuxedo Connector Administration Guide

BEA WebLogic Tuxedo Connector Release 1.0
Document Date: April 30, 2003

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Collaborate, BEA WebLogic Commerce Server, BEA WebLogic E-Business Platform, BEA WebLogic Enterprise, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Server, E-Business Control Center, How Business Becomes E-Business, Liquid Data, Operating System for the Internet, and Portal FrameWork are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

WebLogic Tuxedo Connector Administration Guide

Part Number	Document Date	Software Version
N/A	June 29, 2001	BEA WebLogic Tuxedo Connector 1.0

Contents

About This Document

Audience.....	x
e-docs Web Site.....	x
How to Print the Document.....	x
Related Information.....	xi
Contact Us!	xi
Documentation Conventions	xii

1. Introduction to WebLogic Tuxedo Connector

Retirement of WebLogic Tuxedo Connector 1.0	1-2
WebLogic Tuxedo Connector Overview.....	1-2
Key Functionality and Administrative Features.....	1-4
Limitations	1-5
How WebLogic Tuxedo Connector Differs from Jolt.....	1-5
WebLogic Tuxedo Connector Distribution Files	1-6
Documentation	1-6
Platform Support	1-6
Licensing	1-7

2. Installing and Configuring WebLogic Tuxedo Connector

Summary of Environment Changes and Considerations.....	2-1
Tuxedo Changes	2-2
WebLogic Server Changes.....	2-2
WebLogic Server ThreadPoolSize	2-2
Installing the WebLogic Tuxedo Connector	2-3
Configuring WebLogic Tuxedo Connector for Your Applications	2-7
Create a Configuration File	2-8

Configuration File Components	2-8
Validate the XML file	2-10
Set the WebLogic Server Domain Environment	2-10
Update the Start Server Script	2-11
Create a StartUp Class	2-12
Create a Shutdown Class	2-13
Deploy EJBs	2-13
Restart the Application Server	2-14
Verify Installation	2-14
Uninstalling the WebLogic Tuxedo Connector	2-15
Removing WebLogic Tuxedo Connector from WebLogic Server ..	2-15
Removing WebLogic Tuxedo Connector from the System	2-15

3. Configuring BDMCONFIG

Establishing Connectivity	3-2
Connecting at Boot Time (ON_STARTUP)	3-2
RetryInterval	3-2
MaxRetries	3-2
Connecting on Request (ON_DEMAND)	3-3
Accepting Incoming Connections (INCOMING_ONLY)	3-3
LOCAL	3-3
Dynamic Status	3-4
Configuring Failover and Failback	3-4
Domain Failover and Failback	3-5
Establishing Security	3-5
Domains Passwords	3-5
Generating Encrypted Passwords	3-6
Usage	3-6
Examples	3-6
Access Control Lists	3-7
Security Requirements for servers	3-8
Security Requirements for clients	3-8
Establishing an ACL Policy	3-8
Add Users to TpUserFile	3-8
Modify EJBs for ACL	3-9

Modify Tuxedo Environment for ACL	3–9
Modify WebLogic Tuxedo Connector Environment for ACL	3–9
Example ACL Policy	3–10

4. Configuring tBridge

Overview of the tBridge	4–1
WebLogic Tuxedo Configuration XML File Configuration for tBridge	4–2
Starting the tBridge	4–2
Error Logging	4–2
tBridge Connectivity	4–3
Example Connection Type Configurations	4–3
Example JmsQ2TuxQ Configuration	4–3
Example TuxQ2JmsQ Configuration	4–5
Example JmsQ2TuxS Configuration	4–6
Priority Mapping	4–8
Error Queues	4–12
wlsServerErrorDestination	4–12
Unsupported Message Types	4–12
tuxErrorQueue	4–12
Limitations	4–12

5. Using FML with WebLogic Tuxedo Connector

Overview of FML	5–1
The WebLogic Tuxedo Connector FML API	5–2
FML Field Table Administration	5–2
tBridge XML/FML Translation	5–4
FLAT	5–5
NO	5–5
FML Considerations	5–6

6. Connecting WebLogic Process Integrator and Tuxedo Applications

Synchronous WebLogic Process Integrator-to-Tuxedo Connectivity	6–2
Defining Business Operations	6–2
Invoking an eLink Adapter	6–2

Define Exception handlers	6-2
Synchronous Non-Blocking WebLogic Process Integrator-to-Tuxedo Connectivity	6-3
Asynchronous WebLogic Process Integrator-to-Tuxedo Connectivity.....	6-3
Asynchronous Tuxedo /Q-to-WebLogic Process Integrator Connectivity.....	6-4
Bi-directional Asynchronous Tuxedo-to-WebLogic Process Integrator Connectivity	6-5
7. Troubleshooting The WebLogic Tuxedo Connector	
Monitoring the WebLogic Tuxedo Connector	7-1
Setting Trace Levels.....	7-1
Console Settings	7-2
Frequently Asked Questions.....	7-3
Cannot Find Configuration File.....	7-3
EJB Deployment Message.....	7-4
Connection Problems.....	7-4
8. The WebLogic Tuxedo Connector XML Configuration File	
Creating an XML Configuration File	8-1
Sample XML Configuration File.....	8-1
Validate the XML file	8-3
Element Hierarchy Diagram.....	8-3
9. The wtc_config.dtd	
wtc_config.dtd	9-1
10. Elements and Attributes of the wtc_config.dtd	
WTC_CONFIG	10-1
BDMCONFIG	10-2
T_DM_LOCAL_TDOMAIN.....	10-2
WlsClusterName	10-2
AccessPointId.....	10-2
Type.....	10-3
Security.....	10-3
ConnectionPolicy	10-3
RetryInterval.....	10-4

MaxRetries	10-4
ConnPrincipalName	10-5
NWAddr	10-5
CmpLimit	10-5
MinEncryptBits	10-6
MaxEncryptBits	10-6
Interoperate	10-6
BlockTime.....	10-6
T_DM_REMOTE_TDOMAIN	10-6
LocalAccessPoint.....	10-7
AclPolicy.....	10-7
CredentialPolicy.....	10-7
TpUsrFile	10-8
T_DM_EXPORT	10-8
RemoteName.....	10-8
EJBName	10-9
T_DM_IMPORT.....	10-9
TranTime.....	10-9
T_DM_PASSWORD	10-10
LocalPassword	10-10
RemotePassword.....	10-11
T_DM_RESOURCES.....	10-11
FieldTables.....	10-11
FldTblClass	10-11
AppPassword	10-12
tBridge	10-12
direction	10-12
fromto.....	10-12
redirect	10-12
source	10-13
target.....	10-13
AccessPoint.....	10-13
Qspace	10-13
Name	10-13
ReplyQ	10-13

metadataFile	10-13
translateFML	10-14
transactional	10-14
timeout.....	10-14
idleTime	10-15
retries	10-15
retryDelay	10-15
wlsErrorDestination	10-15
tuxErrorQueue	10-16
defaultRelativeBirthtime	10-16
defaultRelativeExpiration.....	10-16
expirationAdjustment	10-16
priorityMapping	10-17
JmstoTux	10-17
TuxtoJms	10-17
pMap.....	10-18
range	10-18
value	10-18
deliveryModeOverride	10-18
defaultreplyDeliveryMode	10-18
userID	10-19
allowNonStandardTypes	10-19
jndiFactory	10-19
jmsFactory	10-19
tuxFactory.....	10-20

About This Document

This document introduces the BEA WebLogic Tuxedo Connector™ application development environment. It describes how to establish a development environment and how to package applications for deployment.

The document is organized as follows:

- [Chapter 1, “Introduction to WebLogic Tuxedo Connector,”](#) is an overview of the WebLogic Tuxedo Connector.
- [Chapter 2, “Installing and Configuring WebLogic Tuxedo Connector,”](#) describes how to install and configure the WebLogic Tuxedo Connector.
- [Chapter 3, “Configuring BDMCONFIG,”](#) provides configuration information about BDMCONFIG.
- [Chapter 4, “Configuring tBridge,”](#) provides information on tBridge functionality and configuration.
- [Chapter 5, “Using FML with WebLogic Tuxedo Connector,”](#) discusses the Field Manipulation Language (FML) and describes how the WebLogic Tuxedo Connector uses FML.
- [Chapter 6, “Connecting WebLogic Process Integrator and Tuxedo Applications,”](#) provides the necessary infrastructure for WebLogic Process Integrator users to work Tuxedo applications into their business workflows.
- [Chapter 7, “Troubleshooting The WebLogic Tuxedo Connector,”](#) provides WebLogic Tuxedo Connector troubleshooting information.
- [Chapter 8, “The WebLogic Tuxedo Connector XML Configuration File,”](#) describes how to create a WebLogic Tuxedo Connector XML configuration file and provides a hierarchy diagram of elements.
- [Chapter 9, “The wtc_config.dtd,”](#) provides the structure of elements and attributes used to create a WebLogic Tuxedo Connector XML configuration file.

-
- Chapter 10, “Elements and Attributes of the `wtc_config.dtd`,” provides reference information on the elements and attributes contained in the `wtc_config.dtd`.

Audience

This document is intended for system administrators and application developers who are interested in building distributed Java applications that interoperate between WebLogic Server and Tuxedo environments. It assumes a familiarity with the WebLogic Server, Tuxedo, XML and Java programming.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at <http://www.adobe.com>.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server and Tuxedo.

For more information about Java and Java CORBA applications, refer to the following sources:

- The OMG Web Site at <http://www.omg.org/>
- The Sun Microsystems, Inc. Java site at <http://java.sun.com/>

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at <http://www.bea.com>. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
<i>italics</i>	Emphasis and book titles.
monospace text	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard. <i>Examples:</i> <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>monospace italic text</i>	Variables in code. <i>Example:</i> <pre>String CustomerName;</pre>
UPPERCASE TEXT	Device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 BEA_HOME OR</pre>
{ }	A set of choices in a syntax line.
[]	Optional items in a syntax line. <i>Example:</i> <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>

Convention	Usage
	Separates mutually exclusive choices in a syntax line. <i>Example:</i> <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
...	Indicates one of the following in a command line: <ul style="list-style-type: none">■ An argument can be repeated several times in the command line.■ The statement omits additional optional arguments.■ You can enter additional parameters, values, or other information
.	Indicates the omission of items from a code example or from a syntax line.



1 Introduction to WebLogic Tuxedo Connector

The following sections summarize the concepts and functionality of WebLogic Tuxedo Connector:

- [Retirement of WebLogic Tuxedo Connector 1.0](#)
- [WebLogic Tuxedo Connector Overview](#)
- [Key Functionality and Administrative Features](#)
- [How WebLogic Tuxedo Connector Differs from Jolt](#)
- [WebLogic Tuxedo Connector Distribution Files](#)
- [Documentation](#)
- [Platform Support](#)
- [Licensing](#)

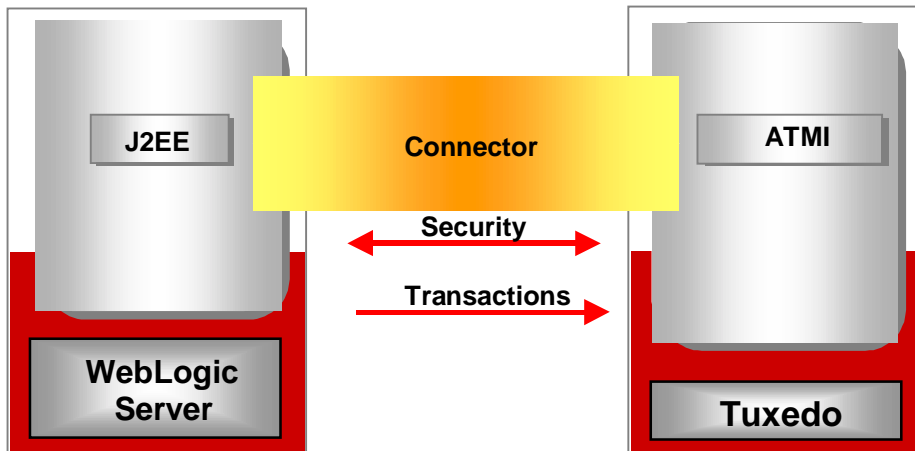
Retirement of WebLogic Tuxedo Connector 1.0

WebLogic Tuxedo Connector 1.0 is retired as of April 30, 2003. Although WebLogic Tuxedo Connector is no longer a standalone product, BEA has continued to develop WebLogic Tuxedo Connector as a service within WebLogic Server. WebLogic Server releases 6.1 and higher provide WebLogic Tuxedo Connector services with many feature enhancements. For more information see:

- [WebLogic Tuxedo Connector for WebLogic 6.1](#)
- [WebLogic Tuxedo Connector for WebLogic 7.0](#)
- [WebLogic Tuxedo Connector for WebLogic 8.1](#)

WebLogic Tuxedo Connector Overview

The WebLogic Tuxedo Connector provides interoperability between WebLogic Server applications and Tuxedo services. The connector uses a collection of jar and XML files that allow WebLogic Server clients to invoke Tuxedo services and Tuxedo clients to invoke WebLogic Server Enterprise Java Beans (EJBs) in response to a service request.



Key Functionality and Administrative Features

The WebLogic Tuxedo Connector Release 1.0 enables you to develop and support applications interoperating WebLogic Server and Tuxedo by using a Java Application-to-Transaction Monitor Interface (JATMI) similar to the Tuxedo ATMI. The WebLogic Tuxedo Connector tBridge functionality provides Tuxedo /Q and JMS advanced messaging services.

The WebLogic Tuxedo Connector provides the following bi-directional interoperability:

- Ability to call WebLogic Server applications from Tuxedo applications and vice versa or to call EJBs
- Ability to integrate WebLogic Server applications into existing Tuxedo environments
- Ability to use WebLogic Process Integrator to manage workflow across Tuxedo ATMI services, including eLink 1.2 adapters
- Ability to define multiple connections between WebLogic Server and Tuxedo

The WebLogic Tuxedo Connector includes the following key administration features:

- Simple implementation. The WebLogic Tuxedo Connector does not require modification of existing Tuxedo application code.
 - Existing Tuxedo clients call WebLogic Server EJBs through the WebLogic Tuxedo Connector
 - New or modified WebLogic Server clients call Tuxedo services through WebLogic Tuxedo Connector.
- Administered through an XML configuration file.
- Out-bound transaction support (transactions originating in WebLogic Server).
- Bi-directional security propagation, including domain and ACL security.
- Domain-level failover and fallback.

- Advanced messaging services provided by Tuxedo /Q and JMS.
- Interoperability with mainframes and other legacy applications using eLink.

Limitations

This release of the WebLogic Tuxedo Connector has the following limitations:

- Does not support dynamic configuration changes to the WebLogic Tuxedo Connector gateway
- Does not support in-bound transactions (a transaction originating in Tuxedo)
- Does not support Conversations
- Does not support Views
- Does not support double-byte character sets or international character sets. These features are dependent on future releases of Tuxedo.

How WebLogic Tuxedo Connector Differs from Jolt

The WebLogic Tuxedo Connector is not a replacement for Jolt. WebLogic Tuxedo Connector differs from Jolt in the following ways:

- WebLogic Tuxedo Connector offers a similar but different API than Jolt.
- Jolt enables the development of generic Java clients and other Web server applications that the WebLogic Tuxedo Connector does not.
- Jolt does not provide a mechanism for an integrated WebLogic Server-Tuxedo transaction.

Users should use Jolt as a solution instead of the WebLogic Tuxedo Connector when a generic Java client or other Web server application is required and WebLogic Server is not part of the solution.

WebLogic Tuxedo Connector Distribution Files

The WebLogic Tuxedo Connector distribution contains the following files:

- The WebLogic Tuxedo Connector jar file named *jatmi.jar*
- A *doc* directory containing the JATMI Javadoc
- A *wtc_config.dtd* data type definition file
- Sample applications: *simpapp*, *qsample*, and *simpserv*

Documentation

You can download the WebLogic Tuxedo Connector from the following locations:

- On the BEA corporate Web Site. From the BEA Home page at <http://www.bea.com>, click on Product Documentation. Select the BEA Tuxedo 8.0 documentation set.
- Go directly to the WebLogic Tuxedo Connector “e-docs” product documentation page at <http://e-docs.bea.com/wtc/wct10/index.html>

Platform Support

See our *Platforms Support* page at <http://www.weblogic.com/platforms/index.html> for the most accurate and current information regarding platform support.

Licensing

This section provides licensing information for the WebLogic Tuxedo Connector:

- There is no license requirement for using the connector without encryption.
- An appropriate Tuxedo LLE license and an appropriate WebLogic Server SSL license is required to use encryption.

2 Installing and Configuring WebLogic Tuxedo Connector

The following sections describe how to install and configure the WebLogic Tuxedo Connector. Installation instructions are provided for Windows NT and UNIX platforms.

- [Summary of Environment Changes and Considerations](#)
- [Installing the WebLogic Tuxedo Connector](#)
- [Configuring WebLogic Tuxedo Connector for Your Applications](#)
- [Uninstalling the WebLogic Tuxedo Connector](#)

Summary of Environment Changes and Considerations

This section provides an overview of the changes you must make to the Tuxedo and WebLogic Server environments before you can start using the WebLogic Tuxedo Connector. These changes are discussed in detail in this chapter.

Tuxedo Changes

Tuxedo users need to make the following environment changes:

- If an existing Tuxedo application is already using Tuxedo /T DOMAINS, then a new domain must be added to the domains configuration file for each connection to a WebLogic Tuxedo Connector instantiation.
- If the existing Tuxedo application does not use domains, then the domain servers must be added to the *TUXCONFIG* of the application. A new *DMCONFIG* must be created with a Tuxedo /T Domain entry corresponding to the WebLogic Tuxedo Connector instantiation.
- WebLogic Tuxedo Connector requires that the Tuxedo domain always have encoding turned on. *MTYPE* should always be unset or set to NULL in the *DMCONFIG* file.

Note: For more information on *DMCONFIG* files, see the *BEA TUXEDO Domains Guide*.

WebLogic Server Changes

WebLogic Server users need to make the following environment changes:

- Create Java clients or servers.
- Create a WebLogic Tuxedo Connector XML configuration file.
- Deploy WebLogic Tuxedo Connector on the WebLogic Tuxedo Connector.

Note: For more information on creating WebLogic Tuxedo Connector clients or servers, see the *WebLogic Tuxedo Connector ATMI Programmer's Guide*.

WebLogic Server ThreadPoolSize

The number of client threads available when dispatching services from the gateway may limit the number of concurrent services running. For this release of WebLogic Tuxedo Connector, there is no WebLogic Tuxedo Connector XML configuration file parameter such as *MAXDISPATCHTHREADS* or *MINDISPATCH* threads. Use a

reasonable thread model when invoking service EJBs. You may need to increase the WebLogic Server threadpool size in the `WebLogic Server config.xml` file to a large value.

Example: `ThreadPoolSize="70"`.

For more information on WebLogic Server performance and tuning, see *BEA WebLogic Server Performance and Tuning*, Release 6.0.

Installing the WebLogic Tuxedo Connector

You install the WebLogic Tuxedo Connector as a WebLogic Server startup class and a WebLogic Server shutdown class.

Use the following steps to install the WebLogic Tuxedo Connector on your system.

1. Download the WebLogic Tuxedo Connector Release 1.0 distribution from the BEA Download Center, located at <http://www.beasys.com/download.shtml>. The WebLogic Tuxedo Connector is available from the Tuxedo 8.0 download page.
2. Execute the WebLogic Tuxedo Connector install program.
 - Windows NT/2000: run the `wtc10_win.exe` file
 - UNIX: run the `wtc10_unix.bin` file
3. Click **NEXT**.

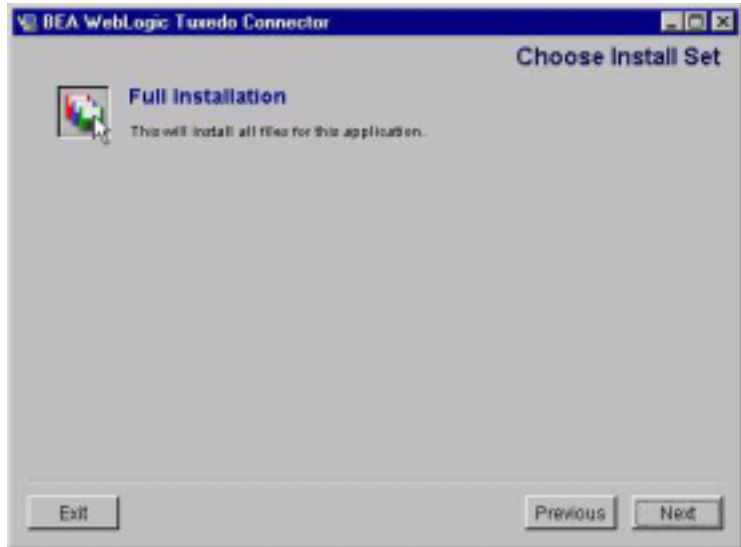
2 *Installing and Configuring WebLogic Tuxedo Connector*



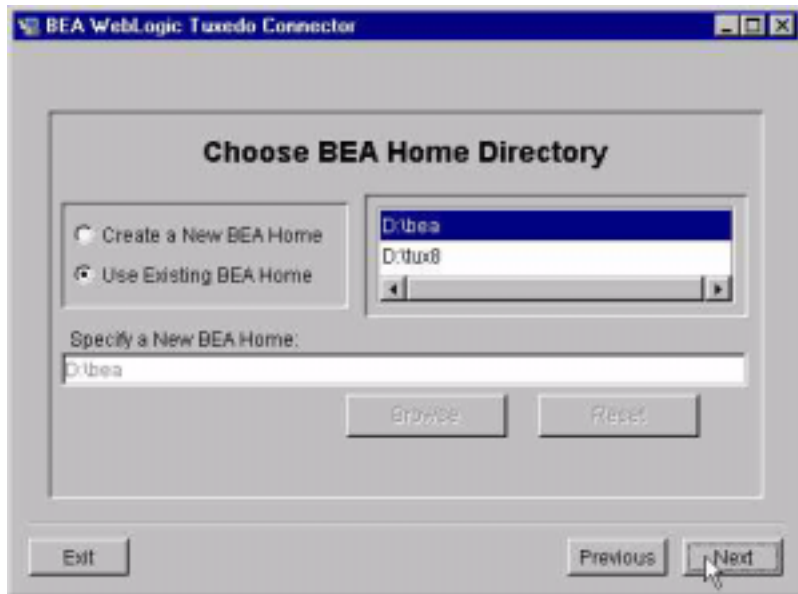
4. Read the License Agreement.
5. Click **Yes**.



6. Click **Next**.
7. Select **Full Installation**.



8. Click **Next**.
9. Select a BEA home directory.
 - If you have an existing BEA directory, click **NEXT**.
 - If you want to create a new BEA directory, click **Create a New BEA Home**. Specify the new directory location and click **NEXT**.



10. Select a directory for the WebLogic Tuxedo Connector and click **Install**.

11. The installation is complete. Click **Done**.

Configuring WebLogic Tuxedo Connector for Your Applications

Note: This release of the WebLogic Tuxedo Connector provides only static configuration. If you need to change any parameters in the XML configuration file, the WebLogic Server must be restarted for the changes to take effect. For example, you can not add or remove domain network links, change network addresses, or import or export new services.

This section provides an information on how to configure the WebLogic Tuxedo Connector to allow WebLogic Server applications and Tuxedo applications to interoperate. Use the following main steps to configure the connector:

- [Create a Configuration File](#)
- [Set the WebLogic Server Domain Environment](#)
- [Update the Start Server Script](#)
- [Create a StartUp Class](#)
- [Create a Shutdown Class](#)
- [Deploy EJBs](#)
- [Restart the Application Server](#)
- [Verify Installation](#)

Create a Configuration File

The WebLogic Tuxedo Connector uses an XML configuration file to describe the Tuxedo /T DOMAINS connections used to link WebLogic Server to Tuxedo.

Note: This file must be installed on all WebLogic Server with JATMI instances.

- Create an XML configuration file by using a text editor, such as vi or WordPad.
- The most efficient method to create a new configuration file is to modify one of the example configuration files located in the WebLogic Tuxedo Connector distribution examples directory, such as: `\examples\simpapp\bdmconfig.xml`
- Save the configuration file with your WebLogic Server application.

Note: For more information on configuration file hierarchy and element definitions, see [Chapter 8, “The WebLogic Tuxedo Connector XML Configuration File.”](#)

Configuration File Components

A WebLogic Tuxedo Connector configuration file consists of the following sections:

- Version
- DOCTYPE Declaration
- WTC_CONFIG
- BDMCONFIG
- tBridge

Version

Required. Specifies the version of XML used.

Example: `<?xml version="1.0"?>`

DOCTYPE

Note: You must modify the DOCTYPE with the name of your configuration file and specify the fully qualified path to its location.

Required. The DOCTYPE declaration provides the location of the `wtc_config.dtd`. When the WebLogic Server is started, the WebLogic Tuxedo Connector XML configuration file is checked against the document type definition (DTD) for errors.

Example:

```
<!DOCTYPE <your configuration file name> SYSTEM "file:<the WTC
installation directory>\weblogic\wtc\gwt\wtc_config_1_0.dtd">
```

WTC_CONFIG

Required. `WTC_CONFIG` element is the root of the configuration file. `WTC_CONFIG` has two children:

- BDMCONFIG
- tBridge

BDMCONFIG

Required. BDMCONFIG describes connectivity information and security protocols used by the WebLogic Tuxedo Connector to process service requests between WebLogic Server and Tuxedo. These configuration parameters are analogous to the interoperability attributes required for communication between Tuxedo domains.

Note: For more information on how to configure BDMCONFIG, see [Chapter 3](#), “Configuring BDMCONFIG.”

tBridge

Optional. The tBridge is a bi-directional JMS interface to imported Tuxedo services. Configuring this section causes the tBridge to start as part of the WebLogic Server application environment.

Note: For more information on how to configure the tBridge, see [Chapter 4](#), “Configuring tBridge.”

Validate the XML file

Validate your configuration file using `WTCValidateCF`. This utility allows you to validate the XML configuration file before booting WebLogic Server.

To validate the XML configuration file, enter the following command:

```
> java weblogic.wtc.gwt.WTCValidateCF your_XML_configuration_file
```

where *your_XML_configuration_file* is the name of your XML configuration file.

Set the WebLogic Server Domain Environment

Update the domain environment of your WebLogic Server application that you will use with the WebLogic Tuxedo Connector. Use the following steps to add the pathname of the `jamti.jar` file to your application's `CLASSPATH`.

1. From the command line, change directories to the location of the WebLogic Server application.

2. Edit the `setEnv` file with a text editor, such as `vi` or WordPad.

- NT/2000 users: `edit setEnv.cmd`

- UNIX users: `edit setEnv.sh`

Note: The `setExamplesEnv` file is used to set the environment for the WebLogic Tuxedo Connector examples provided with your distribution.

3. Add the pathname of the `jamti.jar` file to the `CLASSPATH`.

Example:

```
set HOME=D:\bea\wlserver6.0sp1
```

```
set JAVA_HOME=D:\bea\jdk130
```

```
set CLASSPATH=%JAVA_HOME%\lib\tools.jar;D:\bea\wtc1.0\lib\jamti.jar;%HOME%\lib\weblogic_sp.jar;%HOME%\lib\weblogic.jar;%CLASSPATH%
```

4. Save the file.

5. Set your application environment.

- NT/2000 users: `run setEnv.cmd`

- UNIX users: `run setEnv.sh`

The domain environment is updated.

Update the Start Server Script

Update the script that starts WebLogic Server. Use the following steps to add the pathname of the WebLogic Tuxedo Connector `jamti.jar` file to the `CLASSPATH` of your WebLogic Server.

1. Edit the `startWebLogic` file with a text editor, such as `vi` or WordPad.

- NT/2000 users: `edit startWebLogic.cmd`

- UNIX users: `edit startWebLogic.sh`

2. Add the pathname of the `jamti.jar` file to the `CLASSPATH`.

Example:

```
set CLASSPATH=.;.\lib\weblogic_sp.jar;d:\bea\wtc1.0\lib\jatmi.jar
;.\lib\weblogic.jar;
```

3. Save the file.
4. Run `startWebLogic`.
 - NT/2000 users: `run startWebLogic.cmd`
 - UNIX users: `run startWebLogic.sh`

WebLogic Server starts.

Create a StartUp Class

Create a startup class and assign the location of the configuration file as a property of the startup class. Use the following steps to create a Startup Class for a domain:

1. Start the WebLogic Server Console.
2. If necessary, right-click on the domain root and select **Create or edit other domains**. Left-click to select a domain from the repository.
3. Left-click and expand the **Deployments** branch.
4. Right-click on the **Startup & Shutdown** branch.
5. Select **Create a new Startup Class ...**
The *Configuration* tab is active.
6. Enter the **Name**.
Example: `MyWTCStartup Class`
7. Enter `weblogic.wtc.gwt.WTCStartup` for the **Classname**.
8. Enter the **Arguments**. If more than one argument is used, separate them using a comma. Valid arguments include:
 - BDMCONFIG**: This required argument specifies the location of the WebLogic Tuxedo Connector XML configuration file.
Example: `BDMCONFIG=.\config\mydomain\wtc_config.xml`
 - **TraceLevel**: This optional argument specifies the level of error tracing used.
Example:
`BDMCONFIG=.\config\mydomain\wtc_config.xml,TraceLevel=100000`

- **PasswordKey**: This optional argument specifies the password key used for encrypting passwords when configuring local and remote domains.
`BDMCONFIG=. \config\mydomain\wtc_config.xml, PasswordKey=mykey`
9. Check **Abort startup on failure**.
 10. Click **Create**.
 11. Select the *Target* tab.
 12. From the Available servers list, click on the server you wish to select.
The server is highlighted.
 13. Click on the right-arrow.
The selected server appears in the *Chosen* servers list.
 14. Click **Apply**.

Create a Shutdown Class

Use the following steps to create a shutdown class for a domain:

1. Right-click on the **Startup & Shutdown** branch.
2. Select **Create a new Shutdown Class ...**
3. Enter the **Name**.
Example: MyWTCTShutdown Class
4. Enter `weblogic.wtc.gwt.WTCTShutdown` for the **Classname**.
5. Click **Create**.

Deploy EJBs

Deploy any EJBs that are referenced in the WebLogic Tuxedo Connector XML configuration file.

Restart the Application Server

To make the console changes active, you must shut down and restart the application server.

Verify Installation

Check the WebLogic Server log file. If the WebLogic Tuxedo Connector started properly, messages similar to those shown below are displayed:

```
####<Apr 20, 2001 10:55:08 PM EDT> <Info> <WTC> <mymachine>
<myserver> <Thread-1> <system> <> <180001> <Done Loading the XML
config file.>

####<Apr 20, 2001 10:55:08 PM EDT> <Debug> <WTC> <mymachine>
<myserver> <Thread-2> <> <> <180056> <[/WTCStartup/run/>

####<Apr 20, 2001 10:55:08 PM EDT> <Debug> <WTC> <mymachine>
<myserver> <Thread-2> <> <> <180056> <[/WTCStartup/run/05>

####<Apr 20, 2001 10:55:09 PM EDT> <Info> <WebLogicServer>
<randyr-nt> <myserver> <Thread-1> <system> <> <000288>
<weblogic.wtc.gwt.WTCStartup reports: WTC started...>
```

Check the WebLogic Server `config.xml` file. The WebLogic Tuxedo Connector configuration is added to the WebLogic Server StartUp and Shutdown Classes.

```
<StartupClass
Arguments="BDMCONFIG=d:\wtc_load4\examples\simpapp\bdmconfig.xml"
ClassName="weblogic.wtc1.0.gwt.WTCStartup" FailureIsFatal="false"
Name="MyWTCStartup Class" Targets="myserver"/>

<ShutdownClass Arguments=" "
ClassName="weblogic.wtc1.0.gwt.WTCShutdown" Name="MyWTCShutdown
Class"/>
```

Note: If your installation is not successful, see [Chapter 7, “Troubleshooting The WebLogic Tuxedo Connector.”](#)

Uninstalling the WebLogic Tuxedo Connector

This section describes how to remove the WebLogic Tuxedo Connector from WebLogic Server and from your system.

Removing WebLogic Tuxedo Connector from WebLogic Server

This section provides an information on how to remove the WebLogic Tuxedo Connector using the WebLogic Server Console:

1. Shutdown the WebLogic Server instance that uses WebLogic Tuxedo Connector.
2. Remove necessary EJBs.
3. Remove the StartUp class.
4. Remove the Shutdown class.
5. Remove the path to the `jamti.jar` file from the CLASSPATH of the `setEnv.cmd` or `setEnv.sh` file.
6. Remove the path to the `jamti.jar` file from the CLASSPATH of the `startWebLogic.cmd` or `startWebLogic.sh` file.
7. Restart WebLogic Server.

The WebLogic Tuxedo Connector is removed from your WebLogic Server application.

Removing WebLogic Tuxedo Connector from the System

This section provides information on how to remove WebLogic Tuxedo Connector from your system.

GUI

Use the following steps to uninstall WebLogic Tuxedo Connector using the GUI.

2 *Installing and Configuring WebLogic Tuxedo Connector*

1. Click the Windows **Start** button.
2. Select **Programs > BEA WebLogic E-Business Platform > BEA WebLogic Tuxedo Connector 1.0 > uninstallwtc10_win**.

Command Line

Use the following steps to uninstall the WebLogic Tuxedo Connector from a command line:

1. Change directories to the WebLogic Tuxedo Connector uninstaller directory.
2. Run the uninstall program.
 - Windows NT: run the `uninstallwtc10_win.exe` file.
 - Unix: run the `uninstallwtc10_Unix.bin` file.

Using the Uninstaller

1. Click **Uninstall**.



2. Click **Exit**.



3 Configuring BDMCONFIG

The BDMCONFIG section of the WebLogic Tuxedo Connector XML configuration file describes how to establish connectivity and provide security between domains in the WebLogic Tuxedo Connector and Tuxedo environments. The XML configuration file is composed of configuration parameters that are analogous to the interoperability attributes required for the communication between Tuxedo domains.

The WebLogic Tuxedo Connector is started as part of the WebLogic Server application environment. Any configuration condition that prevents the WebLogic Tuxedo Connector from starting results in an error being logged to the WebLogic Server error log.

The following sections provide configuration information about BDMCONFIG:

- [Establishing Connectivity](#)
- [Dynamic Status](#)
- [Configuring Failover and Failback](#)
- [Establishing Security](#)

Note: For more detailed reference information on the WebLogic Tuxedo Connector XML configuration file, elements and attributes, and the `wtc_config.dtd`, see [“The wtc_config.dtd.”](#)

Establishing Connectivity

Several options can specify the conditions under which a local domain gateway tries to establish a connection with a remote domain. Specify these conditions using the `ConnectionPolicy` parameter in the `T_DM_LOCAL_TDOMAIN` and `T_DM_REMOTE_TDOMAIN` sections of `BDMCONFIG`.

- Valid values for local domains are: `ON_DEMAND`, `ON_STARTUP`, or `INCOMING_ONLY`.
- Valid values for remote domains are: `ON_DEMAND`, `ON_STARTUP`, `INCOMING_ONLY`, or `LOCAL`.

Connecting at Boot Time (`ON_STARTUP`)

A policy of `ON_STARTUP` means that a domain gateway attempts to establish a connection with its remote domain access points at gateway server initialization time. The connection policy retries failed connections at regular intervals determined by the `RetryInterval` parameter.

RetryInterval

The `RetryInterval` parameter enables failed attempts at connections to remote domains to be retried automatically if the connection policy is `ON_STARTUP`. You can control the frequency of automatic connection attempts by specifying the interval (in seconds) during which the gateway should wait before trying to establish a connection again.

- Minimum value: 0
- Maximum value: 2147483647
- Default setting: 60

MaxRetries

The `MaxRetries` number indicates the number of times that a domain gateway tries to establish connections to remote domain access points before quitting. Use only when `ConnectionPolicy` is set to `ON_STARTUP`.

- Minimum value: 0
- Maximum value: 2147483647
- Default value: 2147483647

Use the maximum value to retry processing until a connection is established. Use the minimum value to disable the automatic retry mechanism.

Connecting on Request (ON_DEMAND)

A connection policy of ON_DEMAND means that a connection is attempted only when requested by either a client request to a remote service or an administrative connect command. The default setting for ConnectionPolicy is ON_DEMAND.

Accepting Incoming Connections (INCOMING_ONLY)

A connection policy of INCOMING_ONLY means that a domain gateway does not attempt an initial connection to remote domain access points at startup. The domain gateway is available for incoming connections from remote domain access points and remote services are advertised when the domain gateway for this local domain access point receives an incoming connection. Connection retry processing is not allowed when the connection policy is INCOMING_ONLY.

LOCAL

A connection policy of LOCAL indicates that a remote domain connection policy is explicitly defaulted to the local domain ConnectionPolicy attribute value. If the remote domain ConnectionPolicy is not defined, the system uses the setting specified by the associated local domain (specified by the LocalAccessPoint).

Dynamic Status

Dynamic Status is a feature of the gateway process (GWTDOMAIN) to determine the availability of remote services. The connection policy used in the WebLogic Tuxedo Connector configuration file determines whether the Dynamic Status feature is available for a service. The following table describes how each connection policy affects Dynamic Status capability.

ON_STARTUP	Dynamic Status is on. Services imported from a remote domain are advertised while a connection to that remote domain exists.
ON_DEMAND	Dynamic Status is off. Services imported from remote domains are always advertised.
INCOMING_ONLY	Dynamic Status is on. Remote services are initially suspended. The domain gateway is available for incoming connections from remote domains. Remote services are advertised when the local domain gateway receives an incoming connection.

Configuring Failover and Failback

The WebLogic Tuxedo Connector supports domain level failover and failback

Note: In the Tuxedo T/ Domain, there is a limit of 3 backup remote domains. The WebLogic Tuxedo Connector has no limit to the number of backup domains allowed to be configured for a service.

Domain Failover and Failback

Domain failover provides an alternate access to domain services when a failure is detected on a primary remote domain. Failback is provided if a connection to the primary domain is restored when the domain becomes available.

- Configure domains with a ConnectionPolicy of ON_STARTUP or INCOMING_ONLY to enable failover/failback on the WebLogic Tuxedo Connector.
- A connection policy of ON_DEMAND assumes the domain is always available

Establishing Security

The WebLogic Tuxedo Connector supports authentication of clients, servers, and administrative programs.

- Clients present passwords and are authenticated before joining applications
- Servers are authenticated to be running as the user identified by the application administrator
- Access Control Lists (ACLs) are available to control access to services

Domains Passwords

The Security parameter in the local domain specifies the level of security allowed by a particular local domain. There are three basic security levels:

- NONE: Incoming connections from a remote domain are not authenticated. This is the default value.
- APP_PW: Incoming connections from remote domains are authenticated using the application password defined in the T_DM_PASSWORD element.

- **DM_PW:** Domain password security is enforced when a connection is established from a remote domain. Domain passwords must be defined in the **T_DM_PASSWORD** element.

Generating Encrypted Passwords

Use `weblogic.wtc.gwt.genpasswd` to generate encrypted passwords for **LocalPassword**, **RemotePassword**, and **AppPassword** elements. The utility uses a key to encrypt a password that is copied into the WebLogic Tuxedo Connector XML configuration file. The result is a valid WebLogic Tuxedo Connector XML element.

Note: Use of encryption requires appropriate user licenses. For more information, see [“Licensing” on page 1-7](#).

- The XML configuration file does not store clear text passwords.
- The key must be included in the argument field of the **Startup** class to provide the WebLogic Tuxedo Connector access to the key on startup.
 - Use the `PasswordKey` parameter to assign the key.
 - The `PasswordKey` argument can only be assigned one key value.
 - If there are no **T_DM_PASSWORD** entries in the XML configuration file, the `PasswordKey` argument is ignored.

Usage

Call the utility without any arguments to display the command line options.

Example:

```
$ java weblogic.wtc.gwt.genpasswd
```

```
Usage: genpasswd Key <LocalPassword|RemotePassword|AppPassword>  
<local|remote|application>
```

Examples

This section provides examples of each of the password element types.

LocalPasswords

The following example uses *key1* to encrypt “LocalPassword1” as the password of the local domain.

```
$ java weblogic.wtc.gwt.genpasswd Key1 LocalPassword1 local
<LocalPassword IV="I#^Da0efo1">!djk*87$klbJJ</LocalPassword>
```

RemotePasswords

The following example uses *mykey* to encrypt “RemotePassword1” as the password for the remote domain.

```
$ java weblogic.wtc.gwt.genpasswd mykey RemotePassword1 remote
<RemotePassword IV="Rq$45%%kK">McFrd3#f41Kl</RemotePassword>
```

AppPasswords

The following example uses *key1* to encrypt “test123” as the application password.

```
$ weblogic.wtc.gwt.genpasswd mykey test123 application
<AppPassword IV="gx8aSkAgLFg=">c98Y/P94HY3rCAVmkF=</AppPassword>
```

Access Control Lists

Access Control Lists (ACLs) limit the access to local services within a local domain by restricting the remote domains that can execute these services. Inbound policy from a remote domain is specified using the *AclPolicy* element. Outbound policy towards a remote domain is specified using the *CredentialPolicy* element. This allows WebLogic Server and Tuxedo applications to share the same set of users and the users are able to propagate their credentials from one system to the other.

The valid values for this parameter are:

- LOCAL: The domain gateway’s security token is passed.
- GLOBAL: The user’s security token is passed.

Security Requirements for servers

- If the RemoteAccessPoint is running with an AclPolicy of LOCAL, then any requests from that RemoteAccessPoint must have the credentials of the WebLogic Tuxedo Connector instantiation.
- If the RemoteAccessPoint is running with an AclPolicy of GLOBAL, then any requests from that RemoteAccessPoint must have the user identity from the token received on the request.

Security Requirements for clients

- If a remote domain is running with the CredentialPolicy set to LOCAL then the request to Tuxedo has the credentials of the remote domain gateway.
- If a remote domain is running with the CredentialPolicy set to GLOBAL then the WebLogic Tuxedo Connector constructs tokens based on the identity of the caller.

Establishing an ACL Policy

Use the following steps to establish an ACL policy:

1. [Add Users to TpUserFile](#)
2. [Modify EJBs for ACL](#)
3. [Modify Tuxedo Environment for ACL](#)
4. [Modify WebLogic Tuxedo Connector Environment for ACL](#)

Note: Tuxedo 6.5 does not have the required security infrastructure to support security mapping.

Add Users to TpUserFile

Add users to the TpUsrFile using the WebLogic Server Console.

Modify EJBs for ACL

Add the security-role and security-role-assignment elements to each EJB used in the application.

After you have added the users and modified the EJBs, only WebLogic Server defined users have permission to access the EJBs.

Modify Tuxedo Environment for ACL

Perform the following steps for inbound and outbound requests to prepare the Tuxedo environment:

1. Add the group using `tpgrpadd`.
2. Add users using `tpusradd`.
3. Add the service to be protected by TUXEDO ACL using `tpacladd`.
4. Set the BDMCONFIG for the remote domain (the WebLogic Server domain) with the `ACL_POLICY="GLOBAL"`.

Note: If `ACL_POLICY="LOCAL"`, you must configure the remote `DOMAINID` as a user using `tpusradd`.

Modify WebLogic Tuxedo Connector Environment for ACL

Perform the following steps to prepare the WebLogic Server environment:

1. Copy the `tpusr` file from TUXEDO to the application environment or generate your own `tpusr` file.
2. Add a `TpUserFile` element to the `T_DM_REMOTE_TDOMAIN` section of the XML configuration file.

Example: `<TpUsrFile>full path name to tpusr</TpUsrFile>`.

3. Add a `CredentialPolicy` element to the `T_DM_REMOTE_TDOMAIN` section of the XML configuration file.

Example: `<CredentialPolicy>GLOBAL</CredentialPolicy>`

Note: If the CredentialPolicy value is set to LOCAL, the user information is stripped off.

Example ACL Policy

This section provides an example of how to set up ACL control using the simpapp and simperv examples.

- Only John and Bob have access to Toupper.
- Only Dan and John to access Tolower.
- Toupper is used for accessing remote the Tuxedo service TOUPPER.
- Tolower provides the actual service for remote Tuxedo user.

Use the following steps to establish ACL control:

1. Add user John, Bob, and Dan to WebLogic Security using the WebLogic Server Console.
2. Modify the `ejb-jar.xml` to add the `security-role` and `security-role-assignment` elements for the Tuxedo TOUPPER service.

Note: The `|` at beginning of the line indicates the changes added to support the security implementation.

```
<?xml version="1.0"?>
```

```
<!--
```

```
    Copyright (c) 2000 BEA Systems, Inc.  
    All rights reserved
```

```
    THIS IS UNPUBLISHED PROPRIETARY  
    SOURCE CODE OF BEA Systems, Inc.
```

```
    The copyright notice above does not  
    evidence any actual or intended  
    publication of such source code.
```

```
-->
```

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
```

Enterprise JavaBeans 2.0//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd'>

```

    <ejb-jar>
      <enterprise-beans>
        <session>
          <ejb-name>Toupper</ejb-name>

    <home>weblogic.wtc.examples.simpapp.ToupperHome</home>
      <remote>weblogic.wtc.examples.simpapp.Toupper</remote>
    <ejb-class>weblogic.wtc.examples.simpapp.ToupperBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>dom2</role-name>
    </security-role>
    <method-permission>
      <role-name>dom2</role-name>
      <method>
        <ejb-name>Toupper</ejb-name>
        <method-name>Toupper</method-name>
      </method>
    </method-permission>
    <container-transaction>
      <method>
        <ejb-name>Toupper</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Supports</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>

```

3. Modify the `weblogic-ejb-jar.xml` to add the `security-role` and `security-role-assignment` elements for the Tuxedo TOUPPER service.

Note: The | at beginning of the line indicates the changes added to support the security implementation.

```
<?xml version="1.0"?>
<!--
```

Copyright (c) 2000 BEA Systems, Inc.
All rights reserved

THIS IS UNPUBLISHED PROPRIETARY
SOURCE CODE OF BEA Systems, Inc.
The copyright notice above does not
evidence any actual or intended
publication of such source code.

```
-->
```

```
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB//EN"
'http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd'
```

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>Toupper</ejb-name>
    <stateful-session-descriptor>
      <stateful-session-cache>
        <max-beans-in-cache>100</max-beans-in-cache>
      </stateful-session-cache>
    </stateful-session-descriptor>
    <jndi-name>tuxedo.services.ToupperHome</jndi-name>
  </weblogic-enterprise-bean>
  | <security-role-assignment>
  |   <role-name>dom2</role-name>
  |   <principal-name>john</principal-name>
  |   <principal-name>bob</principal-name>
  | </security-role-assignment>
</weblogic-ejb-jar>
```

4. Modify the `ejb-jar.xml` to add the `security-role` and `security-role-assignment` elements for the Tolower service.

Note: The | at beginning of the line indicates the changes added to support the security implementation.

```
<?xml version="1.0"?>
<!--
```

Copyright (c) 2000 BEA Systems, Inc.
All rights reserved

THIS IS UNPUBLISHED PROPRIETARY
SOURCE CODE OF BEA Systems, Inc.
The copyright notice above does not
evidence any actual or intended
publication of such source code.

```
-->
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" 'http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd'>
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>Tolower</ejb-name>
      <home>weblogic.wtc.jatmi.TuxedoServiceHome</home>
      <remote>weblogic.wtc.jatmi.TuxedoService</remote>
<ejb-class>weblogic.wtc.examples.simpsserv.TolowerBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>

  <assembly-descriptor>
    <security-role>
      <role-name>rdom2</role-name>
    </security-role>
    <method-permission>
      <role-name>rdom2</role-name>
      <method>
        <ejb-name>Tolower</ejb-name>
        <method-name>service</method-name>
      </method>
    </method-permission>
    <container-transaction>
      <method>
        <ejb-name>Tolower</ejb-name>
```

```
        <method-intf>Remote</method-intf>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Supports</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

5. Modify the `Weblogic-ejb-jar.xml` to add the `security-role` and `security-role-assignment` elements for the Tolower service.

Note: The `|` at beginning of the line indicates the changes added to support the security implementation.

```
<?xml version="1.0"?>
<!--
```

Copyright (c) 2000 BEA Systems, Inc.
All rights reserved

THIS IS UNPUBLISHED PROPRIETARY
SOURCE CODE OF BEA Systems, Inc.
The copyright notice above does not
evidence any actual or intended
publication of such source code.

```
-->
```

```
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB//EN"
"http://www.bea.com/servers/wls600/dtd/weblogic-ejb-jar.dtd">
```

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>Tolower</ejb-name>
    <stateless-session-descriptor>
      <pool>
        <max-beans-in-free-pool>100</max-beans-in-free-pool>
      </pool>
    </stateless-session-descriptor>
    <jndi-name>tuxedo.services.TOLOWERHome</jndi-name>
```

```

|         </weblogic-enterprise-bean>
|         <security-role-assignment>
|             <role-name>rdom2</role-name>
|             <principal-name>john</principal-name>
|             <principal-name>dan</principal-name>
|         </security-role-assignment>
|     </weblogic-ejb-jar>

```

6. Perform the following steps to prepare the Tuxedo environment for outbound requests:
 - If needed, add the group using `tpgrpadd`.
 - If needed, add John, Bob, and Dan as users using `tpusradd`.
 - Add TOUPPER service to be protected by TUXEDO ACL using `tpacladd`.
 - Set the BDMCONFIG for the remote domain (the WebLogic Server domain) with the `ACL_POLICY="GLOBAL"`.
7. Perform the following steps to prepare the Tuxedo environment for inbound requests:
 - If needed, add the group using `tpgrpadd`.
 - If needed, add John, Bob, and Dan as users using `tpusradd`.
 - Add TOUPPER service to be protected by TUXEDO ACL using `tpacladd`.
 - Set the BDMCONFIG for the remote domain (the WebLogic Server domain) with the `ACL_POLICY="GLOBAL"`.

Note: If `ACL_POLICY="LOCAL"`, you must configure the remote `DOMAINID` as a user using `tpusradd`.

8. Perform the following steps to prepare the WebLogic Server environment:
 - Copy the `tpusr` file from TUXEDO or generate your own `tpusr` file.
 - Add a `TpUserFile` element to the `T_DM_REMOTE_TDOMAIN` section of the XML configuration file.
Example: `<TpUsrFile>full path name to tpusr</TpUsrFile>`.
 - Add a `CredentialPolicy` element to the `T_DM_REMOTE_TDOMAIN` section of the XML configuration file. Set the value to `GLOBAL`.
Example: `<CredentialPolicy>GLOBAL</CredentialPolicy>`

4 Configuring tBridge

The following sections provide information on tBridge functionality and configuration.

- [Overview of the tBridge](#)
- [WebLogic Tuxedo Configuration XML File Configuration for tBridge](#)
- [tBridge Connectivity](#)
- [Example Connection Type Configurations](#)
- [Priority Mapping](#)
- [Error Queues](#)

Note: For more detailed reference information on the WebLogic Tuxedo Connector XML configuration file, elements and attributes, and the `wtc_config.dtd`, see [Chapter 9, “The wtc_config.dtd.”](#)

Overview of the tBridge

The tBridge is a part of the WebLogic Tuxedo Connector that provides a bi-directional JMS interface for your WebLogic Server applications communicate to Tuxedo application environments. The transfer of messaging between the environments consists of JMS based messages containing text, Byte, or XML data streams used to invoke services on behalf of the client application.

The following features determine the functionality of the tBridge:

- Connectivity is determined by the configuration of the parameters in the tBridge section of the WebLogic Tuxedo Connector XML configuration file. No additional application programming is required.
- The tBridge uses Java Messaging Service (JMS) to provide an interface to a Tuxedo /Q or a Tuxedo service.
- The tBridge provides simple translation between XML and FML32 to provide connectivity to existing Tuxedo systems.

WebLogic Tuxedo Configuration XML File Configuration for tBridge

The WebLogic Tuxedo Connector tBridge connectivity is determined by the XML configuration file which contains the necessary information to establish a connection to Tuxedo.

Starting the tBridge

The tBridge is started as part of the WebLogic Server application environment if the tBridge section of the XML configuration file is configured. Any configuration condition that prevents the tBridge from starting results in an error being logged.

Error Logging

WebLogic Tuxedo Connector errors are logged to the WebLogic Server error log.

tBridge Connectivity

The tBridge establishes a one-way data connection between instances of a JMS queue and a Tuxedo /Q or a JMS queue and a Tuxedo service. This connection is represented in the tBridge section of the configuration file as a <fromto> element. Each data connection provides a one-to-one connection between the identified points. Three types of connections can be configured. Each connection type is represented in the tBridge section of the configuration file as the value of the <direction> element. The following is a description of each of the connection types:

- **JmsQ2TuxQ**: Reads from a given JMS queue and transports the messages to the specified Tuxedo /Q
- **TuxQ2JmsQ**: Reads from a Tuxedo /Q and transports the messages to JMS.
- **JmsQ2TuxS**: Reads from a given JMS queue, synchronously calls the specified Tuxedo service, and places the reply back onto a specified JMS queue.

Note: JMS message types: MapMessage, ObjectMessage, StreamMessage are not valid in WebLogic Tuxedo Connector. If one of these message types is received by the tBridge, a log entry is generated indicating this is an unsupported type and the message is discarded.

Example Connection Type Configurations

The following sections provide example configurations for each connection type.

Example JmsQ2TuxQ Configuration

The following section provides example code for reading from a JMS queue and sending to Tuxedo /Q.

```

<fromto>
    <direction>JmsQ2TuxQ</direction>
    <source>
        <Name>weblogic.jms.Jms2TuxQueue</Name>
    </source>
    <target>
        <AccessPoint>TDOM2</AccessPoint>
        <Qspace>QSPACE</Qspace>
        <Name>STRING</Name>
    </target>
    <replyQ>RPLYQ</replyQ>
    <translateFML>NO</translateFML>
</fromto>

```

The following section describes the components of the **JmsQ2TuxQ** configuration:

- The **<direction>** connection type *JmsQ2TuxQ*.
- The **<source>** **<Name>** keyword specifies the name of the JMS queue to read is *weblogic.jms.Jms2TuxQueue*. The tBridge establishes a JMS client session to this queue using **CLIENT_ACKNOWLEDGE** semantics.
- The **<target>** keyword specifies the elements necessary to explicitly reference the destination.
 - The **<AccessPoint>** keyword specifies the name of the access point is *TDOM2*
 - The **<Qspace>** keyword specifies the name of the Qspace is *Qspace*.
 - The **<Name>** keyword specifies the name of the queue is *STRING*.
- The **<replyQ>** keyword specifies the name of a JMS reply queue is *ReplyQ*. Use of this queue causes *tpenqueue* to provide **TMFORWARD** functionality.
- The **<translateFML>** keyword *NO* specifies that no data translation is provided by the tBridge.

The following table provides information on JmsQtoTuxQ message mapping:

From: JMS Message Type	To: WebLogic Tuxedo Connector JATMI (Tuxedo)
BytesMessage	TypedCArray
TextMessage (translateFML = NONE)	TypedString

From: JMS Message Type	To: WebLogic Tuxedo Connector JATMI (Tuxedo)
TextMessage (translateFML = FLAT)	TypedFML32

Example TuxQ2JmsQ Configuration

The following section provides example code for importing Tuxedo /Q into WebLogic Server.

```
<T_DM_IMPORT
  ResourceName="QSPACE"
  LocalAccessPoint="LDM2"
  RemoteAccessPointList="MYLOCAL"
<TranTime>600</TranTime>
</T_DM_IMPORT>
```

Where

- `ResourceName` is the Tuxedo /Q described in the `*DM_LOCAL_SERVICES` section of your `BDMCONFIG` file.
- `LocalAccessPoint` is the name of your WebLogic Server domain described in the `*DM_REMOTE_DOMAINS` section of your `BDMCONFIG` file.
- `RemoteAccessPointList` is the name of your Tuxedo domain described in the `*DM_LOCAL_DOMAINS` section of your `BDMCONFIG` file.

The following section provides example code for reading from a Tuxedo /Q and sending to a JMS queue.

```
<fromto>
  <direction>TuxQ2JmsQ</direction>
  <source>
    <AccessPoint>TDOM2</AccessPoint>
    <Qspace>QSPACE</Qspace>
    <Name>STRING</Name>
  </source>
  <target>
    <Name>weblogic.jms.Tux2JmsQueue</Name>
  </target>
  <translateFML>NO</translateFML>
</fromto>
```

The following section describes the components of the **TuxQ2JmsQ** configuration:

- The <direction> connection type is *TuxQ2JmsQ*.
- The <source> <Name> keyword specifies the name of the JMS queue to read is *weblogic.jms.Tux2JmsQueue*.
- The <target> specifies the elements necessary to explicitly reference the destination.
 - The <AccessPoint> keyword specifies the name of the access point is *TDOM2*.
 - The <Qspace> keyword specifies the name of the Qspace is *Qspace*.
 - The <Name> keyword specifies the name of the queue is *STRING*.
- The <translateFML> keyword *NO* specifies that no data translation is provided by the tBridge.

The following table provides information on TuxQ2JmsQ message mapping:

From: WebLogic Tuxedo Connector JATMI (Tuxedo)	To: JMS Message Type
TypedCArray	BytesMessage
TypedString (translateFML = NONE)	TextMessage
TypedFML32 (translateFML = FLAT)	TextMessage
TypedFML (translateFML = FLAT)	TextMessage
TypedXML	TextMessage

Example JmsQ2TuxS Configuration

The following section provides example code for reading from a JMS queue, calling a Tuxedo service, and then writing the results back to a JMS queue.

```
<fromto>
    <direction>JmsQ2TuxS</direction>
    <source>
        <Name>weblogic.jms.Jms2TuxQueue</Name>
    </source>
    <target>
```

```

        <AccessPoint>TDOM2</AccessPoint>
    <Name>REVERSE_STRING</Name>
    </target>
    <replyQ>weblogic.jms.Tux2JmsQueue</replyQ>
    <translateFML>FLAT</translateFML>
</fromto>

```

The following section describes the components of the **JmsQ2TuxS** configuration:

- The `<direction>` connection type is *JmsQ2TuxS*.
- The `<source>` `<Name>` keyword specifies the name of the JMS queue to read is *weblogic.jms.Jms2TuxQueue*.
- The `<target>` specifies the elements necessary to explicitly reference the destination.
 - The `<AccessPoint>` keyword specifies the name of the access point is *TDOM2*.
 - The `<Name>` keyword specifies the name of the queue is *REVERSE_STRING*.
- The `<replyQ>` keyword specifies the name of a JMS reply queue is *ReplyQ*.
- The `<translateFML>` keyword *FLAT* specifies that when a JMS message is received, the message is in XML format and is converted into the corresponding FML32 data buffer. The message is then placed in a `tpcall` with arguments *TDOM2* and *REVERSE_STRING*. The resulting message is then translated from FML into XML and placed on the *weblogic.jms.Tux2JmsQueue*.

Note: For more information on XML/FML conversion, see [Chapter 5, “Using FML with WebLogic Tuxedo Connector.”](#)

The following table provides information on the JMSQ2TuxX message mapping:

JMS Message Type	WebLogic Tuxedo Connector JATMI (Tuxedo)	JMS Message Type
BytesMessage	TypedCArray	BytesMessage
TextMessage (translateFML = NONE)	TypedString	TextMessage
TextMessage (translateFML = FLAT)	TypedFML32	TextMessage

Priority Mapping

Use `priorityMapping` to map priorities between the JMS and Tuxedo.

- JMS has ten priorities (0 - 9).
- Tuxedo/Q has 100 priorities (1 - 100).

This section provides a mechanism to map the priorities between the Tuxedo and JMS subsystems. There are two mapping directions:

- **JmstoTux**
- **TuxtoJms**

Defaults are provided for all values, shown below in pairs of `value:range`.

- The `value` specifies the given input priority.
- The `range` specifies a sequential group of resulting output priorities.

`JmstoTux- 0:1 | 1:12 | 2:23 | 3:34 | 4:45 | 5:56 | 6:67 | 7:78 | 8:89 | 9:100`

`TuxtoJms- 1-10:0 | 11-20:1 | 21-30:2 | 31-40:3 | 41-50:4 | 51-60:5 | 61-70:6 | 71-80:7 | 81-90:8 | 91-100:9`

Note: tBridge does not take the priority when setting JMS priority for the message using `setJMSPriority()`, set to QueueSender using `setPriority()`.

Example:

The following `priorityMapping` represents the default mapping:

```
<priorityMapping>
  <TuxtoJms>
    <pMap>
      <value>1-10</value>
      <range>0</range>
    </pMap>
    <pMap>
      <value>11-20</value>
```



```
        <range>1</range>
    </pMap>
    <pMap>
        <value>21-30</value>
        <range>2</range>
    </pMap>
    <pMap>
        <value>31-40</value>
        <range>3</range>
    </pMap>
    <pMap>
        <value>41-50</value>
        <range>4</range>
    </pMap>
    <pMap>
        <value>51-60</value>
        <range>5</range>
    </pMap>
    <pMap>
        <value>61-70</value>
        <range>6</range>
    </pMap>
    <pMap>
        <value>71-80</value>
        <range>7</range>
    </pMap>
    <pMap>
```

```
        <value>81-90</value>
        <range>8</range>
    </pMap>
    <pMap>
        <value>91-100</value>
        <range>9</range>
    </pMap>
</TuxtoJms>
<JmstoTux>
    <pMap>
        <value>0</value>
        <range>1</range>
    </pMap>
    <pMap>
        <value>1</value>
        <range>12</range>
    </pMap>
    <pMap>
        <value>2</value>
        <range>23</range>
    </pMap>
    <pMap>
        <value>3</value>
        <range>34</range>
    </pMap>
    <pMap>
        <value>4</value>
```

```
        <range>45</range>
    </pMap>
    <pMap>
        <value>5</value>
        <range>56</range>
    </pMap>
    <pMap>
        <value>6</value>
        <range>67</range>
    </pMap>
    <pMap>
        <value>7</value>
        <range>78</range>
    </pMap>
    <pMap>
        <value>8</value>
        <range>89</range>
    </pMap>
    <pMap>
        <value>9</value>
        <range>100</range>
    </pMap>
</JmstoTux>
</priorityMapping>
```

For this configuration, a JMS message of priority 7 is assigned a priority of 78 in the Tuxedo /Q. A Tuxedo /Q with a priority of 47 is assigned a JMS priority of 4.

Error Queues

When tBridge encounters a problem retrieving messages from Tuxedo Queue or JMS Queue after the retry interval:

- The information is logged.
- The message is saved in the error queue if it is configured.

wlsServerErrorDestination

The `wlsErrorDestination` queue is used if a JMS message cannot be properly delivered due to Tuxedo failure or a translation error.

Unsupported Message Types

If an unrecognized JMS message is received, an appropriate error message is logged and the message is discarded. This is considered a configuration error and the tBridge does not redirect the message to the error queue.

tuxErrorQueue

The `tuxErrorQueue` is the failure queue for the JATMI primitive `tpenqueue` during a JMS to Tuxedo /Q operation.

Limitations

The tBridge error queues have the following limitations:

- `TuxErrorDestination` can be specified only once. Any error queue name associated with the `ErrorDestination` implies that all the QSPACES have the same error queue name available.

- When there is an error, the message is put back in the source QSPACE. Assuming the QSPACE is corrupted or full, subsequent messages would be lost.
- There is no way to specify to drop messages on error. All messages are received or none are received.
- Information about the error is only available in the server log.

5 Using FML with WebLogic Tuxedo Connector

The following sections discuss the Field Manipulation Language (FML) and describe how the WebLogic Tuxedo Connector uses FML.

- [Overview of FML](#)
- [The WebLogic Tuxedo Connector FML API](#)
- [FML Field Table Administration](#)
- [tBridge XML/FML Translation](#)

Overview of FML

Note: For more information about using FML, see *Programming a BEA Tuxedo Application Using FML*.

FML is a set of java language functions for defining and manipulating storage structures called fielded buffers. Each fielded buffer contains attribute-value pairs in fields. For each field:

- The attribute is the field's identifier.

- The associated value represents the field's data content.
- An occurrence number.

There are two types of FML:

- FML16 based on 16-bit values for field lengths and identifiers. It is limited to 8191 unique fields, individual field lengths of 64K bytes, and a total fielded buffer size of 64K bytes.
- FML32 based on 32-bit values for the field lengths and identifiers. It allows for about 30 million fields, and field and buffer lengths of about 2 billion bytes.

The WebLogic Tuxedo Connector FML API

The FML application program interface (API) is provided by the WebLogic Tuxedo Connector `jatmi.jar` file. The details of the FML API implementation are documented in the `index.html` file included `doc` directory of your WebLogic Tuxedo Connector distribution.

Note: The WebLogic Tuxedo Connector implements a subset of FML functionality. For example, `views` are not supported.

FML Field Table Administration

Note: For more information on field tables, see the `index.html` file included `doc` directory of your WebLogic Tuxedo Connector distribution for references on the `FldTbl` and the `mkfldclass32` classes.

Field tables are generated in a manner similar to Tuxedo field tables. The field tables are text files that provide the field name definitions, field types, and identification numbers that are common between the two systems. To interoperate with a Tuxedo system using FML, the following steps are required:

1. Copy the field tables from the Tuxedo system to WebLogic Tuxedo Connector environment.

For example: Your Tuxedo distribution contains a bank application example called bankapp. It contains a file called bankflds that has the following structure:

```
#Copyright (c) 1990 Unix System Laboratories, Inc.
#All rights reserved
#ident  "@(#) apps/bankapp/bankflds      $Revision: 1.3 $"
# Fields for database bankdb

# name                number  type    flags  comments
ACCOUNT_ID            110     long    -      -
ACCT_TYPE              112     char    -      -
ADDRESS               109     string  -      -
.
.
.
```

2. Converted the field table definition into Java source files. Use the `mkfldclass` utility supplied with the WebLogic Tuxedo Connector jar file. This class is a utility function that reads a FML32 Field Table and produces a Java file which implements the `FldTbl` interface. There are two instances of this utility:

- `mkfldclass`
- and `mkfldclass32`

Use the correct instance of the command to convert the `bankflds` field table into FML32 java source. The following example uses `mkfldclass`.

```
java weblogic.wtc.jatmi.mkfldclass bankflds
```

The resulting file is called `bankflds.java` and has the following structure:

```
import java.io.*;
import java.lang.*;
import java.util.*;
import weblogic.wtc.jatmi.*;

public final class bankflds
    implements weblogic.wtc.jatmi.FldTbl
{
    /** number: 110  type: long */
    public final static int ACCOUNT_ID = 33554542;
    /** number: 112  type: char */
```

```
public final static int ACCT_TYPE = 67108976;  
/** number: 109  type: string */  
public final static int ADDRESS = 167772269;  
/** number: 117  type: float */
```

```
.  
.  
.
```

3. Compile the resulting `bankflds.java` file using the following command:

```
javac bankflds.java
```

The result is a `bankflds.class` file. When loaded, the WebLogic Tuxedo Connector uses the class file to add, retrieve and delete field entries from an FML32 field.

4. Add the field table class file to your application CLASSPATH.
5. Update the WebLogic Tuxedo Connector XML configuration file.
 - Update the `T_DM_RESOURCES` section to reflect the fully qualified location of the field table class file.
 - Use the keywords required to describe the FML buffer type: `fml16` or `fml32`.
 - You can enter multiple `<FldTblClass>` lines to specify additional field tables.

For example:

```
<T_DM_RESOURCES>  
  <FieldTables>  
    <FldTblClass Type="fml32">com.bea.mystuff.bankflds</FldTblClass>  
  </FieldTables>  
</T_DM_RESOURCES>
```

6. Restart your WebLogic Server to load the field table class definitions.

tBridge XML/FML Translation

The `<translateFML>` element is used to indicate if FML translation is performed on the message payload. There are two types of FML translation: `FLAT` and `NO`.

Note: The data type specified must be FLAT or NO. If any other data type is specified, the redirection fails.

FLAT

The message payload is translated using the WebLogic Tuxedo Connector internal FML/XML translator. Fields are converted field-by-field values without knowledge of the message structure (hierarchy) and repeated grouping.

In order to convert an FML buffer to XML, the tBridge pulls each instance of each field in the FML buffer, converts it to a string, and places it within a tag consisting of the field name. All of these fields are placed within a tag consisting of the service name. For example, an FML buffer consisting of the following fields:

```
NAME      JOE
ADDRESS   CENTRAL CITY
PRODUCTNAME BOLT
PRICE     1.95
PRODUCTNAME SCREW
PRICE     2.50
```

The resulting XML buffer would be:

```
<FML32>
  <NAME>JOE</NAME>
  <ADDRESS>CENTRAL CITY</ADDRESS>
  <PRODUCTNAME>BOLT</PRODUCTNAME>
  <PRODUCTNAME>SCREW</PRODUCTNAME>
  <PRICE>1.95</PRICE>
  <PRICE>2.50</PRICE>
</FML32>
```

NO

No translation is used. The tBridge maps a JMS TextMessage into a Tuxedo TypedBuffer (TypedString) and vice versa depending on the direction of the redirection. JMS BytesMessage are mapped into Tuxedo TypedBuffer (TypedCarray) and vice versa.

FML Considerations

Remember to consider the following information when working with FML:

- For XML input, the root element is required but ignored.
- For XML output, the root element is always <FML32>.
- The field table names must be loaded as described in [“FML Field Table Administration” on page 5-2](#).
- The tBridge translator is capable of only “flat” or linear grouping. This means that information describing FML ordering is not maintained, therefore buffers that contain a series of repeating data could be presented in an unexpected fashion. For example, consider a FML buffer that contains a list of parts and their associated price. The expectation would be PART A, PRICE A, PART B, PRICE B, etc. however since there is no structural group information contained within the tBridge, the resulting XML could be PART A, PART B, etc., PRICE A, PRICE B, etc.
- When translating XML into FML32, the translator ignores blank values. For example, <STRING></STRING> is skipped in the resulting FML buffer.
- Embedded FML is not supported in this release.
- TypedCArray is not supported for FML to XML conversion. Select from the following list of supported field types:
 - SHORT
 - LONG
 - CHAR
 - FLOAT
 - DOUBLE
 - STRING
 - INT (FML32)
 - DECIMAL (FML32)
- If you have TypedCArray in your FML, encode to TypedString and decode the XML to TypedCArray.

- If you need to pass binary data, encode to a field type of your choice and decode the XML on the receiving side.

6 Connecting WebLogic Process Integrator and Tuxedo Applications

The WebLogic Tuxedo Connector tBridge provides the necessary infrastructure for WebLogic Process Integrator users to integrate Tuxedo applications into their business workflows. The following sections discuss WebLogic Process Integrator - Tuxedo integration using the WebLogic Tuxedo Connector.

- [Synchronous WebLogic Process Integrator-to-Tuxedo Connectivity](#)
- [Synchronous Non-Blocking WebLogic Process Integrator-to-Tuxedo Connectivity](#)
- [Asynchronous WebLogic Process Integrator-to-Tuxedo Connectivity](#)
- [Asynchronous Tuxedo /Q-to-WebLogic Process Integrator Connectivity](#)
- [Bi-directional Asynchronous Tuxedo-to-WebLogic Process Integrator Connectivity](#)

Synchronous WebLogic Process Integrator-to-Tuxedo Connectivity

WebLogic Process Integrator executes a blocking invocation against a Tuxedo service using a JATMI EJB. This process consists of three parts:

- Defining WebLogic Process Integrator Business Operations.
- Invoking an eLink Adapter.
- Defining WebLogic Process Integrator Exception Handlers.

Defining Business Operations

Define WebLogic Process Integrator Business Operations for the JATMI methods to be used:

- TypedFML32 buffer manipulation methods.
- Use the JATMI `tpcall()` method.

Example: `out_buffer = tpcall(service_name, in_buffer, flags)`

Invoking an eLink Adapter

Invoke an eLink adapter from a WebLogic Process Integrator process flow:

- Build TypedFML32 request buffers using defined Business Operations.
- Using the defined Business Operation invoke the JATMI `tpcall()` method specifying the service name.
- Process TypedFML32 response buffers using defined Business Operations.

Define Exception handlers

Define WebLogic Process Integrator Exception handlers to process exceptions.

Synchronous Non-Blocking WebLogic Process Integrator-to-Tuxedo Connectivity

WebLogic Process Integrator sends a message to synchronously invoke a Tuxedo service:

- 1:1 relationship between JMS queue and the call to a Tuxedo service.
- 1:1 relationship between the response from the Tuxedo service and a JMS queue.
- WebLogic Process Integrator writes a message to JMS queue.
- Once the message is on the JMS queue then tBridge moves the message to the target Tuxedo service.
- The message is translated from/to XML/FML32.
- The response is written to the specified JMS reply queue.
- The WebLogic Process Integrator event node waits on the response queue for a response message.

Asynchronous WebLogic Process Integrator-to-Tuxedo Connectivity

WebLogic Process Integrator sends a guaranteed asynchronous message to a Tuxedo /Q:

- 1:1 relationship between JMS queue and Tuxedo /Q.
- WebLogic Process Integrator writes a message to JMS queue.
- Once the message is on the JMS queue then tBridge moves the message to the target Tuxedo /Q on a per message basis.

- Messages in error are forwarded to a specified JMS error queue:
 - Infrastructure errors.
 - XML/FML32 translation errors.

Asynchronous Tuxedo /Q-to-WebLogic Process Integrator Connectivity

Tuxedo /Q sends a guaranteed asynchronous message to WebLogic Process Integrator:

- 1:1 relationship between JMS queue and Tuxedo /Q.
- Tuxedo writes a message to Tuxedo /Q.
- Once the message is committed on Tuxedo /Q, the message is forwarded via the Tuxedo /T Domain Gateway to the Weblogic Tuxedo Connector tBridge and target JMS queue.

Note: This cannot be performed transactionally in WebLogic Server 6.0.

- Messages which cannot be forwarded from Tuxedo are enqueued on a Tuxedo /Q error queue.
- Messages in error are forwarded to a specified Tuxedo /Q error queue, including:
 - Infrastructure errors.
 - FML32/XML translation errors.
- A workflow is created that waits for the message on the JMS queue. It is defined in the Start workflow node or in the Event node of an existing workflow instance.

Bi-directional Asynchronous Tuxedo-to-WebLogic Process Integrator Connectivity

Tuxedo executes a blocking invocation of a WebLogic Process Integrator process flow. Use two asynchronous instances to connect from JMS to Tuxedo /Q and from Tuxedo /Q back to JMS.

7 Troubleshooting The WebLogic Tuxedo Connector

The following sections provide WebLogic Tuxedo Connector troubleshooting information.

- [Monitoring the WebLogic Tuxedo Connector](#)
- [Frequently Asked Questions](#)

Monitoring the WebLogic Tuxedo Connector

The WebLogic Tuxedo Connector uses the WebLogic Server log file to record log information.

Setting Trace Levels

Use the `TraceLevel` parameter to set the tracing level of the WebLogic Tuxedo Connector on a server node. To enable tracing, add the keyword `TraceLevel` and with the desired trace value in the `Arguments` field of the startup class window. Use a comma to separate arguments in the `Arguments` field.

Example:

```
Arguments=BDMCONFIG=.\mydomain\wtc_config.xml,TraceLevel=100000
```

Use the following values to set the TraceLevel:

Value	Components Traced	Description
10000	TBRIDGE_IO	tBridge input and output
15000	TBRIDGE_EX	more tBridge information
20000	GWT_IO	Gateway input and output, including the ATMI verbs
25000	GWT_EX	more Gateway information
50000	JAMTI_IO	JAMTI input and output, including low-level JAMTI calls
55000	JAMTI_EX	more JAMTI information
100000	All Components	information on all WebLogic Tuxedo Connector components

Console Settings

To ensure that the all the necessary trace information is written to the log file, verify that server logging settings specify the following:

- Debug to Stdout is selected
- Stdout severity threshold is set to Info



Frequently Asked Questions

This section provides solutions to common user questions.

Cannot Find Configuration File

The error log displays the message **Can not find the XML configuration file**. What am I doing wrong?

- Make sure that the Startup class is configured properly. See [“Create a StartUp Class” on page 2-12](#).
- Check the WebLogic Server configuration file, `config.xml`. Check the Startup Class entry and edit if necessary. An example of a StartUp class follows:

```
<StartupClass
```

```
Arguments="BDMCONFIG=. \mydomain\dmconfig.xml,TraceLevel=100000"
```

```
ClassName="weblogic.wtc.gwt.WTCStartup" FailureIsFatal="true"
Name="MyWtcStartup Class" Targets="myserver"/>
```

EJB Deployment Message

When I build the `simpserve` example, I get the following error:

```
<date> <Error> <EJB> <EJB Deployment: Tolower has a class
weblogic.wtc.jatmi.tpserviceHome which is in the classpath. This
class should only be located in the ejb-jar file.>
```

This error message can be ignored for this release of the WebLogic Tuxedo Connector. The EJB wants all of the interfaces for an EJB call in the EJB jar file. However, some interfaces for the WebLogic Tuxedo Connector are implemented through the CLASSPATH, and the compiler throws an exception. When the EJB is deployed, the compiler complains that the EJB cannot be redeployed because some of its classes are found in the CLASSPATH.

Connection Problems

I'm having trouble getting a connection established between WebLogic Tuxedo Connector and Tuxedo. What should I do?

- Check the WebLogic Tuxedo Connector configuration against the Tuxedo remote domain. The remote domain must match the name of a remote domain configured in WebLogic Tuxedo Connector.

For example: If the name `simpapp` is configured in the Tuxedo DMCONFIG `*DM_LOCAL_DOMAINS` section, then this name must match the name in the WebLogic Tuxedo Connector configuration file `<AccessPointId>` field.

- Check the WebLogic Tuxedo Connector and Tuxedo log files for error messages.
- Enable the WebLogic Tuxedo Connector trace and repeat the connectivity test.
- Request assistance from BEA Customer Support.

8 The WebLogic Tuxedo Connector XML Configuration File

The following sections describe how to create a WebLogic Tuxedo Connector XML configuration file and provide a hierarchy diagram of elements.

- [Creating an XML Configuration File](#)
- [Element Hierarchy Diagram](#)

Creating an XML Configuration File

You create the WebLogic Tuxedo Connector XML configuration file using a text editor. The most efficient method to create a new configuration file is to modify one of the example `DBMCONFIG.xml` files to meet your application needs.

Sample XML Configuration File

This section provides an example WebLogic Tuxedo Connector XML configuration file for the `simpapp` example application.

```
<?xml version="1.0"?>
```

8 *The WebLogic Tuxedo Connector XML Configuration File*

```
<!DOCTYPE BDMCONFIG SYSTEM
"file:[WTC installation directory]\weblogic\wtc\gwt\wtc_config_1_0.dtd">

<!--Java and XML-->

<WTC_CONFIG>

<    BDMCONFIG>

<    T_DM_LOCAL_TDOMAIN AccessPoint="TDOM2">

        <WlsClusterName>Coolio</WlsClusterName>

        <AccessPointId>TDOM2</AccessPointId>

        <Type>TDOMAIN</Type>

        <Security>NONE</Security>

        <ConnectionPolicy>ON_DEMAND</ConnectionPolicy>

        <BlockTime>30</BlockTime>

        <NWAddr>[Network address of WTC domain]</NWAddr>

        <!-- Example address: //mydomain.acme.com:20304 -->

    </T_DM_LOCAL_TDOMAIN>

    <T_DM_REMOTE_TDOMAIN AccessPoint="TDOM1">

        <LocalAccessPoint>TDOM2</LocalAccessPoint>

        <AccessPointId>TDOM1</AccessPointId>

        <Type>TDOMAIN</Type>

        <NWAddr>[Network address of Tuxedo domain]</NWAddr>

    <    !-- Example address: //mydomain.acme.com:20305 -->

    </T_DM_REMOTE_TDOMAIN>

    <T_DM_EXPORT ResourceName="TOLOWER"

        LocalAccessPoint="TDOM2">

        <EJBName>tuxedo.services.TOLOWERHome</EJBName>

    </T_DM_EXPORT>

    <T_DM_IMPORT

        ResourceName="TOUPPER"
```

```
LocalAccessPoint="TDOM2"

RemoteAccessPointList="TDOM1">

<TranTime>600</TranTime>

</T_DM_IMPORT>

</BDMCONFIG>

</WTC_CONFIG>
```

Note: You must modify the DOCTYPE with the name of your configuration file and specify the fully qualified path to its location.

The DOCTYPE declaration provides the location of the `wtc_config.dtd`. When the WebLogic Server is started, the WebLogic Tuxedo Connector XML configuration file is checked against the document type definition (DTD) for errors.

Validate the XML file

Validate your configuration file using `WTCValidateCF`. This utility allows you to validate the WebLogic Tuxedo Connector XML configuration file before booting WebLogic Server.

To validate the XML configuration file, enter the following command:

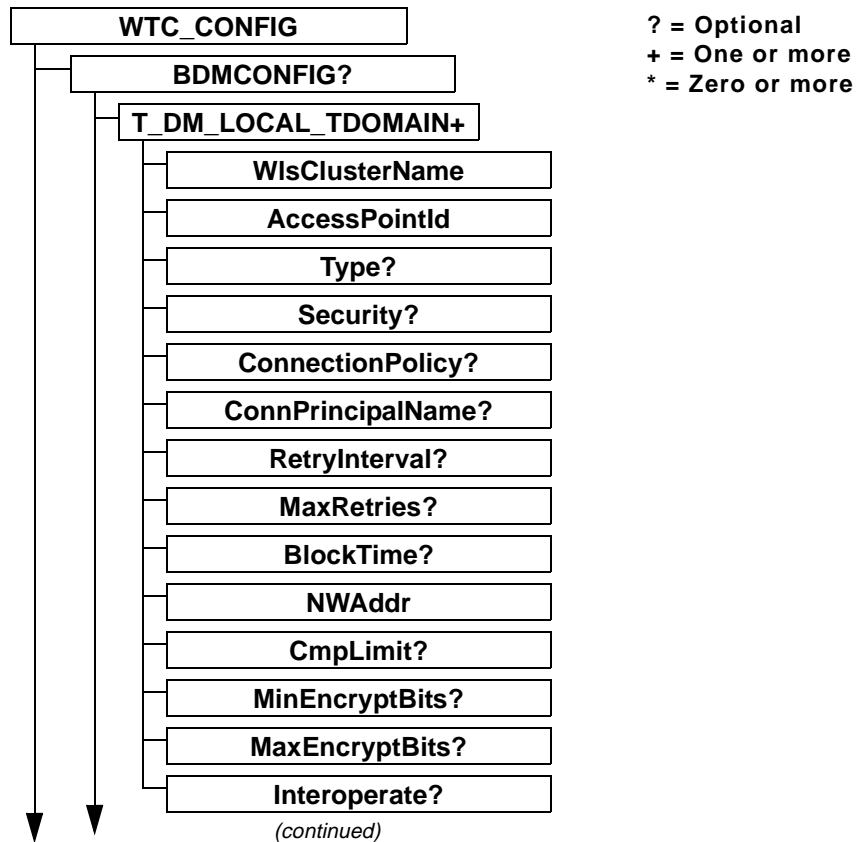
```
> java weblogic.wtc.gwt.WTCValidateCF your_XML_configuration_file
```

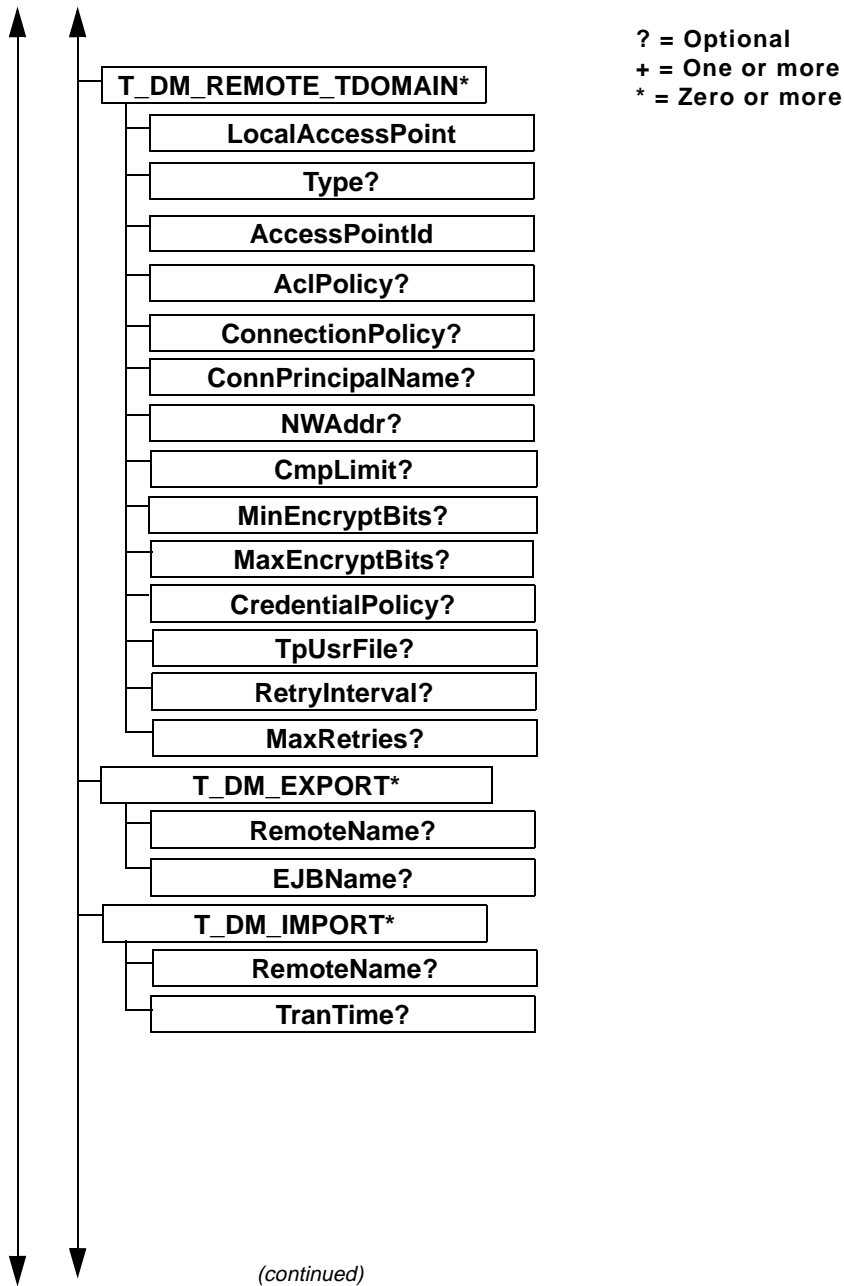
where *your_XML_configuration_file* is the name of your XML configuration file.

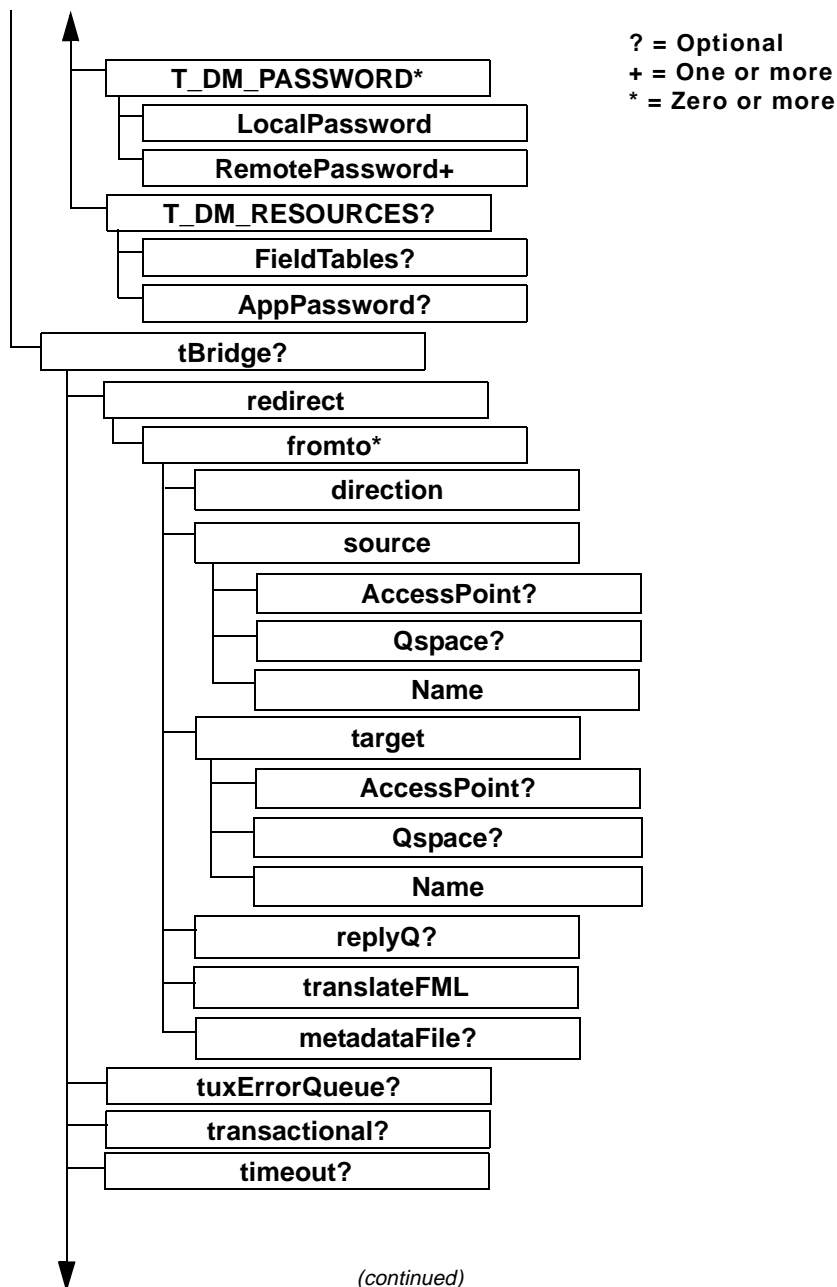
Element Hierarchy Diagram

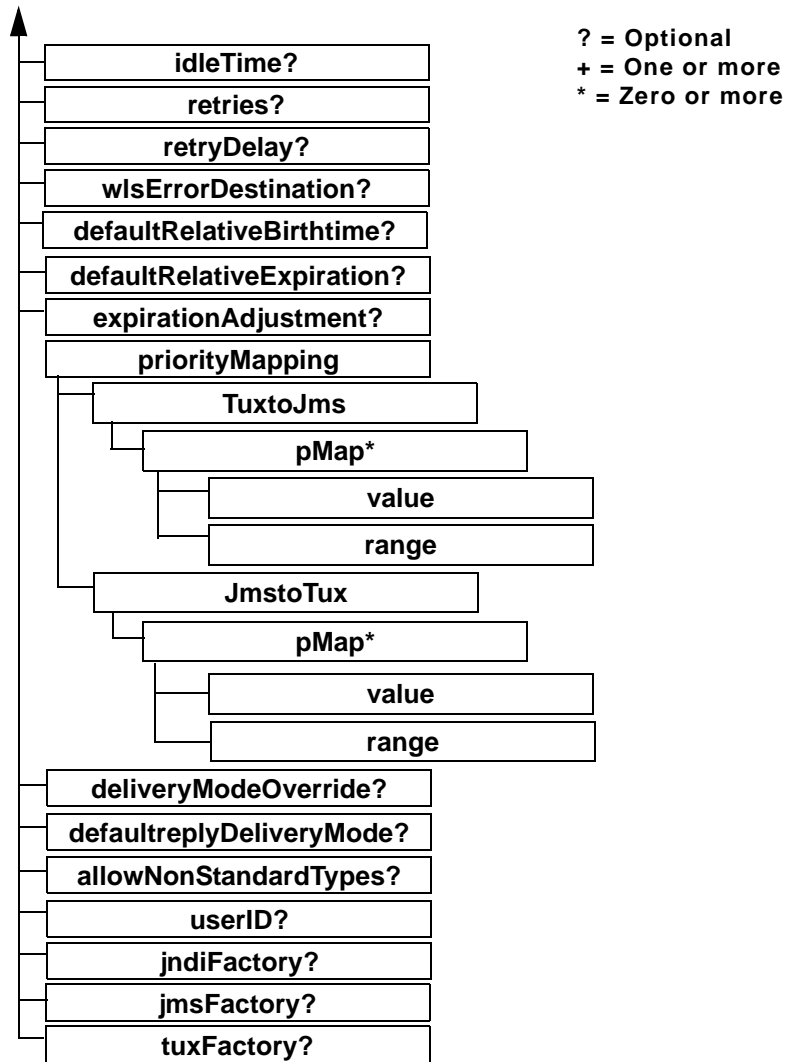
A WebLogic Tuxedo Connector configuration file consists of a series of XML elements. The `WTC_CONFIG` element is the top-level element, and all elements in the `WTC_CONFIG` are children of the `WTC_CONFIG` element. Child elements may have

children themselves. For example, the `BDMCONFIG` element contains the `T_DM_LOCAL_TDOMAIN` child element, and the `T_DM_LOCAL_TDOMAIN` element contains child elements, as shown in the following diagram.









9 The wtc_config.dtd

The `wtc_config.dtd` file is used to define the elements and attributes used in the WebLogic Tuxedo Connector XML configuration file.

wtc_config.dtd

```
<!ELEMENT WTC_CONFIG (  
    BDMCONFIG?,  
    tBridge?)>  
  
<!ELEMENT BDMCONFIG (  
    T_DM_LOCAL_TDOMAIN+,  
    T_DM_REMOTE_TDOMAIN*,  
    T_DM_EXPORT*,  
    T_DM_IMPORT*,  
    T_DM_PASSWORD*,  
    T_DM_RESOURCES?)>  
  
<!ELEMENT T_DM_LOCAL_TDOMAIN (  
    WlsClusterName,  
    AccessPointId,  
    Type?,  
    Security?,
```

```

        ConnectionPolicy?,
        ConnPrincipalName?,
        RetryInterval?
        MaxRetries?,
        BlockTime?,
        NWAddr,
        CmpLimit?,
        MinEncryptBits?,
        MaxEncryptBits?,
        Interoperate?)>
<!ATTLIST T_DM_LOCAL_TDOMAIN
        AccessPoint CDATA #REQUIRED>
<!ELEMENT WlsClusterName (#PCDATA)>
<!ELEMENT AccessPointId (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Security (#PCDATA)>
<!ELEMENT ConnectionPolicy (#PCDATA)>
<!ELEMENT RetryInterval (#PCDATA)>
<!ELEMENT MaxRetries (#PCDATA)>
<!ELEMENT ConnPrincipalName (#PCDATA)>
<!ELEMENT NWAddr (#PCDATA)>
<!ELEMENT CmpLimit (#PCDATA)>
<!ELEMENT MinEncryptBits (#PCDATA)>
<!ELEMENT MaxEncryptBits (#PCDATA)>
<!ELEMENT Interoperate (#PCDATA)>
<!ELEMENT BlockTime (#PCDATA)>
<!ELEMENT T_DM_REMOTE_TDOMAIN (

```

```
        LocalAccessPoint,
        AccessPointId,
        Type?,
        AclPolicy?,
        ConnectionPolicy?,
        ConnPrincipalName?,
        NWAddr,
        CmpLimit?,
        MinEncryptBits?,
        MaxEncryptBits?,
        CredentialPolicy?,
        TpUsrFile?
        RetryInterval?
        MaxRetries?})>
<!--ATTLIST T_DM_REMOTE_TDOMAIN
        AccessPoint CDATA #REQUIRED>
<!--ELEMENT LocalAccessPoint (#PCDATA)>
<!--ELEMENT AclPolicy (#PCDATA)>
<!--ELEMENT CredentialPolicy (#PCDATA)>
<!--ELEMENT TpUsrFile (#PCDATA)>
<!--ELEMENT T_DM_EXPORT (RemoteName?, EJbName?)>
<!--ATTLIST T_DM_EXPORT
        ResourceName      CDATA #REQUIRED
        LocalAccessPoint CDATA #REQUIRED>
<!--ELEMENT RemoteName (#PCDATA)>
<!--ELEMENT EJbName (#PCDATA)>
<!--ELEMENT T_DM_IMPORT (RemoteName?, TranTime?)>
```

```

<!--ATTLIST T_DM_IMPORT
                ResourceName          CDATA #REQUIRED
                LocalAccessPoint       CDATA #REQUIRED
                RemoteAccessPointList  CDATA #REQUIRED>

<!--ELEMENT TranTime (#PCDATA)>

<!--ELEMENT T_DM_PASSWORD (LocalPassword, RemotePassword+)>

<!--ATTLIST T_DM_PASSWORD
                LocalAccessPoint       CDATA #REQUIRED
                RemoteAccessPointList  CDATA #REQUIRED>

<!--ELEMENT LocalPassword (#PCDATA)>

<!--ATTLIST LocalPassword
                IV CDATA #REQUIRED>

<!--ELEMENT RemotePassword (#PCDATA)>

<!--ATTLIST RemotePassword
                IV CDATA #REQUIRED>

<!--ELEMENT T_DM_RESOURCES (
                FieldTables?,
                AppPassword?)>

<!--ELEMENT FieldTables (FldTblClass+)>

<!--ELEMENT FldTblClass (#PCDATA)>

<!--ATTLIST FldTblClass
                Type (fml16 | fml32) #REQUIRED>

<!--ELEMENT AppPassword (#PCDATA)>

<!--ATTLIST AppPassword
                IV CDATA #REQUIRED>

<!--ELEMENT tBridge (
                redirect,

```

```
        transactional?,
        timeout?,
        idleTime?,
        retries?,
        retryDelay?,
        wlsErrorDestination?,
        tuxErrorQueue?,
        defaultRelativeBirthtime?,
        defaultRelativeExpiration?,
        expirationAdjustment?,
        priorityMapping?,
        deliveryModeOverride?,
        defaultreplyDeliveryMode?,
        userID?,
        allowNonStandardTypes?,
        jndiFactory,
        jmsFactory,
        tuxFactory)>
<!--ELEMENT direction (#PCDATA)>
<!--ELEMENT translateFML (#PCDATA)>
<!--ELEMENT metadataFile (#PCDATA)>
<!--ELEMENT AccessPoint (#PCDATA)>
<!--ELEMENT Qspace (#PCDATA)>
<!--ELEMENT Name (#PCDATA)>
<!--ELEMENT ReplyQ (#PCDATA)>
<!--ELEMENT source (
        AccessPoint?,
```

```

        Qspace?,
        Name )>
<!ELEMENT target (
        AccessPoint?,
        Qspace?,
        Name )>
<!ELEMENT fromto (
        direction,
        source,
        target,
        ReplyQ?,
        translateFML,
        metadataFile? )>
<!ELEMENT redirect (fromto*)>
<!ELEMENT transactional (#PCDATA)>
<!ELEMENT timeout (#PCDATA)>
<!ELEMENT idleTime (#PCDATA)>
<!ELEMENT retries (#PCDATA)>
<!ELEMENT retryDelay (#PCDATA)>
<!ELEMENT wlsErrorDestination (#PCDATA)>
<!ELEMENT tuxErrorQueue (#PCDATA)>
<!ELEMENT defaultRelativeBirthtime (#PCDATA)>
<!ELEMENT defaultRelativeExpiration (#PCDATA)>
<!ELEMENT expirationAdjustment (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT range (#PCDATA)>
<!ELEMENT pMap (value,range)>

```

```
<!ELEMENT TuxtoJms (pMap*)>
<!ELEMENT JmstoTux (pMap*)>
<!ELEMENT priorityMapping (
    TuxtoJms ,
    JmstoTux)>
<!ELEMENT deliveryModeOverride (#PCDATA)>
<!ELEMENT defaultreplyDeliveryMode (#PCDATA)>
<!ELEMENT userID (#PCDATA)>
<!ELEMENT allowNonStandardTypes (#PCDATA)>
<!ELEMENT jndiFactory (#PCDATA)>
<!ELEMENT jmsFactory (#PCDATA)>
<!ELEMENT tuxFactory (#PCDATA)>
```


10 Elements and Attributes of the `wtc_config.dtd`

The following sections contain reference information on the elements and attributes contained in the `wtc_config.dtd`:

- [WTC_CONFIG](#)
- [BDMCONFIG](#)
- [tBridge](#)

WTC_CONFIG

`WTC_CONFIG` is the root of the WebLogic Tuxedo Connector deployment descriptor. `WTC_CONFIG` has two children:

- [BDMCONFIG](#)
- [tBridge](#)

BDMCONFIG

The BDMCONFIG element provides information on how to configure domains in the WebLogic Tuxedo Connector. The configuration parameters are analogous to the DM_MIB attributes and classes used for Tuxedo domains.

T_DM_LOCAL_TDOMAIN

The T_DM_LOCAL_DOMAIN provides a view of local domains as they appears to other domains.

Attribute	Description	Type	Use
AccessPoint	Label used to identify a domain in a configuration XML file. This label must be unique within the scope of T_DM_LOCAL_TDOMAIN and T_DM_REMOTE_TDOMAIN AccessPoint names in a configuration XML file. Example: TDOM2	CDATA	#REQUIRED

WlsClusterName

Note: Although WebLogic Tuxedo Connector does not support clustering in this release of WebLogic Server, you must provide a WlsClusterName.

Required. WlsClusterName gives the name of the WebLogic Server cluster in the WSL domain on which this local access point resides.

Example: cluster20

AccessPointId

Required. Specifies the connection principal name used to identify a domain when establishing a connection to another domain.

The AccessPointId of a T_DM_LOCAL_TDOMAIN must match the corresponding DOMAINID in the *DM_REMOTE_DOMAINS section of your Tuxedo DMCONFIG file.

The AccessPointId of a T_DM_REMOTE_TDOMAIN must match the corresponding DOMAINID in the *DM_LOCAL_DOMAINS section of your Tuxedo DMCONFIG file.

Example: TDOM2

Type

Optional. If specified the value must be the string TDOMAIN.

Security

Optional. Security specifies the type of application security to be enforced. Valid values for this parameter are: NONE, APP_PW, or DM_PW.

- NONE: No security is used. This is the default value.
- APP_PW: Password security is enforced when a connection is established from a remote domain. The application password must be defined in a T_DM_PASSWORD element.
- DM_PW: Domain password security is enforced when a connection is established from a remote domain. Domain passwords must be defined in the T_DM_PASSWORD element.

ConnectionPolicy

Optional. Specifies the conditions under which a local domain tries to establish a connection to a remote domain.

- Valid values for local domains are: ON_DEMAND, ON_STARTUP, or INCOMING_ONLY.
- Valid values for remote domains are: ON_DEMAND, ON_STARTUP, INCOMING_ONLY, or LOCAL.
- Default setting is ON_DEMAND
 - ON_DEMAND: A connection is attempted only when requested by either a client request to a remote service or an administrative connect command.
 - ON_STARTUP: A domain gateway attempts to establish a connection with its remote domain access points at gateway server initialization time. Remote

services (services advertised in JNDI by the domain gateway for this local access point) are advertised only if a connection is successfully established to that remote domain access point. If there is no active connection to a remote domain access point, then the remote services are suspended. By default, this connection policy retries failed connections every 60 seconds. Use the `MaxRetry` and `RetryInterval` elements to specify application specific values.

- **INCOMING_ONLY**: A domain gateway does not attempt an initial connection to remote domain access points at startup and remote services are initially suspended. The domain gateway is available for incoming connections from remote domain access points and remote services are advertised when the domain gateway for this local domain access point receives an incoming connection. Connection retry processing is not allowed
- **LOCAL**: Indicates the remote domain connection policy is explicitly defaulted to the local domain `ConnectionPolicy` attribute value. If the remote domain `ConnectionPolicy` is not defined, the system uses the setting specified by the associated local domain (specified by the `LocalAccessPoint`).

RetryInterval

Optional. The time (seconds) between automatic attempts to establish a connection to remote domain access points. Use only when `ConnectionPolicy` is set to `ON_STARTUP`.

- Minimum value: 0
- Maximum value: 2147483647
- Default setting: 60

MaxRetries

Optional. The number of times that a domain gateway tries to establish connections to remote domain access points. Use only when `ConnectionPolicy` is set to `ON_STARTUP`.

- Minimum value: 0
- Maximum value: 2147483647.
- Default value: 2147483647

Use the maximum value to retry processing until a connection is established. Use the minimum value to disable the automatic retry mechanism.

ConnPrincipalName

Note: ConnPrincipalName is not supported in this release of WebLogic Server.

Optional. Specifies the connection principal name identifier. This is the principal name for identifying a domain when establishing a connection to another domain.

- If this element is not specified, the connection principal name defaults to the AccessPointId element for this domain.

NWAddr

Note: When configuring the NWAddr for a T_DM_LOCAL_DOMAIN, the port number used should be different from any port numbers assigned to other WebLogic Server processes. Example: Setting the NWAddr to `//myachine:7001` is not valid if the WebLogic Server listening port is assigned to `//myachine:7001`.

Required. The network address of the local domain gateway. Specify the TCP/IP address in one of the following formats:

- `//hostname:port_number`
- `//#. #. #. #:port_number`

If hostname is used, the domain finds an address for hostname using the local name resolution facilities (usually DNS). If dotted decimal format is used, each # should be a number from 0 to 255. This dotted decimal number represents the IP address of the local machine. The port_number is the TCP port number at which the domain process listens for incoming requests.

CmpLimit

Optional. Specifies the compression threshold used when sending data to the remote domain. Application buffers larger than this size are compressed.

- Default value: 2147483647 bytes

MinEncryptBits

Optional. Specifies the minimum level encryption key length (in bits) used when establishing a network link for this domain. Valid values for this parameter are: 0, 56, and 128.

- Default value: 0 bits
- A value of 0: No encryption used
- If this minimum level of encryption cannot be met, the network link fails.

MaxEncryptBits

Optional. Specifies the maximum level encryption key length (in bits) used when establishing a network link for this domain. Valid values for this parameter are: 0, 56, and 128.

- Default value: 128 bits
- A value of 0: No encryption used

Interoperate

Optional. Specifies whether the local domain interoperates with remote domains that are based upon Tuxedo Release 6.5. Valid values for this parameter are: Yes, No.

- Yes: Interoperate with Tuxedo 6.5
- No: Operates with domains other than Tuxedo 6.5.
- Default value: No

BlockTime

Optional. Specifies the maximum wait time (seconds) allowed for a blocking call.

T_DM_REMOTE_TDOMAIN

The T_DM_REMOTE_TDOMAIN provides a view of remote domains as they appear to the local domain.

Attribute	Description	Type	Use
AccessPoint	Label used to identify a domain in a configuration XML file. This label must be unique within the scope of T_DM_LOCAL_TDOMAIN and T_DM_REMOTE_TDOMAIN AccessPoint names in a configuration XML file. Example: TDOM3	CDATA	#REQUIRED

LocalAccessPoint

Required. The local domain name from which a remote domain is reached.

Example: TDOM2

AclPolicy

Note: If the Interperate parameter is set to Yes, the AclPolicy is ignored. For more information see, [“Interoperate” on page 10-6](#).

Optional. Specifies the inbound access control list (ACL) policy toward requests from a remote domain. Valid values for this parameter are: LOCAL, GLOBAL.

- LOCAL: The local domain modifies the identity of the service requests received from a given remote domain to the principal name specified in the local principal name for a given remote domain.
- GLOBAL: The local domain passes the service request with no change in identity.
- Default value: LOCAL

CredentialPolicy

Note: If the Interperate parameter is set to Yes, the CredentialPolicy is ignored. For more information see, [“Interoperate” on page 10-6](#).

Optional. Specifies the outbound access control list (ACL) policy toward requests to a remote domain. Valid values for this parameter are: LOCAL, GLOBAL.

- **LOCAL:** The remote domain controls the identity of service requests received from the local domain to the principal name specified in the local principal name for this remote domain.
- **GLOBAL:** The remote domain passes the service request with no change.
- **Default value:** LOCAL

TpUsrFile

Optional. Full path to user password file containing uid/gid information. This is the same file generated by the Tuxedo `tpusradd` utility on the remote domain. Username, uid and gid information must be included and valid for correct authorization, authentication, and auditing.

T_DM_EXPORT

T_DM_EXPORT provides information on services exported by a local domain.

- If not specified, all local domains accept requests to all of the services according to the default JNDI lookup rules (see [EJBName](#)).
- If the section is defined, use it to restrict the set of local services requested from a remote domain.

Attribute	Description	Type	Use
ResourceName	The ResourceName attribute describes a exported service entry.	CDATA	#REQUIRED
LocalAccessPoint	The local access point name.	CDATA	#REQUIRED

RemoteName

Optional. The remote name of the service.

- If not specified, the ResourceName attribute is used.

EJBName

Optional. The complete name of the EJB home interface to use when invoking a service.

- If this element is not specified, the default interface used is `tuxedo.services.servicenameHome`.

Example: If the service being invoked is TOUPPER and the EJBName attribute is not specified, the home interface looked up in JNDI would be `tuxedo.service.TOUPPERHome`.

T_DM_IMPORT

The T_DM_IMPORT provides information on services imported and available on remote domains. If T_DM_IMPORT is not configured, remote domains handle all remote services.

Attribute	Description	Type	Use
ResourceName	The ResourceName attribute describes an imported service entry. The combination of the ResourceName, LocalAccessPoint and RemoteAccessPointList attributes must be unique among all objects of this type Example: QSPACE	CDATA	#REQUIRED
LocalAccessPoint	Specifies the local access point through which a service is offered. Example: TDOM2	CDATA	#REQUIRED
RemoteAccessPointList	A comma-separated failover list that identifies the remote domain access points through which a resource is imported. Example: TDOM3,TDOM4,TDOM5	CDATA	#REQUIRED

TranTime

Note: TranTime is not supported for this release of WebLogic Tuxedo Connector.

Optional. Specifies the default timeout value (seconds) for a transaction started in an associated service. 0 implies the maximum timeout value.

- Minimum value: 0
- Maximum value: 2147483648
- Default value: 30

T_DM_PASSWORD

The T_DM_PASSWORD class represents configuration information for inter-domain authentication through access points of type TDOMAIN.

Attribute	Description	Type	Use
LocalAccessPoint	The name of the local domain access point to which the password applies. Example: TDOM2	CDATA	#REQUIRED
RemoteAccessPoint	The name of the remote domain access point to which the password applies. Example: TDOM3	CDATA	#REQUIRED

LocalPassword

Required. The encrypted local password as returned from the `genpasswd` utility. This password is used to authenticate connections between the local domain access point identified by LocalAccessPoint and the remote domain access point identified by RemoteAccessPoint

Attribute	Description	Type	Use
IV	The initialization vector used to encrypt the local password	CDATA	#REQUIRED

RemotePassword

Required. The encrypted remote password as returned from the `genpasswd` utility. This password is used to authenticate connections between the local domain access point identified by `LocalAccessPoint` and the remote domain access point identified by `RemoteAccessPoint`.

Attribute	Description	Type	Use
IV	The initialization vector used to encrypt the remote password	CDATA	#REQUIRED

T_DM_RESOURCES

Use to specify global field table classes and application passwords for domains.

FieldTables

Optional. Identifies a list of `FldTbls` tables available to WebLogic Tuxedo Connector users. These tables are created via the `mkfldclass` or `mkfldclass32` utilities and compiled into java classes. The array is used for FML and FML32 operations and is accessible through the `getFldTbls(fml16|fml32)` utility.

FldTblClass

Required. The name of the field table class loaded via a class loader and added to a `FldTbl` array. The class name used is the fully qualified name of the desired class.

Attribute	Description	Use
Type	Flag indicating what class type is used for FML / FML32 operations. Valid values are: <ul style="list-style-type: none">■ fml16■ fml32	#REQUIRED

AppPassword

Optional. The encrypted application password as returned from the `genpasswd` utility. This global password is used to authenticate connections between all domain access points.

Attribute	Description	Type	Use
IV	The initialization vector used to encrypt the global password	CDATA	#REQUIRED

tBridge

Note: The tBridge handles one or more redirections by starting a new thread for each redirection defined. At least one redirection must be specified or the tBridge fails and an error is logged.

Configuring the optional tBridge provides bi-directional transport of XML messages between WebLogic Server and Tuxedo.

direction

Required. Specifies the direction of data flow. Valid parameter values are: JmsQ2TuxQ, TuxQ2JmsQ, JmsQ2TuxS.

- JmsQ2TuxQ: From JMS to TUXEDO /Q
- TuxQ2JmsQ: From TUXEDO /Q to JMS
- JmsQ2TuxS: From JMS to TUXEDO Service reply to JMS

fromto

Required. Element used to specify the source, target, direction, and transport of a message.

redirect

Required. Element used to redirect a message.

source

Required. The source location of a message.

target

Required. The target location to place a message

AccessPoint

Optional. An identifier unique within the scope of T_DM_LOCAL_TDOMAIN and T_DM_REMOTE_TDOMAIN entry names in the domain configuration.

Example: TDOM2

Qspace

Optional. Name of the Qspace for a source or target location.

Name

Required. Name of a JMS queue name, TUXEDO queue name, or a TUXEDO service name.

ReplyQ

Optional. Name of a JMS queue specifically for the synchronous call to a TUXEDO service. The response is returned to the JMS ReplyQ.

metadataFile

Note: This element is not supported in this release of WebLogic Tuxedo Connector.

Optional. URL of the metadata file.

translateFML

Note: The WLXT parameter value is not supported in this release of WebLogic Tuxedo Connector.

Required. Specifies the type of XML/FML translation. Valid parameter values are: NONE, FLAT, WLXT.

- NO: No data translation is performed. `TextMessage` maps into `STRING` and vice versa depending on the direction of transfer. `BytesMessage` maps into `CARRAY` and vice versa. All other data types cause the redirection to fail.
- FLAT: The message payload is transformed using the WebLogic Tuxedo Connector built-in translator.
- WLXT: The translation is done by the XML-to-nonXML WL XML Translator (WLXT). The `metadataFile` URL provided is passed to call the WLXT external methods to do the translation.
- Default value: NO

transactional

Note: This element is not supported in this release of WebLogic Tuxedo Connector.

Optional. Specifies the use of transactions when retrieving messages from a source location and when placing messages on a target location. Valid parameter values are: YES, NO.

- YES: Transactions are used for both operations
- NO: Transactions are not used for either operation.

timeout

Optional. Specifies the transaction timeout value (seconds) used to place a message on a target location. This parameter reflects the effective length of the timeout for the entire redirection. Required when `transactional` parameter is set to YES.

- Default value is 60
- The value must be 0 or a positive integer

-
- 0 indicates an infinite wait

idleTime

Note: This element is not supported in this release of WebLogic Tuxedo Connector.

Optional. Specifies the amount of idle time (seconds) to wait before checking the source location for new messages. If a message is detected, all the messages on the source location are processed sequentially.

- Default value is 0
- The value must be 0 or a positive integer

retries

Optional. Specifies the number of attempts to redirect a message before putting the message in the specified error location and logging an error.

- Default value is 0
- The value must be 0 or a positive integer

retryDelay

Note: During the `retryDelay`, the thread processing the message can not redirect any other messages.

Optional. Specifies the minimum amount of time (milliseconds) to wait before redirecting a message.

- Default value is 10
- The value a positive integer

wlsErrorDestination

Optional. Name of the location used to store WebLogic Server JMS messages when a message cannot be redirected.

- If a `wlsErrorDestination` is not specified, all messages that cannot be redirected are lost.
- If the message cannot be placed into the `wlsErrorDestination` for any reason, an error is logged and the message is lost.

tuxErrorQueue

Optional. Name of the Tuxedo queue used to store a message that cannot be redirected to a Tuxedo/Q source queue. This queue is in the same queue space as the source queue.

- If `tuxErrorDestination` is not specified, all messages that cannot be redirected are lost.
- If the message cannot be placed into the `tuxErrorQueue` for any reason, an error is logged and the message is lost.

defaultRelativeBirthtime

Note: This element is not supported in this release of WebLogic Tuxedo Connector.

Optional. Specifies the `deq_time` (seconds) for a message being redirected into Tuxedo from WebLogic Server if the `JMS_BEA_TuxGtway_Tuxedo_Birthtime` property is not set.

defaultRelativeExpiration

Note: This element is not supported in this release of WebLogic Tuxedo Connector.

Optional. Specifies the `exp_time` (seconds) for a message being redirected into Tuxedo from WebLogic Server if the `JMSExpiration` property is not set.

It is not possible to get the expiration time from a Tuxedo/Q message. Use `defaultRelativeExpiration` to set an expiration time for messages being redirected to Tuxedo/Q.)

expirationAdjustment

Note: This element is not supported in this release of WebLogic Tuxedo Connector.

Optional. Specifies the amount of time (seconds) that is added to the expiration time of a message before it is put onto the target location.

- This value is added to the expiration time when the expiration time for a message is non-zero prior to being retrieved from the source location.
- Use this adjustment to account for any network or processing latency incurred as a result of using the tBridge.

priorityMapping

Required. Provides a method to map the priorities between the JMS and Tuxedo.

- JMS has ten priorities (0 - 9)
- Tuxedo/Q has 100 priorities (1 - 100)

The priorityMapping section provides a mechanism to map the priorities between the Tuxedo and JMS subsystems. There are two mapping directions:

- JmstoTux
- TuxtoJms

Defaults are provided for all values, shown below in pairs of `value:range`. The default values are overridden by describing `pMap` pairs for a mapping direction. The `value` specifies the given input priority and the `range` specifies a sequential group of resulting output priorities.

JmstoTux- 0:1 | 1:12 | 2:23 | 3:34 | 4:45 | 5:56 | 6:67 | 7:78 | 8:89 | 9:100

TuxtoJms- 1-10:0 | 11-20:1 | 21-30:2 | 31-40:3 | 41-50:4 | 51-60:5 | 61-70:6 | 71-80:7 | 81-90:8 | 91-100:9

JmstoTux

Required. Used to specify the priority mapping direction from JMS into Tuxedo /Q message priority.

TuxtoJms

Required. Used to specify the priority mapping direction from Tuxedo /Q into JMS message priority.

pMap

Required. The `pMap` element contains two elements: `value` and `range`. The `value` specifies the given input priority and the `range` specifies a sequential group of resulting output priorities.

range

Required. The `range` specifies a sequential group of resulting output priorities (right side of the `pMap`).

value

Required. The `value` specifies the given input priority (left side of the `pMap`).

deliveryModeOverride

Note: If `deliveryModeOverride` is not specified, then the message is placed on the target location with the same delivery mode specified from the source location.

Optional. Specifies the delivery mode to use when placing messages onto the target location. The `deliveryModeOverride` value overrides any delivery mode associated with a message. Valid values for this parameter are: `PERSIST`, `NONPERSIST`.

defaultreplyDeliveryMode

Note: If neither `defaultReplyDeliveryMode` is specified or `JMS_BEA_TuxGtway_Tuxedo_ReplyDeliveryMode` is set, the default semantics defined for Tuxedo are used by the Tuxedo/Q subsystem.

Optional. Specifies the reply delivery mode to associate with a message when placing messages onto the target location. Use this element for messages being redirected to Tuxedo/Q from JMS when the `JMS_BEA_TuxGtway_Tuxedo_ReplyDeliveryMode` property is not set for a message. Valid values for this parameter are: `PERSIST`, `NONPERSIST`, `DEFAULT`.

userID

Optional. Specifies a user identity for all messages handled by the tBridge for ACL checks when security options are configured.

- All messages assume this identity until the security/authentication contexts are passed between subsystems. Until security contexts are passed, there is no secure method to identify who generated a message recieved from the source location.
- The argument user may be specified as either a user name or a user identification number (uid).

allowNonStandardTypes

Optional. Specifies if non-standard data types are allowed to pass through the tBridge. Valid values for this parameter are: NO, YES.

- NO: Non-standard data types are rejected and placed onto the specified error location.
- YES: Non-Standard data types are placed on the target location as BLOBs with a tag indicating the original type.

Standard data types are:

- ASCII text (TextMessage, STRING)
- BLOB (BytesMessage, CARRAY).

jndiFactory

Required. Name of the jndi lookup factory.

Example: `weblogic.jndi.WLInitialContextFactory`

jmsFactory

Required. Name of the JMS connection factory.

Example: `weblogic.jms.ConnectionFactory`

tuxFactory

Required. Name of the Tuxedo Connection factory.

Example: `tuxedo.services.TuxedoConnection`