BEA WebLogic Server™

WebLogic Tuxedo Connector Administration Guide

BEA WebLogic Server Version 6.1
Document Date: April 24, 2003

Copyright

Copyright © 2002 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

Document Date	Software Version
June 24, 2002	BEA WebLogic Server 6.1

Contents

Audience xii
e-docs Web Site xii
How to Print the Document xiii
Related Information xiii
Contact Us! xiii
Documentation Conventions xiv

1. Introduction to WebLogic Tuxedo Connector

WebLogic Tuxedo Connector Overview 1–2
Key Functionality and Administrative Features 1–2
Known Limitations 1–3
How WebLogic Tuxedo Connector Differs from Jolt 1–4
Documentation 1–4
Platform Support 1–5
Licensing 1–5

2. Configuring WebLogic Tuxedo Connector

Summary of Environment Changes and Considerations 2–1
    Tuxedo Changes 2–1
    WebLogic Server Changes 2–2
        WebLogic Server Threads 2–2
Configuring WebLogic Tuxedo Connector for Your Applications 2–3
    Create a WebLogic Tuxedo Connector XML Configuration File 2–3
        Configuration File Components 2–4
        Using a Local Copy of the wtc_config.dtd 2–6
        Validate the XML file 2–6
    Set the WebLogic Server Environment 2–6

Create a StartUp Class for the WebLogic Tuxedo Connector 2–7
Create a Shutdown Class for the WebLogic Tuxedo Connector..... 2–9
Restart the Application Server..... 2–9
Verify Installation..... 2–9
Configuring WebLogic Tuxedo Connector for Non-ASCII Coding Sets 2–11

3. Configuring BDMCONFIG

Configuring the Connections Between Domains 3–2
How to Request a Connection at Boot Time (ON_STARTUP)..... 3–2
How to Configure RetryInterval 3–3
How to Configure MaxRetries 3–3
How to Request Connections for Client Demands (ON_DEMAND)..... 3–4
Accepting Incoming Connections (INCOMING_ONLY) 3–5
How to use LOCAL Connection Policy..... 3–5
How ConnectionPolicy Affects Dynamic Status..... 3–6
Configuring Domains-level Failover and Failback 3–6
Prerequisite to Using Domains-level Failover and Failback..... 3–7
How to Configure Domains to Support Failover 3–7
How to Configure Domains to Support Failback..... 3–8
Authentication of Remote Domains 3–8
Setting the T_DM_PASSWORD Element..... 3–9
Usage 3–10
Examples 3–10
LocalPasswords..... 3–10
RemotePasswords..... 3–10
AppPasswords 3–10
User Authentication..... 3–11
Security Requirements for servers..... 3–11
Security Requirements for clients 3–12
How to Configure WebLogic Tuxedo Connector to Provide Security between Tuxedo and WebLogic Server..... 3–12
Configure the T_DM_LOCAL_TDOMAIN..... 3–12
Example T_DM_LOCAL_TDOMAIN Configuration 3–13
Example Tuxedo *DM_LOCAL_DOMAINS Configuration..... 3–13
Configure the T_DM_REMOTE_TDOMAIN..... 3–14

Example T_DM_LOCAL_TDOMAIN Configuration 3–15
Example Tuxedo *DM_LOCAL_DOMAINS Configuration 3–15
Example ACL Policy for Simpapp and Simpserv Examples 3–16
Link-Level Encryption 3–21

4. Administration of CORBA Applications

How to Configure WebLogic Tuxedo Connector for CORBA Service Applications 4–1
Example XML Configuration File 4–2
How to Administer and Configure WebLogic Tuxedo Connector for Inbound RMI/IIOP 4–3
Configuring the XML Configuration file 4–3
Administering the Tuxedo Application Environment 4–4
How to Configure WebLogic Tuxedo Connector for Outbound RMI/IIOP 4–5
Example XML Configuration File 4–6

5. Configuring tBridge

Overview of the tBridge 5–2
How tBridge connects JMS with Tuxedo 5–3
How tBridge connects Tuxedo to JMS 5–4
tBridge Limitations 5–4
WebLogic Tuxedo Configuration XML File Configuration for tBridge 5–5
Starting the tBridge 5–5
Error Logging 5–5
tBridge Connectivity 5–6
Example Connection Type Configurations 5–6
Example JmsQ2TuxQ Configuration 5–6
Example TuxQ2JmsQ Configuration 5–8
Example JmsQ2TuxS Configuration 5–9
Priority Mapping 5–11
Error Queues 5–15
wlsServerErrorDestination 5–15
Unsupported Message Types 5–16
tuxErrorQueue 5–16
Limitations 5–16

6. Using FML with WebLogic Tuxedo Connector

Overview of FML 6–1
The WebLogic Tuxedo Connector FML API 6–2
FML Field Table Administration 6–2
Using the DynRdHdr Property for mkfldclass32 Class 6–4
tBridge XML/FML32 Translation 6–6
Using FLAT Translation 6–6
Using NO Translation 6–7
FML32 Considerations 6–7

7. Connecting WebLogic Process Integrator and Tuxedo Applications

Synchronous WebLogic Process Integrator-to-Tuxedo Connectivity 7–2
Defining Business Operations 7–2
Invoking an eLink Adapter 7–2
Define Exception handlers 7–2
Synchronous Non-Blocking WebLogic Process Integrator-to-Tuxedo Connectivity 7–3
Asynchronous WebLogic Process Integrator-to-Tuxedo Connectivity 7–3
Asynchronous Tuxedo /Q-to-WebLogic Process Integrator Connectivity 7–4
Bi-directional Asynchronous Tuxedo-to-WebLogic Process Integrator Connectivity 7–5

8. Troubleshooting The WebLogic Tuxedo Connector

Monitoring the WebLogic Tuxedo Connector 8–1
Setting Trace Levels 8–1
Console Settings 8–2
Frequently Asked Questions 8–3
Cannot Find Configuration File 8–3
EJB Deployment Message 8–4
Connection Problems 8–4

9. The WebLogic Tuxedo Connector XML Configuration File

Creating an XML Configuration File 9–1
Sample XML Configuration File 9–2

Validate the XML file 9–3
Element Hierarchy Diagram 9–4

10. The wtc_config.dtd

wtc_config.dtd 10–1

11. Elements and Attributes of the wtc_config.dtd

WTC_CONFIG 11–1
BDMCONFIG 11–2
T_DM_LOCAL_TDOMAIN 11–2
WlsClusterName 11–2
AccessPointId 11–2
Type 11–3
Security 11–3
ConnectionPolicy 11–3
RetryInterval 11–4
MaxRetries 11–4
ConnPrincipalName 11–5
NWAddr 11–5
CmpLimit 11–6
MinEncryptBits 11–6
MaxEncryptBits 11–6
Interoperate 11–7
BlockTime 11–7
T_DM_REMOTE_TDOMAIN 11–7
LocalAccessPoint 11–7
AclPolicy 11–8
CredentialPolicy 11–8
FederationURL 11–8
FederationName 11–9
TpUsrFile 11–9
T_DM_EXPORT 11–9
RemoteName 11–9
EJBName 11–10
T_DM_IMPORT 11–10

TranTime 11–10
T_DM_PASSWORD 11–11
LocalPassword 11–11
RemotePassword 11–12
T_DM_RESOURCES 11–12
FieldTables 11–12
FldTblClass 11–12
AppPassword 11–13
tBridge 11–13
direction 11–13
fromto 11–13
redirect 11–14
source 11–14
target 11–14
AccessPoint 11–14
Qspace 11–14
Name 11–14
ReplyQ 11–14
metadataFile 11–14
translateFML 11–15
transactional 11–15
timeout 11–15
idleTime 11–16
retries 11–16
retryDelay 11–16
wlsErrorDestination 11–17
tuxErrorQueue 11–17
defaultRelativeBirthtime 11–17
defaultRelativeExpiration 11–17
expirationAdjustment 11–18
priorityMapping 11–18
JmstoTux 11–18
TuxtoJms 11–19
pMap 11–19
range 11–19

value 11–19
deliveryModeOverride 11–19
defaultReplyDeliveryMode 11–19
userID 11–20
allowNonStandardTypes 11–20
jndiFactory 11–20
jmsFactory 11–21
tuxFactory 11–21

About This Document

This document introduces the BEA WebLogic Tuxedo Connector™ application development environment. This document provides information on how to configure and administer the WebLogic Tuxedo Connector to interoperate between WebLogic Server and Tuxedo.

The document is organized as follows:

- Chapter 1, "Introduction to WebLogic Tuxedo Connector," is an overview of the WebLogic Tuxedo Connector.
- Chapter 2, "Configuring WebLogic Tuxedo Connector," describes how to configure the WebLogic Tuxedo Connector.
- Chapter 3, "Configuring BDMCONFIG," provides configuration information about BDMCONFIG.
- Chapter 4, "Administration of CORBA Applications," provides information on how to administer CORBA applications.
- Chapter 5, "Configuring tBridge," provides information on tBridge functionality and configuration.
- Chapter 6, "Using FML with WebLogic Tuxedo Connector," discusses the Field Manipulation Language (FML) and describes how the WebLogic Tuxedo Connector uses FML.
- Chapter 7, "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 8, "Troubleshooting The WebLogic Tuxedo Connector," provides WebLogic Tuxedo Connector troubleshooting information.

- Chapter 9, "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 10, "The wtc_config.dtd," provides the structure of elements and attributes used to create a WebLogic Tuxedo Connector XML configuration file.
- Chapter 11, "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.
italics	Emphasis and book titles.

Convention	Usage

`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.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String CustomerName;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*: `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
`|`	Separates mutually exclusive choices in a syntax line. *Example*: `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
`...`	Indicates one of the following in a command line: ■ 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

Convention	Usage
. . .	Indicates the omission of items from a code example or from a syntax line.

CHAPTER

1 Introduction to WebLogic Tuxedo Connector

The following sections summarize the concepts and functionality of WebLogic Tuxedo Connector for WebLogic Server Release 6.1 Service Pack 2:

- WebLogic Tuxedo Connector Overview
- Key Functionality and Administrative Features
- Known Limitations
- How WebLogic Tuxedo Connector Differs from Jolt
- Documentation
- Platform Support
- Licensing

WebLogic Tuxedo Connector Overview

The WebLogic Tuxedo Connector provides interoperability between WebLogic Server applications and Tuxedo services. The connector uses an XML configuration file that allows 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 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.
- Transaction support.
- Ability to provide interoperability between CORBA Java and CORBA C++ server applications.
- 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.
- 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.

Known Limitations

WebLogic Tuxedo Connector has the following limitations:

- Does not support dynamic configuration changes to WebLogic Tuxedo Connector gateway
- Does not support View buffers.
- Does not support in-bound RMI/IIOP transactions from a Tuxedo application..
- Does not support clustering. Only one instance of WebLogic Tuxedo Connector can be used in a clustered environment.
- Does not support Tuxedo 6.5 running on VMS, AS/400, and OS/390 platforms.
- Does not support outbound CORBA applications from WebLogic Server running on IBM AIX platforms. WebLogic Server uses a Sun specific extension for creating sockets that is not compatible with IBM JDK.

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.

Documentation

You can download the WebLogic Tuxedo Connector documentation from the following location:

- On the BEA corporate Web Site. From the BEA Home page at http://www.bea.com, click on Product Documentation.
- Go directly to the WebLogic Server "e-docs" product documentation page at http://e-docs.bea.com/index.html. Follow the links from the WebLogic Server 6.1 Documentation Center.

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 Configuring WebLogic Tuxedo Connector

The following sections describe how to configure the WebLogic Tuxedo Connector.

- Summary of Environment Changes and Considerations
- Configuring WebLogic Tuxedo Connector for Your Applications
- Configuring WebLogic Tuxedo Connector for Non-ASCII Coding Sets

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.

Tuxedo Changes

Note: For more information on Tuxedo domains, see the *BEA TUXEDO Domains Guide*.

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.

WebLogic Server Changes

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

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 Server.

WebLogic Server Threads

Note: For more information on WebLogic Server performance and tuning, see BEA WebLogic Server Performance and Tuning, Release 6.1.

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 to increase the number of available threads. Use a reasonable thread model when invoking service EJBs. You may need to increase the number of WebLogic Server threads available to a larger value.

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 WebLogic Tuxedo Connector 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 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 WebLogic Tuxedo Connector XML Configuration File
- Set the WebLogic Server Environment
- Create a StartUp Class for the WebLogic Tuxedo Connector
- Create a Shutdown Class for the WebLogic Tuxedo Connector
- Restart the Application Server
- Verify Installation

Create a WebLogic Tuxedo Connector XML 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: A WebLogic Tuxedo Connector XML configuration file must be installed on all WebLogic Servers 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\DBMCONFIG.xml`
- Save the configuration file with your WebLogic Server application.

Note: For more information on configuration file hierarchy and element definitions, see Chapter 9, "The WebLogic Tuxedo Connector XML Configuration File."

Configuration File Components

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

- Version
- DOCTYPE
- WTC_CONFIG
- BDMCONFIG
- tBridge

Version

Required. Specifies the version of XML used.

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

DOCTYPE

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. BEA maintains the WebLogic Tuxedo Connector DTD at the following location: `http://www.bea.com/servers/wls610/dtd/wtc_config.dtd`

Example:

```
<!DOCTYPE WTC_CONFIG SYSTEM
"http://www.bea.com/servers/wls610/dtd/wtc_config.dtd">
```

WTC_CONFIG

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

- `BDMCONFIG`
- `tBridge`

BDMCONFIG

Note: For more information on how to configure BDMCONFIG, see Chapter 3, "Configuring 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. The following child elements make up the BDMCONFIG:

- `T_DM_LOCAL_DOMAIN`: provides a view of local domains as they appears to other (remote) domains.
- `T_DM_REMOTE_TDOMAIN`: provides a view of remote domains as they appear to the local domains.
- `T_DM_EXPORT`: provides information on services exported by a local domain.
- `T_DM_IMPORT`: provides information on services available on remote domains imported to local domains.
- `T_DM_PASSWORD`: provides configuration information for inter-domain authentication through access points of type TDOMAIN.
- `T_DM_RESOURCES`: provides information to specify global field table classes and application passwords for domains.

tBridge

Note: For more information on how to configure the tBridge, see Chapter 5, "Configuring 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.

Using a Local Copy of the wtc_config.dtd

WebLogic Tuxedo Connector uses an external internet connection lookup the `wtc_config.dtd` file from the BEA DTD repository. You may need to use a local copy of the `wtc_config.dtd` file to meet the security requirements of your installation. If you need to create a local copy of the `wtc_config.dtd`, do the following:

- Save a copy of the `wtc_config.dtd` in your local directory. Make sure it is saved as a text file.
- Update the DOCTYPE declaration of the `wtc_config.dtd` in your local directory to reflect the new fully qualified path and file name. Remember to change **`http`** to **`file`**.

Example:

```
<!DOCTYPE BDMCONFIG SYSTEM
"file:my_bea_directory\weblogic\wtc\gwt\wtc_config.dtd">
```

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 Environment

You need to set the environment of your WebLogic Server application by running the `setEnv` script.

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

- UNIX users: run `setEnv.sh`

If you are setting the environment for the first time, you will need to review the settings in the script. If necessary, use the following steps to modify the settings for your application environment:

1. From the command line, change directories to the location of the WebLogic Server application.
2. Edit the `setEnv` script with a text editor, such as `vi`.

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

   - UNIX users: edit `setEnv.sh`
3. Save the file.

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

Create a StartUp Class for the WebLogic Tuxedo Connector

In order to use the WebLogic Tuxedo Connector, you must create a WebLogic Server 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. Run the `startWebLogic` script.

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

   - UNIX users: run `startWebLogic.sh`

   WebLogic Server starts.
2. Start the WebLogic Server Console.
3. If necessary, right-click on the domain root and select **Create or edit other domains**. Left-click to select a domain from the repository.
4. Left-click and expand the **Deployments** branch.

5. Right-click on the **Startup & Shutdown** branch.
6. Select **Create a new Startup Class . . .**

   The *Configuration* tab is active.
7. Enter the **Name**.

   Example: MyWTCStartup Class
8. Enter **`weblogic.wtc.gwt.WTCStartup`** for the **Classname**.
9. 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`
10. Check **Abort startup on failure**.
11. Click **Create**.
12. Select the *Target* tab.
13. From the Available servers list, click on the server you wish to select.

    The server is highlighted.
14. Click on the right-arrow.

    The selected server appears in the *Chosen* servers list.
15. Click **Apply**.

Create a Shutdown Class for the WebLogic Tuxedo Connector

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: MyWTCShutdown Class
4. Enter **`weblogic.wtc.gwt.WTCShutdown`** for the **Classname**.
5. Click **Create**.

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

```
.
.
.
<StartupClass
     Arguments="BDMCONFIG=d:\bea\wlserver6.1\config\examples\bdmconfig.xml,Trac
eLevel=100000"
     ClassName="weblogic.wtc.gwt.WTCStartup" FailureIsFatal="true"
     Name="MyWTCStartup Class" Targets="examplesServer"/>
<ShutdownClass ClassName="weblogic.wtc.gwt.WTCShutdown" Name="MyWTCShutdown
Class"/>
```

.

.

.

If you have configured **`TraceLevel`**, you can view the Weblogic Server log file for WebLogic Tuxedo Connector entries. You should see messages similar to those shown below when **`TraceLevel=100000`**:

.

.

.

```
####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <]/WTCStartup/crossCheck/50/true>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <]/WTCStartup/extractInfo/80/true; DONE>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <]/WTCStartup/loadFile/90/void; LOADED>

####<Jul 25, 2001 6:27:38 PM EDT> <Info> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180001> <Done Loading the XML config file.>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <Done loading the XML Config File.>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <Setting up federation points>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <Federating [TDOM1] to [tgiop://TDOM1]>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-5> <> <> <180056> <[/WTCStartup/OatmialListener/run/>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> </WTCStartup/recover returned null>

####<Jul 25, 2001 6:27:38 PM EDT> <Debug> <WTC> <randyr-nt> <examplesServer>
<Thread-3> <system> <> <180056> <]/WTCStartup/startup/100/WTC started...>

####<Jul 25, 2001 6:27:38 PM EDT> <Info> <WebLogicServer> <randyr-nt>
<examplesServer> <Thread-3> <system> <> <000288> <weblogic.wtc.gwt.WTCStartup
reports: WTC started...>
```

.

.

.

If your installation is not successful, see Chapter 8, "Troubleshooting The WebLogic Tuxedo Connector."

Configuring WebLogic Tuxedo Connector for Non-ASCII Coding Sets

Note: For more information about setting WebLogic Server properties, see Starting and Stopping WebLogic Servers at http://e-docs.bea.com/wls/docs61/adminguide/startstop.html.

To transfer non-ascii (multibyte) strings between WebLogic Server and Tuxedo applications, you must configure WebLogic Tuxedo Connector to provide character set translation. WebLogic Tuxedo Connector uses a WebLogic Server property to match the encoding used by all the Tuxedo remote domains specified in a WebLogic Tuxedo Connector service. If you require more than one coding set running simultaneously, you will require WebLogic Tuxedo Connector services running in separate WebLogic Server instances.

To enable character set translation, update the JAVA_OPTIONS variable in your server start script. Example:

```
JAVA_OPTIONS=-Dweblogic.wtc.encoding=codesetname
```

where *codesetname* is the name of a supported codeset used by a remote Tuxedo domain. See Supported Encodings at http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html for list of supported base and extended coding sets.

You may not be able to select the exact encoding name to match the encoding used by the remote domain. In this situation, you should select an encoding name that is equivalent to the remote domain.

Example:

- The Supported Encoding list includes `EUC_JP`

- The remote domain is supported by a Solaris operating system using `eucJP`

Although the names don't match exactly, `EUC_JP` and `eucJP` are equivalent encoding sets and provide the correct string translation between WebLogic Server and your remote domain. You should set the encoding property to `EUC_JP`:

```
JAVA_OPTIONS=-Dweblogic.wtc.encoding=EUC_JP
```

3 Configuring BDMCONFIG

Note: For more detailed reference information on the WebLogic Tuxedo Connector XML configuration file, elements and attributes, and the `wtc_config.dtd`, see Chapter 10, "The wtc_config.dtd."

The BDMCONFIG section of the WebLogic Tuxedo Connector XML configuration file describes how to establish connectivity and provide security between domains in the WebLogic Server 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.

If the WebLogic Tuxedo Connector is configured, it 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:

- Configuring the Connections Between Domains
- How ConnectionPolicy Affects Dynamic Status
- Configuring Domains-level Failover and Failback
- Authentication of Remote Domains
- User Authentication
- How to Configure WebLogic Tuxedo Connector to Provide Security between Tuxedo and WebLogic Server
- Example ACL Policy for Simpapp and Simpserv Examples

- Link-Level Encryption

Configuring the Connections Between Domains

Note: For more information on Dynamic Status, see "How ConnectionPolicy Affects Dynamic Status" on page 3-6.

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. You can select any of the following connection policies:

- How to Request a Connection at Boot Time (ON_STARTUP)
- How to Request Connections for Client Demands (ON_DEMAND)
- Accepting Incoming Connections (INCOMING_ONLY)
- How to use LOCAL Connection Policy

For connection policies of `ON_STARTUP` and `INCOMING_ONLY`, Dynamic Status is invoked. Dynamic Status checks and reports on the status of remote services.

How to Request a Connection 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 and the `MaxRetries` parameter. To request a connection at boot time, the following entry is required in the BDMCONFIG section of your XML configuration file:

Example:

```
<ConnectionPolicy>ON_STARTUP</ConnectionPolicy>
```

How to Configure RetryInterval

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. The minimum value is 0; the default value is 60, and maximum value is 2147483647. The following example specifies that the gateway waits 30 seconds before trying to establish a connection again.

Example:

```
<RetryInterval>30</RetryInterval>
```

How to Configure MaxRetries

Note: Use only when ConnectionPolicy is set to ON_STARTUP. For other connection policies, retry processing is disabled.

You indicate the number of times a domain gateway tries to establish connections to remote domain access points before quitting by assigning a value to the `MaxRetries` parameter: the minimum value is 0; the default and maximum value is 2147483647.

- If you set `MaxRetries` to 0, automatic connection retry processing is turned off. The server does not attempt to connect to the remote gateway automatically.

  Example:

  ```
  <MaxRetries>0</MaxRetries>
  ```

- If you set `MaxRetries` to a number, the gateway tries to establish a connection the specified number of times before quitting. In the following example, the server tries to establish a connection 10 times before quitting.

  Example:

  ```
  <MaxRetries>10</MaxRetries>
  ```

- If you set `MaxRetries` to 2147483647, retry processing is repeated indefinitely or until a connection is established.

  Example:

```
<MaxRetries>2147483647</MaxRetries>
```

Table 3-1 Example Settings of MaxRetries and RetryInterval Parameters

If you set ...	Then ...

`<ConnectionPolicy>ON_STARTUP</ConnectionPolicy>`
`<RetryInterval>30</RetryInterval>`
`<MaxRetries>3</MaxRetries>`
	The gateway makes 3 attempts to establish a connection, at 30 seconds intervals, before quitting.

`<ConnectionPolicy>ON_STARTUP</ConnectionPolicy>` `<MaxRetries>0</MaxRetries>`	The gateway attempts to establish a connection at initialization time but does not retry if the first attempt fails.
`<ConnectionPolicy>ON_STARTUP</ConnectionPolicy>` `<RetryInterval>30</RetryInterval>`	The gateway attempts to establish a connection every 30 seconds until a connection is established.

How to Request Connections for Client Demands (ON_DEMAND)

Note: If the `ConnectionPolicy` is not specified in the XML configuration file, the WebLogic Tuxedo Connector uses a `ConnectionPolicy` of `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. To allow requests for client demands for a connection, use the following entry in the BDMCONFIG section of your XML configuration file:

Example:

```
<ConnectionPolicy>ON_DEMAND</ConnectionPolicy>
```

Accepting Incoming Connections (INCOMING_ONLY)

A connection policy of `INCOMING_ONLY` means that a domain gateway does not to establish a connection to remote domains upon starting. 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. To allow requests for client demands for a connection, use the following entry in the BDMCONFIG section of your XML configuration file:

Example:

```
<ConnectionPolicy>INCOMING_ONLY</ConnectionPolicy>
```

How to use LOCAL Connection Policy

Note: A ConnectionPolicy of LOCAL is not valid for local domains.

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`). To set a remote domain connection policy to `LOCAL`, use the following entry in the T_DM_REMOTE_TDOMAIN section of your XML configuration file:

Example:

```
<ConnectionPolicy>LOCAL</ConnectionPolicy>
```

How ConnectionPolicy Affects 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 `ConnectionPolicy` 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 Domains-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.

Domains-level failover is a mechanism that transfers requests to alternate remote domains when a failure is detected with a primary remote domain. It also provides failback to the primary remote domain when that domain is restored.

This level of failover/failback depends on Dynamic Status. The domain must be configured with a `CONNECTION_POLICY` of `ON_STARTUP` or `INCOMING_ONLY` to enable Domains-level failover/failback.

Domains-level failover/failback defines a remote domain as available when a network connection to the remote domain exists, and unavailable when a network connection to the remote domain does not exist.

Prerequisite to Using Domains-level Failover and Failback

To use Domains-level failback, you must specify `ON_STARTUP` or `INCOMING_ONLY` as the value of the `CONNECTION_POLICY` parameter.

A connection policy of `ON_DEMAND` is unsuitable for Domains-level failback as it operates on the assumption that the remote domain is always available. If you do not specify `ON_STARTUP` or `INCOMING_ONLY` as your connection policy, your servers cannot fail over to the alternate remote domains that you have specified with the Tuxedo `RDOM` parameter.

Note: A remote domain is *available* if a network connection to it exists; a remote domain is *unavailable* if a network connection to it does not exist.

How to Configure Domains to Support Failover

To support failover, you must specify the remote domains responsible for executing a particular service. Specifically, you must specify each remote domain with a T_DM_REMOTE_TDOMAIN section in your XML configuration file.

Suppose a service is available from two remote domains: `TDOM1` and `TDOM3`. Include the following entry in the BDMCONFIG section on your XML configuration file:

```
<T_DM_REMOTE_TDOMAIN AccessPoint="TDOM1">
                <LocalAccessPoint>TDOM2</LocalAccessPoint>
                <AccessPointId>TDOM1</AccessPointId>
                <Type>TDOMAIN</Type>
```

```
                <NWAddr>//mydomain.acme.com:20305</NWAddr>
</T_DM_REMOTE_TDOMAIN>
<T_DM_REMOTE_TDOMAIN AccessPoint="TDOM3">
                <LocalAccessPoint>TDOM2</LocalAccessPoint>
                <AccessPointId>TDOM3</AccessPointId>
                <Type>TDOMAIN</Type>
                <NWAddr>//myotherdomain.com:50302</NWAddr>
</T_DM_REMOTE_TDOMAIN>
```

How to Configure Domains to Support Failback

Failback occurs when a network connection to the primary remote domain is reestablished for any of the following reasons:

- Automatic retries (`ON_STARTUP` only)
- Incoming connections

Authentication of Remote Domains

Note: Tuxedo 6.5 users should set the `Interoperate` parameter to `Yes` and set the `Security` parameter to `NONE`. If you require security features and use the WebLogic Tuxedo Connector, you will need to upgrade to Tuxedo 7.1 or higher.

Domain gateways can be made to authenticate incoming connections requested by remote domains and outgoing connections requested by local domains. Application administrators can define when security should be enforced for incoming connections from remote domains. You can specify the level of security used by a particular local domain by setting the `SECURITY` parameter in the T_DM_LOCAL_TDOMAIN section of your XML configuration file. There are three levels of password security:

- No Security (using the `NONE` option)—incoming connections from remote domains are not authenticated.
- Application Password (using the `APP_PW` option)— incoming connections from remote domains are authenticated using the application password defined in the T_DM_PASSWORD element of the XML configuration file. You use the `weblogic.wtc.gwt.genpasswd` utility to create encrypted application passwords.
- Remote Domains Password (using the `DM_PW` option)—this feature enforces security between two or more domains. Connections between the local and remote domains are authenticated using password pairs defined in the T_DM_PASSWORD element of the XML configuration file. You use the `weblogic.wtc.gwt.genpasswd` utility to create encrypted remote domains passwords.

The `SECURITY` parameter in the T_DM_LOCAL_TDOMAIN section of your XML configuration file must match the SECURITY parameter of the *DM_LOCAL_DOMAINS section of the Tuxedo domain configuration file.

- If authentication is required, it is done every time a connection is established between the local domain and a remote domain.
- If the security type of the T_DM_LOCAL_TDOMAIN does not match the security type of the *DM_LOCAL_DOMAINS or if the passwords do not match, the connection fails.

Setting the T_DM_PASSWORD Element

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.

- The XML configuration file does not store clear text passwords.
- The key must 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>
```

User Authentication

Note: Tuxedo 6.5 users should set the `Interoperate` parameter to `Yes` and set the `Security` parameter to `NONE`. The `AclPolicy` and `CredentialPolicy` elements are ignored if the WebLogic Tuxedo Connector is used to interoperate with Tuxedo 6.5. If you require security features and use the WebLogic Tuxedo Connector, you will need to upgrade to Tuxedo 7.1 or higher.

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.

How to Configure WebLogic Tuxedo Connector to Provide Security between Tuxedo and WebLogic Server

Note: Tuxedo 6.5 does not have the required security infrastructure to support security mapping. Tuxedo 6.5 users should set the `Interoperate` parameter to `Yes` and set the `Security` parameter to `NONE`. If you require security features and use the WebLogic Tuxedo Connector, you will need to upgrade to Tuxedo 7.1 or higher.

Use the following steps to configure WebLogic Tuxedo Connector to provide security between Tuxedo and WebLogic Server applications:

- Configure the T_DM_LOCAL_TDOMAIN
- Configure the T_DM_REMOTE_TDOMAIN

Configure the T_DM_LOCAL_TDOMAIN

Set the `SECURITY` parameter in the T_DM_LOCAL_TDOMAIN section of your XML configuration file to match the SECURITY parameter of the *DM_LOCAL_DOMAINS section of the Tuxedo domain configuration file.

Example T_DM_LOCAL_TDOMAIN Configuration

```
.
.
.
<T_DM_LOCAL_TDOMAIN AccessPoint="Your weblogic domain">
     <Type>TDOMAIN</Type>
     <Security>DM_PW</Security>
     <ConnectionPolicy>ON_DEMAND</ConnectionPolicy>
     <ConnPrincipalName>wldom1</ConnPrincipalName>
     <NWAddr>//DKUMAR:3045</NWAddr>
     <Interoperate>Yes</Interoperate>
</T_DM_LOCAL_TDOMAIN>
.
.
.
```

Example Tuxedo *DM_LOCAL_DOMAINS Configuration

```
.
.
.
*DM_LOCAL_DOMAINS
domain1      DOMAINID = "domain1"
      GWGRP = "GWGRP"
      CONNECTION_POLICY=ON_DEMAND
      DMTLOGDEV = "/nfs/home1/dkumar/lcsol5/tmp.100/DMTLOG"
      DMTLOGNAME = "DMTLG1"
      SECURITY=DM_PW
```

```
        BLOCKTIME=10
.
.
.
```

Configure the T_DM_REMOTE_TDOMAIN

Configure the T_DM_REMOTE_TDOMAIN to establish an inbound and outbound Access Control List (ACL) policy.

Perform the following steps to prepare the WebLogic Server environment:

1. Add an AclPolicy element to the T_DM_REMOTE_TDOMAIN section of the XML configuration file.

   Example:

   ```
   <AclPolicy>GLOBAL</AclPolicy>
   ```

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

   Example:

   ```
   <CredentialPolicy>GLOBAL</CredentialPolicy>
   ```

3. If the `CredentialPolicy` is set to `GLOBAL`, you must have a copy of the Tuxedo `tpusr` file in your WebLogic Server environment. Use the following steps to configure the WebLogic Tuxedo Connector by adding a `TpUserFile` element to your XML configuration file:

   a. Copy the `tpusr` file from TUXEDO to the WebLogic Server application environment or generate your own `tpusr` file.

   b. Add a TpUserFile element to the T_DM_REMOTE_TDOMAIN section of the XML configuration file.

      Example:

      ```
      <TpUsrFile>pathname_filename_of_tpusr_file</TpUsrFile>
      ```

Note: For more information on how to create a Tuxedo tpusr file, see How to Enable User-Level Authentication at http://e-docs.bea.com/tuxedo/tux80/atmi/secadm19.htm.

Example T_DM_LOCAL_TDOMAIN Configuration

```
.
.
.
<T_DM_REMOTE_TDOMAIN AccessPoint="Your Tuxedo domain">
     <LocalAccessPoint>wldom1</LocalAccessPoint>
     <AccessPointId>domain1</AccessPointId>
     <Type>TDOMAIN</Type>
     <AclPolicy>LOCAL</AclPolicy>
     <ConnPrincipalName>domain1</ConnPrincipalName>
     <CredentialPolicy>LOCAL</CredentialPolicy>

     <TpUsrFile>C:\runs\tmp.100\config\wldom1\tpusr</TpUsrFile>
     <NWAddr>//lcsol5:3500</NWAddr>
</T_DM_REMOTE_TDOMAIN>
.
.
.
```

Example Tuxedo *DM_LOCAL_DOMAINS Configuration

```
.
.
.
*DM_REMOTE_DOMAINS
     wldom1 DOMAINID = "wldom1"
```

```
        ACCESSPOINTID="wldom1"
        ACL_POLICY=”LOCAL”
        CREDENTIAL_POLICY="LOCAL"
.
.
.
```

Example ACL Policy for Simpapp and Simpserv Examples

This section provides an example of how to set up ACL control using the `simpapp` and `simpserv` 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 `method_permission` elements for the Tuxedo TOUPPER service. The bolded statements indicate the changes added to support the security implementation.

Listing 3-1 TOUPPER ejb-jar.xml Security Example Code

```
<?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-assignment` element for the Tuxedo TOUPPER service. The bolded statements indicate the changes added to support the security implementation.

Listing 3-2 TOUPPER Weblogic-ejb-jar.xml Security Example Code

```
<?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 `method-permission` elements for the Tolower service. The bolded statements indicate the changes added to support the security implementation.

Listing 3-3 Tolower ejb-jar.xml Security Example Code

```
<?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.simpserv.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-assignment` element for the Tolower service.The bolded statements indicate the changes added to support the security implementation.

Listing 3-4 Tolower Weblogic-ejb-jar.xml Security Example Code

```
<?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". 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>
  ```

Link-Level Encryption

You can use encryption across domains to ensure data privacy. In this way, a network-based eavesdropper cannot learn the content of messages or application-generated messages flowing from one domain gateway to another. You configure this security mechanism by setting the `MINENCRYPTBITS` and `MAXENCRYPTBITS` parameters of the T_DM_LOCAL_TDOMAIN and the T_DM_REMOTE_TDOMAIN sections of your XML configuration file.

Note: Encryption requires appropriate licensing. For more information on license requirements, see "Licensing" on page 1-5.

4 Administration of CORBA Applications

Note: For more information on CORBA applications, see Tuxedo CORBA at `http://e-docs.bea.com/tuxedo/tux80/interm/corba.htm`.

The following sections provide information on how to administer and configure the WebLogic Tuxedo Connector to support Tuxedo CORBA clients and services.

- How to Configure WebLogic Tuxedo Connector for CORBA Service Applications
- How to Administer and Configure WebLogic Tuxedo Connector for Inbound RMI/IIOP
- How to Configure WebLogic Tuxedo Connector for Outbound RMI/IIOP

How to Configure WebLogic Tuxedo Connector for CORBA Service Applications

Note: For more information on how to configure your XML configuration file, see "The WebLogic Tuxedo Connector XML Configuration File" on page 9-1.

This section provides information on how to configure your XML configuration file to support a call to a Tuxedo CORBA server from a WebLogic Server EJB. Use the following steps to modify the `BDMCONFIG` section of your XML configuration file:

1. Configure a `T_DM_LOCAL_TDOMAIN section` for your WebLogic Server domain.
2. Configure a `T_DM_REMOTE_TDOMAIN` section for your Tuxedo CORBA domain.
3. Configure a `T_DM_IMPORT` section.
   - `Set` **`ResourceName`** to *"//domain_id"* where *domain_id* is `DOMAINID` specified in the Tuxedo `UBBCONFIG` file. The maximum length of this unique identifier for CORBA domains is 15 characters and includes the //.
   - `Set` **`LocalAccessPoint`** to the value of the `LocalAccessPoint` element of the `T_DM_REMOTE_TDOMAIN`.
   - **`RemoteAccessPointList`** to the value of the `AccessPointId` element of the `T_DM_REMOTE_TDOMAIN`.

For information on how to develop client applications that call a Tuxedo CORBA service using a WebLogic Server EJB, see the *WebLogic Tuxedo Connector Programmer's Guide* at http://e-docs.bea.com/wls/docs61/wtc_atmi/index.html.

Example XML Configuration File

The following XML configuration file provides an example of how to configure the WebLogic Tuxedo Connector to a TUXEDO CORBA server.

Listing 4-1 Example XML Configuration File for a CORBA Server Application

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

<!DOCTYPE WTC_CONFIG SYSTEM
"http://www.bea.com/servers/wls610/dtd/wtc_config.dtd">

<!--Java and XML-->
     <WTC_CONFIG>
     <BDMCONFIG>
          <T_DM_LOCAL_TDOMAIN AccessPoint="examples">
               <WlsClusterName>Coolio</WlsClusterName>
               <AccessPointId>examples</AccessPointId>
               <Type>TDOMAIN</Type>
               <Security>NONE</Security>
               <ConnectionPolicy>ON_DEMAND</ConnectionPolicy>
```

```
            <BlockTime>30</BlockTime>
            <NWAddr>//localhost:20304</NWAddr>
        </T_DM_LOCAL_TDOMAIN>
        <T_DM_REMOTE_TDOMAIN AccessPoint="TUXDOM">
            <LocalAccessPoint>examples</LocalAccessPoint>
            <AccessPointId>TUXDOM</AccessPointId>
            <Type>TDOMAIN</Type>
            <NWAddr>//localhost:20305</NWAddr>
        </T_DM_REMOTE_TDOMAIN>
        <T_DM_IMPORT
            ResourceName="//simpapp"
            LocalAccessPoint="examples"
            RemoteAccessPointList="TUXDOM">
        </T_DM_IMPORT>
</BDMCONFIG>
</WTC_CONFIG>
```

How to Administer and Configure WebLogic Tuxedo Connector for Inbound RMI/IIOP

This section provides information on how to administer your application environment and configure your XML configuration file to enable Tuxedo CORBA objects to invoke upon EJBs deployed in WebLogic Server using the RMI/IIOP API.

- Configuring the XML Configuration file
- Administering the Tuxedo Application Environment

Configuring the XML Configuration file

Note: For more information on how to configure your XML configuration file, see "The WebLogic Tuxedo Connector XML Configuration File" on page 9-1.

This section provides information on how to configuring the WebLogic Tuxedo Connector to enable Tuxedo CORBA objects to invoke upon EJBs deployed in WebLogic Server using the RMI/IIOP API. Use the following steps to modify the `BDMCONFIG` section of your XML configuration file:

1. Configure the `T_DM_LOCAL_TDOMAIN section` for your WebLogic Server domain
2. Configure the `T_DM_REMOTE_TDOMAIN section` for your Tuxedo CORBA domain

Administering the Tuxedo Application Environment

Note: For more information on how to configure your Tuxedo application environment, see Tuxedo Administration Topics at http://e-docs.bea.com/tuxedo/tux80/interm/admin.htm.

You must perform some additional steps when configuring your Tuxedo application environment.

1. Set the TOBJADDR for your environment.

   Example: `//<hostname>:2468`

2. Register WebLogic Server (WLS) Naming Service in the Tuxedo domain's CosNaming namespace by entering the following command:

   ```
   cnsbind -o ior.txt WLS
   ```

The `ior.txt` file contains the URL of the WebLogic Server's domain Naming Service.

Listing 4-2 ior.txt File for iiop.ejb.stateless.server.tux Tuxedo Client Example

```
corbaloc:tgiop:examples/NameService
```

How to Configure WebLogic Tuxedo Connector for Outbound RMI/IIOP

Note: For more information on how to configure your XML configuration file, see "The WebLogic Tuxedo Connector XML Configuration File" on page 9-1.

This section provides information on how to configure your XML configuration file to enable WebLogic Server EJBs to invoke upon Tuxedo CORBA objects using the RMI/IIOP API. Use the following steps to modify the `BDMCONFIG` section of your XML configuration file:

1. Configure the `T_DM_LOCAL_TDOMAIN section` for your WebLogic Server domain.
2. Configure a `T_DM_REMOTE_TDOMAIN` section for your Tuxedo CORBA domain. Outbound RMI/IIOP requires two additional elements: **`FederationURL`** and **`FederationName`**.
   - Set **`FederationURL`** to the URL for a foreign name service that is federated into the JNDI. This must be the same URL used by the EJB to obtain the initial context used to access the remote Tuxedo CORBA object.
   - Set **`FederationName`** to the symbolic name of the federation point.

`Note:` `For more information, see "FederationURL" on page 11-8 and "FederationName" on page 11-9.`

3. Configure a `T_DM_IMPORT` section.
   - Set **`ResourceName`** to "*`//domain_id`*" where *domain_id* is `DOMAINID` specified in the Tuxedo `UBBCONFIG` file of the remote Tuxedo domain where the object is deployed. The maximum length of this unique identifier for CORBA domains is 15 characters including the //.
   - Set **`LocalAccessPoint`** to the value of the `LocalAccessPoint` element of the `T_DM_REMOTE_TDOMAIN`.
   - **`RemoteAccessPointList`** to the value of the `AccessPointId` element of the `T_DM_REMOTE_TDOMAIN`.
   - Optional. Specify the transaction timeout value.

For information on how to develop applications that use RMI/IIOP to call a Tuxedo service using a WebLogic Server EJB, see the *WebLogic Tuxedo Connector Programmer's Guide* at http://e-docs.bea.com/wls/docs61/wtc_atmi/index.html.

Example XML Configuration File

The following XML configuration file provides an example of how to configure the WebLogic Tuxedo Connector for outbound RMI/IIOP.

Listing 4-3 Example XML Configuration File for Outbound RMI/IIOP

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

<!DOCTYPE WTC_CONFIG SYSTEM
"http://www.bea.com/servers/wls610/dtd/wtc_config.dtd">

<!--Java and XML-->
<WTC_CONFIG>
<BDMCONFIG>
     <T_DM_LOCAL_TDOMAIN AccessPoint="examples">
          <WlsClusterName>Coolio</WlsClusterName>
          <AccessPointId>examples</AccessPointId>
          <Type>TDOMAIN</Type>
          <Security>NONE</Security>
          <ConnectionPolicy>ON_DEMAND</ConnectionPolicy>
          <BlockTime>30</BlockTime>
          <NWAddr>//127.0.0.1:5000</NWAddr>
     </T_DM_LOCAL_TDOMAIN>
     <T_DM_REMOTE_TDOMAIN>
          <T_DM_REMOTE_TDOMAIN AccessPoint="TDOM1">
          <LocalAccessPoint>examples</LocalAccessPoint>
          <AccessPointId>TDOM1</AccessPointId>
          <Type>TDOMAIN</Type>
          <FederationURL>corbaloc:tgiop:simpapp/NameService</FederationURL>
          <FederationName>tuxedo.corba.remote</FederationName>
          <NWAddr>//127.0.0.1:4000</NWAddr>
     </T_DM_REMOTE_TDOMAIN>
     <T_DM_IMPORT
          ResourceName="//simpapp"
          LocalAccessPoint="examples"
          RemoteAccessPointList="TDOM1">
          <TranTime>600</TranTime>
     </T_DM_IMPORT>
```

```
</BDMCONFIG>
</WTC_CONFIG>
```

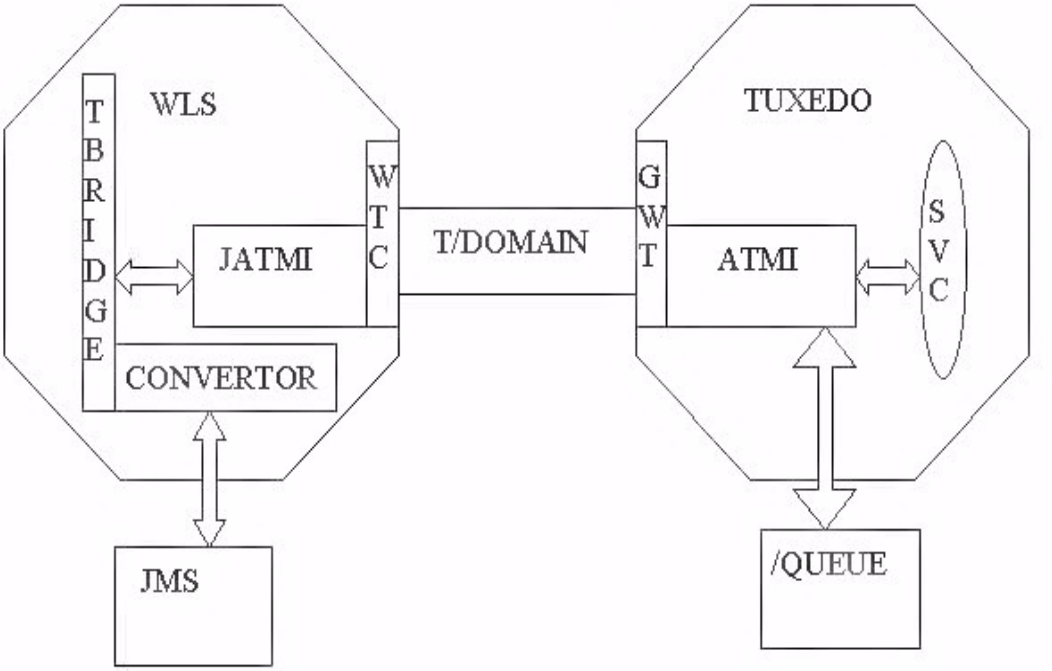
5 Configuring tBridge

Note: For more detailed reference information on the WebLogic Tuxedo Connector XML configuration file, elements and attributes, and the `wtc_config.dtd`, see Chapter 10, "The wtc_config.dtd."

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

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.

How tBridge connects JMS with Tuxedo

Note: All messages remain on the JMS queue until they have been acknowledged.

This section provides information on how JMS messages flow through the tBridge to Tuxedo queues and services.

1. A JMS client, such as a web enabled WLPI application, places a message to be processed by Tuxedo on a JMS Queue. If this message was part of a transaction, the transaction commits.
2. The message is removed from the JMS queue to be processed by the tBridge Converter.
3. The tBridge Converter checks the message type and converts supported JMS types to JATMI buffer types.
   - BytesMessage, TextMessage, XML are converted respectively to TypedCArray, TypedString, and TypedFML32. XML/FML translation is performed according to the `TranslateFML` attribute.
   - Translation errors are sent to the `wlsServerErrorDestination` queue and the message is acknowledged in the JMS session.
   - If an unrecognized JMS message is received: an appropriate error message is logged, the message is acknowledged, and then is discarded. This is considered a configuration error and the tBridge does not redirect the message to the error queue.
4. The converted message is sent to Tuxedo using the T/Domain gateway.
   - Messages with a redirect set to `JmsQ2TuxQ` use JATMI `tpenqueue` to deliver the message to a Tuxedo queue.
   - Messages with a redirect set to `JmsQ2TuxS` use JATMI `tpcall` to deliver the message to a Tuxedo service.
5. The `tpenqueue` is successful or `tpcall` is successful and the return results are placed in the replyQ. The message is acknowledged in the JMS session.

- If the `tpenqueue` or `tpcall` fails, tBridge delivers the message to the `wlsServerErrorDestination` queue and the message is acknowledged in the JMS session. If a `wlsServerErrorDestination` queue is not configured, the message is discarded and the tBridge processes the next available unacknowledged message.

How tBridge connects Tuxedo to JMS

Note: tBridge uses a transaction to prevent the loss of messages while transferring messages from Tuxedo /Q to a JMS queue.

This section provides information on how Tuxedo messages flow through the tBridge to a JMS queue using the `TuxQ2JmsQ` redirect.

1. tBridge polls the Tuxedo queue for available messages.
2. A Tuxedo service places a message on a Tuxedo queue.
3. tBridge uses JATMI `tpdequeue` to forward the message from Tuxedo and places the message in the JMS queue.
   - If a message cannot be redirected to a JMS queue for any reason after the specified retries have been exhausted, the message is put into the `tuxErrorDestination` queue within the same queue space as the Tuxedo queue.
   - If the tBridge is not able to put the message into the `tuxErrorDestination` queue for any reason, an error is logged and the message is lost.
   - If the `tuxErrorDestination` queue is not specified, the message is lost.

tBridge Limitations

The tBridge has the following limitations:

- Transactions are not used when retrieving messages from the JMS location and placing them on the Tuxedo queue or invoking a Tuxedo service.

- tBridge is thread intensive. A thread is used to transport each message from JMS queue to Tuxedo. A polling thread is required to monitor the configured Tuxedo queue.
- The XML/FML translator is intended to construct simple message structures. For more information on XML to FML conversion see, “FML32 Considerations” on page 6-7.

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

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.

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.

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
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="LDOM2"
     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>
     <ReplyQ>NO</ReplyQ>
     <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

Note: For more information on XML/FML conversion, see Chapter 6, "Using FML with WebLogic Tuxedo Connector."

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 FML32 into XML and placed on the `weblogic.jms.Tux2JmsQueue`.

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

WebLogic Tuxedo Connector supports multiple tBridge redirect instances. In many environments, using multiple redirect instances significantly improves application scalability and performance. However, it does randomizes the order in which messages are processed. Although priority mapping does not guarantee ordering, it does provides a mechanism to react to messages based on an assigned importance. If the order of delivery must be guaranteed, use a single tBridge redirect instance.

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.
- Tthe `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

Example:

The following `priorityMapping` represents the default mapping:

Listing 5-1 Example Configuration for Priority 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 `tpdequeue` during a `TuxQ2JmsQ` redirect.

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.

6 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/FML32 Translation

Overview of FML

Note: For more information about using FML, see Programming a BEA Tuxedo Application Using FML at http://e-docs.bea.com/tuxedo/tux80/atmi/fml01.htm.

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

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

The FML application program interface (API) is documented in the `weblogic.wtc.jatmi` package included in the Javadocs for WebLogic Server Classes.

FML Field Table Administration

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 in the `weblogic.wtc.jatmi` package. 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:

   a. `mkfldclass`

   b. `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.

Using the DynRdHdr Property for mkfldclass32 Class

WebLogic Tuxedo Connector provides a property that provides an alternate method to compile FML tables. You may need to use the `DynRdHdr` utility if:

- You are using very large FML tables and the `.java` method created by the mkfldclass32 class exceeds the internal Java Virtual Machine limit on the total complexity of a single class or interface.
- You are using very large FML tables and are unable to load the class created when compiling the `.java` method.

Use the following steps to use the `DynRdHdr` property when compiling your FML tables:

1. Convert the field table definition into Java source files.

   ```
   java -DDynRdHdr=Path_to_Your_FML_Table weblogic.wtc.jatmi.mkfld
   class32 userTable
   ```

The arguments for this command are defined as follows:

Attribute	Description
`-DDynRdHdr`	WebLogic Tuxedo Connector property used to compile an FML table.
`Path_to_Your_FML_Table`	Fully qualified path and the file name of your FML table.
`weblogic.wtc.jatmi.mkfldclass32`	This class is a utility function that reads an FML32 Field Table and produces a Java file which implements the FldTbl interface.
`userTable`	Name of the `.java` method created by the mkfldclass32 class.

2. Compile the *`userTable`* file using the following command:

   ```
   javac userTable.java
   ```

3. Add the *`userTable`*`.class` file to your application CLASSPATH.

4. Update the `T_DM_RESOURCES` section of your WebLogic Tuxedo Connector XML configuration file to reflect the fully qualified location of the *`userTable`*`.class` file.

5. Start your server. When the server is started, it will load the FML table using the location specified in the `T_DM_RESOURCES` of your WebLogic Tuxedo Connector XML configuration file.

Once you have created the *`userTable`*`.class` file, you can modify the FML table and deploy the changes without having to manually create an updated *`userTable`*`.class`. When the server is started, it will load the updated FML table using the location specified in the `T_DM_RESOURCES` of your WebLogic Tuxedo Connector XML configuration file. If the *`Path_to_Your_FML_Table`* attribute changes, you will need to use the preceding procedure to update your *`userTable`*`.java` and *`userTable`*`.class` files.

tBridge XML/FML32 Translation

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

The `<translateFML>` element is used to indicate if FML32 translation is performed on the message payload. There are two types of FML32 translation: FLAT and NO.

Using FLAT Translation

The message payload is translated using the WebLogic Tuxedo Connector internal FML32/XML translator. Fields are converted field-by-field values without knowledge of the message structure (hierarchy) and repeated grouping.

In order to convert an FML32 buffer to XML, the tBridge pulls each instance of each field in the FML32 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 FML32 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>
```

Using NO Translation

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.

FML32 Considerations

Remember to consider the following information when working with FML32:

- 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 6-2.
- The tBridge translator is capable of only "flat" or linear grouping. This means that information describing FML32 ordering is not maintained, therefore buffers that contain a series of repeating data could be presented in an unexpected fashion. For example, consider a FML32 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 FML32 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.

7 Connecting WebLogic Process Integrator and Tuxedo Applications

Note: For more information on how to integrate applications, see BEA WebLogic Integration at http://e-docs.bea.com/wlintegration/v2_0/index.html.

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.
- 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.

8 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`

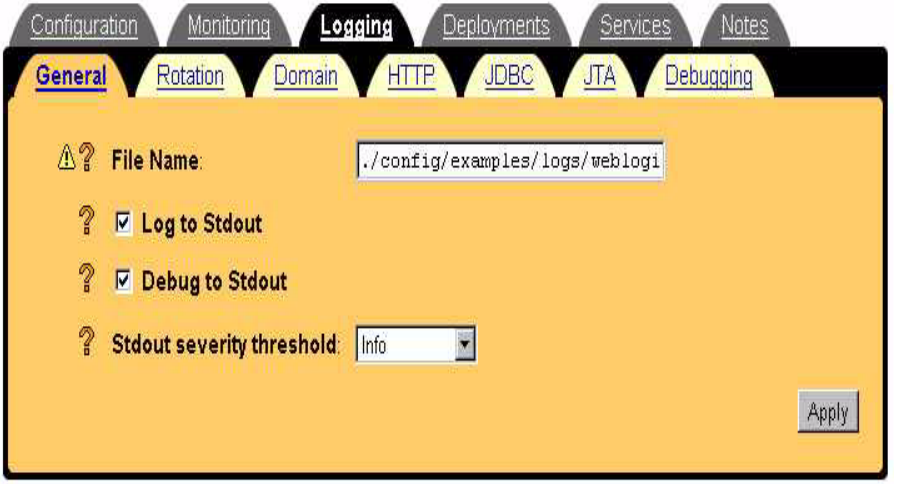
Use the following values to set the TraceLevel:

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
60000	CORBA_IO	CORBA input and output
65000	CORBA_EX	more CORBA 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 for the WebLogic Tuxedo Connector" on page 2-7.
- 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 `simpserv` 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.

9 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

Note: For more information on creating an XML Configuration file, see "Create a WebLogic Tuxedo Connector XML Configuration File" on page 2-3.

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.

Listing 9-1 simpapp BDMCONFIG.XML Configuration File

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

<!DOCTYPE WTC_CONFIG SYSTEM
"http://www.bea.com/servers/wls610/dtd/wtc_config.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>
```

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.

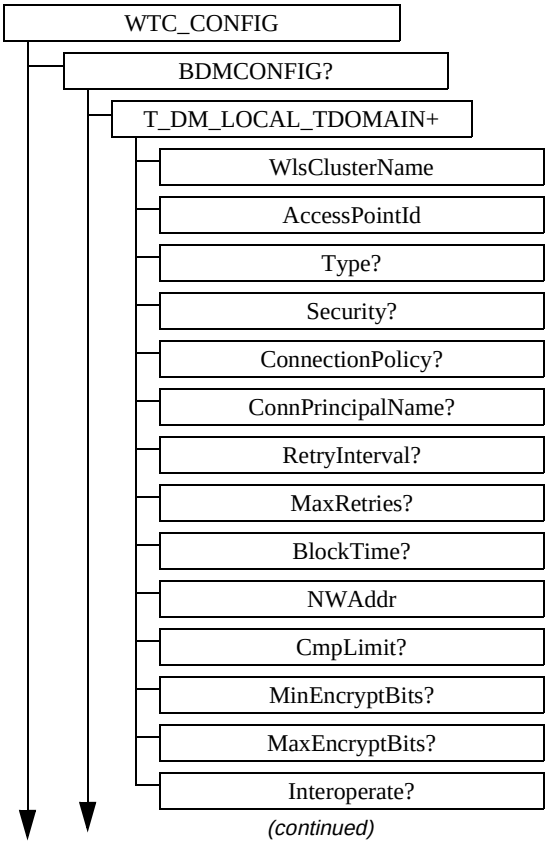
To validate the XML configuration file, enter the following command:

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

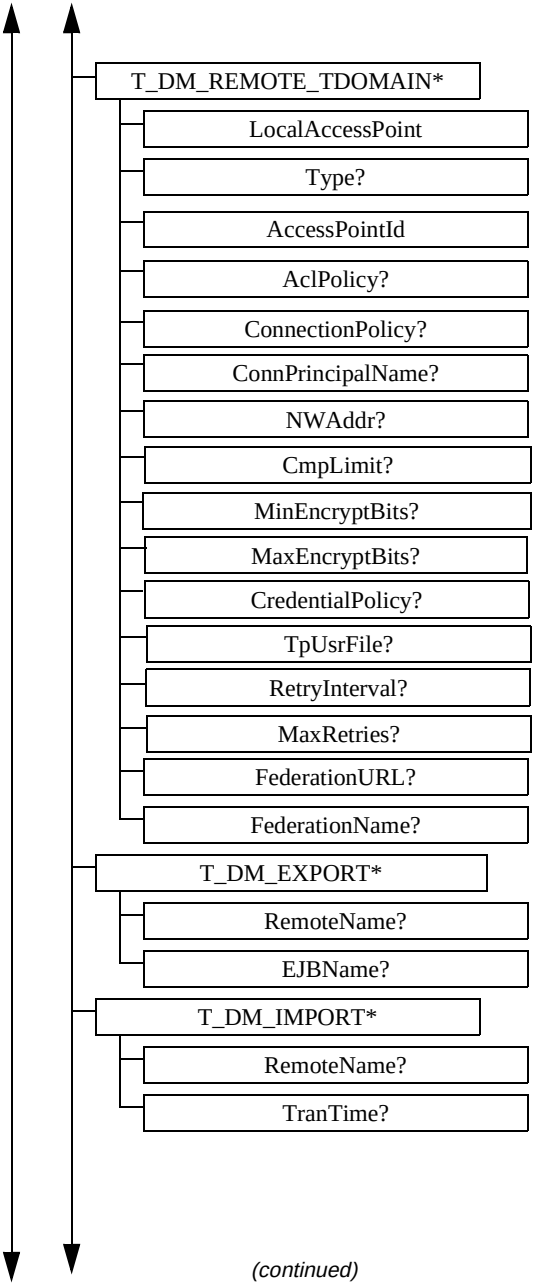
where ***your_XML_configuration_file*** is the name of your XML configuration file.

Element Hierarchy Diagram

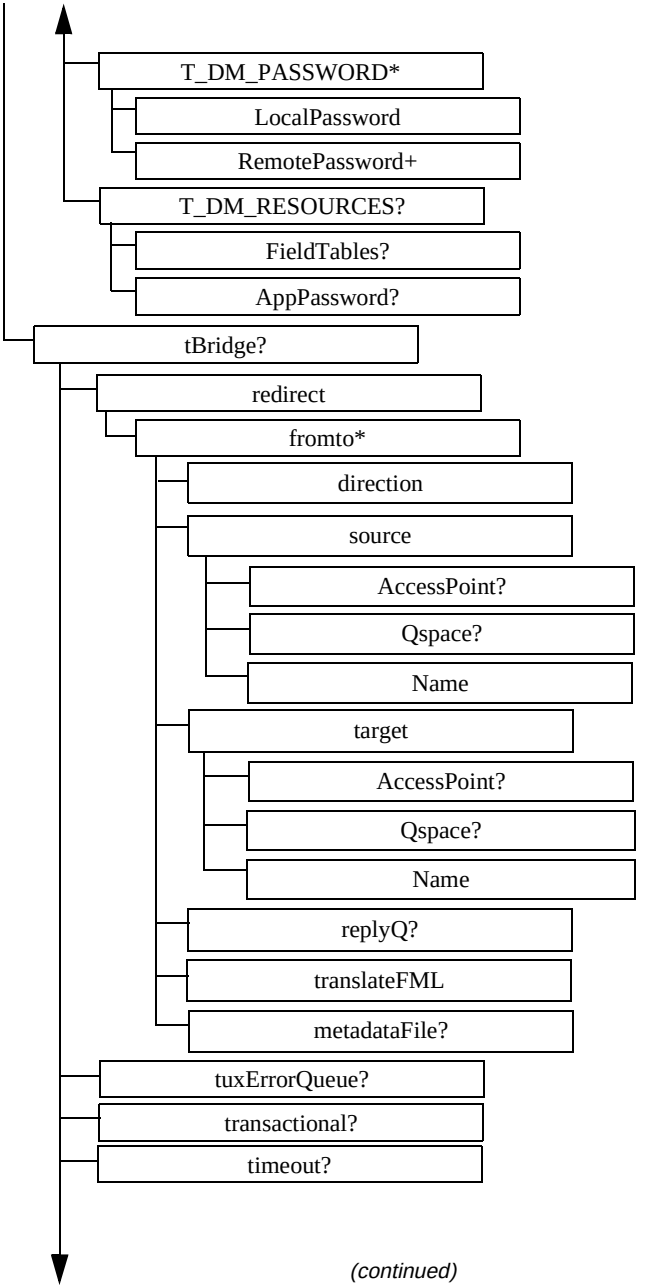
A WebLogic Tuxedo Connector configuration file consists of a series of XML elements. The `WTC_CONFIG` element is the top-level element, and all elements in the `WTC_CONFIG` are children of the `WTC_CONFIG` element. Child elements may have children themselves. For example, the `BDMCONFIG` element contains the `T_DM_LOCAL_TDOMAIN` child element, and the `T_DM_LOCAL_TDOMAIN` element contains child elements, as shown in the following diagram.



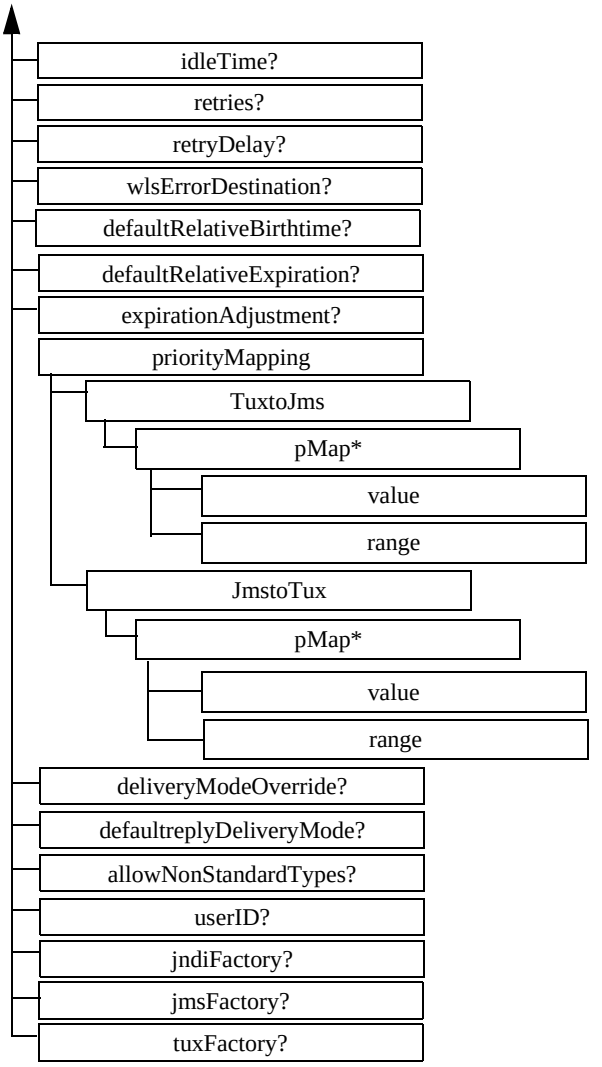
? = Optional
+ = One or more
*** = Zero or more**


T_DM_REMOTE_TDOMAIN*
LocalAccessPoint
Type?
AccessPointId
AclPolicy?
ConnectionPolicy?
ConnPrincipalName?
NWAddr?
CmpLimit?
MinEncryptBits?
MaxEncryptBits?
CredentialPolicy?
TpUsrFile?
RetryInterval?
MaxRetries?
FederationURL?
FederationName?
T_DM_EXPORT*
RemoteName?
EJBName?
T_DM_IMPORT*
RemoteName?
TranTime?

(continued)


T_DM_PASSWORD*
LocalPassword
RemotePassword+
T_DM_RESOURCES?
FieldTables?
AppPassword?
tBridge?
redirect
fromto*
direction
source
AccessPoint?
Qspace?
Name
target
AccessPoint?
Qspace?
Name
replyQ?
translateFML
metadataFile?
tuxErrorQueue?
transactional?
timeout?

(continued)


idleTime?
retries?
retryDelay?
wlsErrorDestination?
defaultRelativeBirthtime?
defaultRelativeExpiration?
expirationAdjustment?
priorityMapping
TuxtoJms
pMap*
value
range
JmstoTux
pMap*
value
range
deliveryModeOverride?
defaultreplyDeliveryMode?
allowNonStandardTypes?
userID?
jndiFactory?
jmsFactory?
tuxFactory?

10 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,
                FederationURL?,
                FederationName?,
                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 FederationURL (#PCDATA)>
<!ELEMENT FederationName (#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)>
```

11 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

Note: Tuxedo 6.5 users should set the security parameter to NONE.

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. This parameter only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or later software.

- If this element is not specified, the connection principal name defaults to the AccessPoint 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 `//mymachine:7001` is not valid if the WebLogic Server listening port is assigned to `//mymachine: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

Note: A MinEncryptBits value of 40 only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or later software.

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, 40, 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

Note: A MaxEncryptBits value of 40 only applies to domains of type TDOMAIN that are running BEA Tuxedo 7.1 or higher.

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, 40, 56, and 128.

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

Interoperate

Note: Optional. Specifies whether the local domain interoperates with remote domains that are based upon Tuxedo Release 6.5. Tuxedo 6.5 does not have the required security infrastructure to support security mapping. If you require security features and use the WebLogic Tuxedo Connector, you will need to upgrade to Tuxedo 7.1 or higher.

Valid values for this parameter are: Yes, No.

- Yes: Interoperate with Tuxedo 6.5
- No: Operates with domains Tuxedo 7.1 and higher.
- 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 11-7.

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 11-7.

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

FederationURL

The URL for a foreign name service that is federated into the JNDI.

If ommitted, the WebLogic Tuxedo Connector assumes that there is a CosNaming server in the foreign domain. The WebLogic Tuxedo Connector federates to the CosNaming server using TGIOP. It is possible to federate to non-CORBA service providers.

FederationName

The context at which to federate to a foreign name service. If ommitted, the federation point is *tuxedo.domains*.

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: //simpapp	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. It is the total time from the start of a transaction through the execution of the rollback or commit. 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: ■ 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

Note: The timeout for the transaction used to retrieve the message from the source location is larger than timeout. This parameter reflects the effective length of the timeout for the entire redirection.

Optional. Specifies the transaction timeout value (seconds) used to place a message on a target location. 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 must be 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`