Contents

About This Document

Audience ................................................................. x

e-docs Web Site ......................................................... x

How to Print the Document ........................................... x

Related Information ..................................................... x

Contact Us! .............................................................. xi

Documentation Conventions ........................................... xi

Introduction to WebLogic Tuxedo Connector

WebLogic Tuxedo Connector Overview ................................ 1-1

Key Functionality and Administrative Features ........................ 1-2

Known Limitations ......................................................... 1-2

How WebLogic Tuxedo Connector Differs from Jolt .................... 1-3

Platform Support ......................................................... 1-3

Licensing .............................................................. 1-3

Upgrading WebLogic Tuxedo Connector 6.x Applications ................ 1-4

Upgrading WebLogic Tuxedo Connector 7.0 Applications ................ 1-4

Configuring WebLogic Tuxedo Connector

Summary of Environment Changes and Considerations ................ 2-1

Tuxedo Changes ......................................................... 2-1

WebLogic Server Changes ............................................ 2-2

Administration and Programming ...................................... 2-2
WebLogic Tuxedo Connector Administration

Configuring the Connections Between Access Points ........................................ 3-2
  How to Request a Connection at Boot Time (On Startup) ............................ 3-3
    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) ..................................... 3-4
    How to use LOCAL Connection Policy .......................................... 3-4
Dynamic Administration of Connections ......................................................... 3-5
Configuring Failover and Failback .............................................................. 3-8
  Prerequisite to Using Failover and Failback ........................................ 3-8
  How to Configure Failover .............................................................. 3-8
    How to Configure Failback ....................................................... 3-10
  How to Configure Link-level Failover ............................................... 3-10
    Sample Link-level Failover Configuration ..................................... 3-10
Configuring for TypedMBString Support ........................................ 3-11
Authentication of Remote Access Points ........................................ 3-12
  Configuring a Password Configuration ..................................... 3-13
  Generating Encrypted Passwords .......................................... 3-13
  Usage ............................................................................... 3-14
Examples .............................................................................. 3-14
  Local Passwords .................................................................. 3-14
  Remote Passwords ................................................................ 3-15
  App Passwords .................................................................... 3-15
User Authentication ................................................................. 3-15
  ACL Policy is LOCAL ......................................................... 3-16
  ACL Policy is GLOBAL ....................................................... 3-16
    Remote Access Point Credential Policy is GLOBAL .................. 3-16
    Remote Access Point Credential Policy is LOCAL .................. 3-17
User Authentication for Tuxedo 6.5 ............................................. 3-17
How to Configure WebLogic Tuxedo Connector to Provide Security between Tuxedo and WebLogic Server ......................................................... 3-17
TpUsrFile Plug-in ................................................................. 3-17
  Configuring the Local Tuxedo Access Point for the TpUsrFile Plug-in . . . . . . 3-17
  Configure the Remote Tuxedo Access Point for the TpUsrFile Plug-in .... 3-18
LDAP Plug-in ................................................................. 3-19
  Implementing Single Point Security Administration .................. 3-19
  Configure the Local Tuxedo Access Point for the LDAP Plug-in .... 3-19
  Configure the Remote Tuxedo Access Point for the LDAP Plug-in .... 3-19
Custom Plug-in ................................................................. 3-20
  Configure the Local Tuxedo Access Point for the Custom Plug-in .... 3-20
  Configure the Remote Tuxedo Access Point for the Custom Plug-in .... 3-20
Anonymous Users ................................................................. 3-21
Anonymous Users and CORBA Services ....................................... 3-21
Link-Level Encryption ................................................................. 3-22

Administration of CORBA Applications
How to Configure WebLogic Tuxedo Connector for CORBA Service Applications . . 4-1
  Example WTC Service and Tuxedo UBB Files ................................. 4-2
How to Administer and Configure WebLogic Tuxedo Connector for Inbound RMI-IIOP .4-4
  Configuring Your WTC Service for Inbound RMI-IIOP ..................... 4-5
  Administering the Tuxedo Application Environment ........................ 4-5
  Guidelines About Using Your Server Name as an Object Reference .... 4-7
How to Configure WebLogic Tuxedo Connector for Outbound RMI-IIOP . . . . . . . . 4-7
  Example Outbound RMI-IIOP Configuration ................................ 4-8

How to Manage WebLogic Tuxedo Connector in a Clustered Environment
  WebLogic Tuxedo Connector Guidelines for Clustered Environments ........ 5-1
  How to Configure for Clustered Nodes .................................... 5-2
  Limitations for Clustered Nodes ........................................... 5-2
  How to Configure OutBound Requests to Tuxedo Domains ............... 5-2
  Example Clustered WebLogic Tuxedo Connector Configuration ......... 5-2
  How to Configure Inbound Requests from Tuxedo Domains ............. 5-9
  Load Balancing ................................................................. 5-9
  Fail Over ............................................................................. 5-9

How to Configure the Tuxedo Queuing Bridge
  Overview of the Tuxedo Queuing Bridge .................................... 6-1
  How Tuxedo Queuing Bridge connects JMS with Tuxedo ............... 6-2
  How Tuxedo Queuing Bridge connects Tuxedo to JMS .................. 6-3
  Tuxedo Queuing Bridge Limitations ....................................... 6-4
Configuring the Tuxedo Queuing Bridge .................................................. 6-4
Starting the Tuxedo Queuing Bridge .................................................. 6-4
Error Logging ................................................................. 6-5
Tuxedo Queuing Bridge Connectivity ............................................. 6-5
Example Connection Type Configurations ......................................... 6-5
Example JmsQ2TuxQ Configuration .................................................. 6-5
Example TuxQ2JmsQ Configuration .................................................. 6-6
Example JmsQ2TuxS Configuration .................................................. 6-7
Priority Mapping ................................................................. 6-9
Error Queues ................................................................. 6-9
WLS Error Destination .......................................................... 6-10
Unsupported Message Types ...................................................... 6-10
Tuxedo Error Queue ............................................................ 6-10
Limitations ................................................................. 6-10

Connecting WebLogic Integration and Tuxedo Applications

Synchronous WebLogic Integration-to-Tuxedo Connectivity ................. 7-1
   Defining Business Operations ............................................ 7-2
   Invoking an eLink Adapter ............................................. 7-2
   Define Exception handlers ............................................ 7-2
Synchronous Non-Blocking WebLogic Integration-to-Tuxedo Connectivity 7-2
Asynchronous WebLogic Integration-to-Tuxedo Connectivity ............... 7-3
Asynchronous Tuxedo /Q-to-WebLogic Integration Connectivity .......... 7-3
Bi-directional Asynchronous Tuxedo-to-WebLogic Integration Connectivity 7-3

Troubleshooting The WebLogic Tuxedo Connector

Monitoring the WebLogic Tuxedo Connector .................................... 8-1
   Set Trace Levels ......................................................... 8-1
Enable Debug Mode .................................................. 8-2
Enable a User Data Dump ............................................. 8-3
Frequently Asked Questions .......................................... 8-3
What does this EJB Deployment Message Mean? ......................... 8-3
How do I Start the Connector? ......................................... 8-4
How do I Start the Tuxedo Queuing Bridge? ............................ 8-4
How do I Assign a WTC Service to a Server? .......................... 8-4
How do I Resolve Connection Problems? ............................. 8-4
How do I Migrate from Previous Releases? .......................... 8-5
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, “WebLogic Tuxedo Connector Administration,” provides configuration information about the WebLogic Tuxedo Connector.
- Chapter 4, “Administration of CORBA Applications,” provides information on how to administer CORBA applications.
- Chapter 5, “How to Manage WebLogic Tuxedo Connector in a Clustered Environment,” provides information on how to use WebLogic Tuxedo Connector in a clustered environment.
- Chapter 6, “How to Configure the Tuxedo Queuing Bridge,” provides information on tBridge functionality and configuration.
- Chapter 7, “Using FML with WebLogic Tuxedo Connector,” discusses the Field Manipulation Language (FML) and describes how the WebLogic Tuxedo Connector uses FML.
• Chapter 8, “Troubleshooting The WebLogic Tuxedo Connector,” provides WebLogic Tuxedo Connector troubleshooting information.

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, 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 or go directly to the WebLogic Server Product Documentation page at http://e-docs.bea.com.

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.


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.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+Tab</td>
<td>Keys you press simultaneously.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Emphasis and book titles.</td>
</tr>
</tbody>
</table>
### 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:*

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

**italic text** | Variables in code.

*Example:*

```java
String CustomerName;
```

**UPPERCASE TEXT** | Device names, environment variables, and logical operators.

*Examples:*

```text
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}
```
<table>
<thead>
<tr>
<th>Convention</th>
<th>Usage</th>
</tr>
</thead>
</table>
| . . .      | 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 |
| .          | Indicates the omission of items from a code example or from a syntax line.  
  .          |
Introduction to WebLogic Tuxedo Connector

The following sections summarize the concepts and functionality of WebLogic Tuxedo Connector for this release of WebLogic Server:

- WebLogic Tuxedo Connector Overview
- Key Functionality and Administrative Features
- Known Limitations
- How WebLogic Tuxedo Connector Differs from Jolt
- Platform Support
- Licensing
- Upgrading WebLogic Tuxedo Connector 6.x Applications
- Upgrading WebLogic Tuxedo Connector 7.0 Applications

**WebLogic Tuxedo Connector Overview**

The WebLogic Tuxedo Connector provides interoperability between WebLogic Server applications and Tuxedo services. The connector 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.
- 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 provide interoperability between Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP) applications and Tuxedo CORBA remote objects.
- Ability to use WebLogic Integration to manage workflow across Tuxedo ATMI services.
- 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.
- 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:
Support for runtime MBean exists, so the configuration can be modified after deployment. There is an exception in tBridge. Both tBridge Globals and tBridge Redirect changes will not be in effect until WTC is undeployed and redeployed.

- Does not support inbound TGIOP in clustered environments.
- Does not support Tuxedo 8.1 GWTDOMAIN TCP and APP KeepAlive functionality.
- WebLogic Tuxedo Connector does not support Tuxedo 6.5 running on OS/390 platform.

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.

Platform Support

See our Platforms Support page at http://e-docs.bea.com/platform/suppconfigs/index.html for the most accurate and current information regarding platform support.

Licensing

**Note:** For more information on WebLogic Server licensing information, see Installing and Updating License Files at http://e-docs.bea.com/wls/docs90/../../common/docs90/install/license.html.

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.
Upgrading WebLogic Tuxedo Connector 6.x Applications

You must make some changes in your WebLogic Tuxedo Connector 6.x applications (including WebLogic Tuxedo Connector 1.0) to use them with WebLogic Server 9.0. For detailed information on the administration and programming changes required to upgrade to WebLogic Tuxedo Connector in WebLogic Server 9.0, see Upgrading WebLogic Application Environments at http://e-docs.bea.com/wls/docs90/../../common/docs90/upgrade/index.html.

Upgrading WebLogic Tuxedo Connector 7.0 Applications

You may want to make some changes in your WebLogic Tuxedo Connector 7.0 RMI-IIOP applications to use them with WebLogic Server 9.0. For detailed information, see Upgrading WebLogic Application Environments at http://e-docs.bea.com/wls/docs90/../../common/docs90/upgrade/index.html.
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

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 Using the Tuxedo Domains Component.

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

The following sections describe WebLogic Server changes required to use the WebLogic Tuxedo Connector:

- Administration and Programming
- WebLogic Server Threads

Administration and Programming

WebLogic Server users need to make the following environment changes:

- Create Java clients or servers. For more information on creating WebLogic Tuxedo Connector clients or servers, see the WebLogic Tuxedo Connector Programmer’s Guide.

- Configure the WebLogic Tuxedo Connector using the WebLogic Server console, command-line interface, or WLST. For more information on how to configure the WebLogic Tuxedo Connector, see “Configuring WebLogic Tuxedo Connector for Your Applications” on page 2-2.

- If the WebLogic Tuxedo Connector ACL Policy is set to Local, access to local services does not depend on the CredentialPolicy. The Tuxedo remote domain DOMAINID must be authenticated as a local WebLogic Server user. For more information, see “User Authentication” on page 3-15.

WebLogic Server Threads

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 attribute 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: Deciding when to target a WTC Service is very important. Support for runtime MBean exists, so the configuration can be modified after deployment. There is an exception in tBridge. Both tBridge Globals and tBridge Redirect changes will not be in effect until WTC is undeployed and redeployed.

This section provides information on how to configure the WebLogic Tuxedo Connector to allow WebLogic Server applications and Tuxedo applications to interoperate.
WebLogic Tuxedo Connector MBean Classes

The WebLogic Tuxedo Connector uses MBeans to describe connectivity information and security protocols 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 configuration parameters are stored in the WebLogic Server config.xml file. The following table lists the MBean types used to configure WebLogic Tuxedo Connector:

<table>
<thead>
<tr>
<th>MBean Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTCServer</td>
<td>Parent MBean containing the interoperability attributes required for a connection between WebLogic Server and Tuxedo. Defines your WTC Service when configured using the Administration Console.</td>
</tr>
</tbody>
</table>
| WTCLocalTuxDom | Provides configuration information to connect available remote Tuxedo domains to a WTC Service. You must configure at least one local Tuxedo access point. Defines your Local Tuxedo Access Points when configured using the Administration Console.  
**Note:**  
Because of dynamic configuration, you can create and deploy an empty WTC Service.
Configuring WebLogic Tuxedo Connector Using the Administration Console

The Administration Console allows you to configure, manage, and monitor WebLogic Tuxedo Connector connectivity. To display the tabs that you use to perform these tasks, complete the following procedure:

<table>
<thead>
<tr>
<th>MBean Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTCRemoteTuxDom</td>
<td>Provides configuration information to connect a WTC Service to available remote Tuxedo domains. You may configure multiple remote domains. Defines your Remote Tuxedo Access Points when configured using the Administration Console.</td>
</tr>
<tr>
<td>WTCExport</td>
<td>Provides information on services exported by a local Tuxedo access point. Defines your Exported Services when configured using the Administration Console.</td>
</tr>
<tr>
<td>WTCImport</td>
<td>Provides information on services imported and available on remote domains. Defines your Imported Services when configured using the Administration Console.</td>
</tr>
<tr>
<td>WTCResources</td>
<td>Specifies global field table classes, view table classes, and application passwords for domains. Defines your Resources when configured using the Administration Console. Support for MBSTRING is provided using RemoteMBEncoding and MBEncodingMapFile attributes.</td>
</tr>
<tr>
<td>WTCPassword</td>
<td>Specifies the configuration information for inter-domain authentication. Defines your Passwords when configured using the Administration Console.</td>
</tr>
<tr>
<td>WTCtBridgeGlobal</td>
<td>Specifies global configuration information for the transfer of messages between WebLogic Server and Tuxedo. Defines your Tuxedo Queuing Bridge when configured using the Administration Console.</td>
</tr>
<tr>
<td>WTCtBridgeRedirect</td>
<td>Specifies the source, target, direction, and transport of messages between WebLogic Server and Tuxedo. Defines your Tuxedo Queuing Bridge Redirects when configured using the Administration Console.</td>
</tr>
</tbody>
</table>
1. Start the Administration Console.
2. Locate the Interoperability node in the left pane, then expand the WTC Service.
3. Create or modify the WTC Server you want to configure.
4. Follow the instructions in the Online Help. For links to the Online Help, see Table 2-1.

The following table shows the connectivity tasks, listed in typical order in which you perform them. You may change the order; just remember you must configure an object before associating or assigning it.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1    | Creating a WTC Service  
      | On the General tab in the right pane, you set the attributes for Name and Deployment Order. |
| 2    | Creating a Local Tuxedo Access Point  
      | Set the attributes that describe your local Tuxedo access point in the General, Connections, and Security tabs. You must configure at least one local Tuxedo access point.  
      | **Note:** Because of dynamic configuration, you can create and deploy an empty WTC Service. |
| 3    | Creating a Remote Tuxedo Access Point  
      | Set the attributes that describe your remote Tuxedo domains in the Local APs tab. |
| 4    | Creating Exported Services  
      | Set the attributes that describe your exported WebLogic Server services in the Exported tab. |
| 5    | Creating Imported Services  
      | Set the attributes that describe your imported Tuxedo services in the Imported tab. |
| 6    | Creating a Password Configuration  
      | Set the attributes that describe your passwords in the Password tab. |
| 7    | Creating a Resource  
      | Set the attributes that describe your WebLogic Tuxedo Connector resources in the Resources tab. |
| 8    | Creating a Tuxedo Queuing Bridge Connection  
      | Set the global configuration information for the transfer of messages between WebLogic Server and Tuxedo. |
Configuring WebLogic Tuxedo Connector Using the Command-Line Interface

The command-line interface provides a way to create and manage WebLogic Tuxedo Connector connections. For information on how to use the command-line interface, see WebLogic Server Scripting Tool at http://e-docs.bea.com/wls/docs90/config_scripting/index.html.

Set the WebLogic Server Environment

You need to set the environment of your WebLogic Server application by running the setExamplesEnv script located at WL_HOME\samples\domains\examples.

- Windows users: run setExamplesEnv.cmd
- UNIX users: run setExamplesEnv.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. Copy the setExamplesEnv script located at WL_HOME\samples\domains\examples to your application directory.

2. Edit the setExamplesEnv script with a text editor, such as vi.
   - Windows users: edit setExamplesEnv.cmd
   - UNIX users: edit setExamplesEnv.sh

3. Save the file.

### Table 2-1  WebLogic Tuxedo Connector Configuration Tasks

<table>
<thead>
<tr>
<th>Task #</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Creating a tBridge Direction</td>
</tr>
<tr>
<td>10</td>
<td>Assign a WTC Service to a Server</td>
</tr>
</tbody>
</table>
How to Set WebLogic Tuxedo Connector Properties

PasswordKey, and encoding are WebLogic Server Properties. If you need to set these properties, update the JAVA_OPTIONS variable in your server start script. Example:

```
JAVA_OPTIONS=-Dweblogic.wtc.PasswordKey=mykey
```

**Set PasswordKey**

Note: For more information on PasswordKey, see “Configuring a Password Configuration” on page 3-13.

Use PasswordKey to specify the key used by the `weblogic.wtc.gwt.genpassword` utility to encrypt passwords:

```
JAVA_OPTIONS=-Dweblogic.wtc.PasswordKey=mykey
```

where *mykey* is the key value.

**Set encoding**

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.


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

Set Dumping of User Data

To enable dumping of user data, add the following line to the java.weblogic.Server command.

JAVA_OPTIONS=-Dweblogic.debug.DebugWTCUData=true

Enabling this causes user data to be dumped after the connection is connected. If no other debugging properties are enabled, then this will be the only WTC information dumped, except normal WTC error/informational messages. The dump is available in the WLS server log file. The dump has the following format.

- For outbound messages
  
  Outbound UDATA: buffer type (<type>, <subtype>)
  ++++ User Data(size) ++++
  ......

- For inbound messages
  
  Inbound UDATA: buffer type (<type>, <subtype>)
  ++++ User Data(size) ++++
  ......

For example, a WLS client sends data “strings” in a STRING typed buffer and the Tuxedo TOUPPER service converts it to “STRINGS”. The WLS server log shows the following dump.

```
Outbound UDATA: buffer type (STRING, null)
+++++ User Data(16) ++++
00 00 00 07 73 74 72 69 6E 67 73 00 00 00 00 ....strings.....
+++++ END ++++
Outbound UDATA: buffer type (String, null)
+++++ User Data(12) ++++
00 00 00 07 53 54 52 54 49 4E 47 53 00 ....STRINGS.
+++++ END ++++
```
System Level Debug Settings

Because `traceLevel` is deprecated, use system debugging. By default all the debug tracing is off. Use the following settings to turn debug trace on.

- For tracing WTC-CORBA runtime
  
  `-Dweblogic.debug.DebugWTCCorbaEx=true`

- For tracing WTC-GWT runtime
  
  `-Dweblogic.debug.DebugWTCGwtEx=true`

- For tracing WTC-JATMI runtime
  
  `-Dweblogic.debug.DebugWTCJatmiEx=true`

- For tracing WTC-tBridge runtime
  
  `-Dweblogic.debug.DebugWTCtBridgeEx=true`

- For tracing WTC Configuration runtime
  
  `-Dweblogic.debug.DebugWTCConfig=true`

WebLogic Tuxedo Connector Configuration Guidelines

Use the following guidelines when configuring WebLogic Tuxedo Connector:

- You may have more than one WTC Service in your configuration.
- You cannot target 2 or more WTC Services to the same server. A server can only be a target for one WTC Service.
- Any configuration changes implemented in a WTC Service after a target server is selected will not be updated in the target server instance. You must remove the WTC Service from the server and then add the updated WTC Service add to the target server. For more information on selecting a target server, see Assign a WTC Service to a Server.
CHAPTER 3

WebLogic Tuxedo Connector Administration

Note: For more information on the WebLogic Server management, including the WebLogic Tuxedo Connector, see the WebLogic Server MBean Reference at http://e-docs.bea.com/wls/docs90/wlsmbbeanref/index.html.

The following sections describe how to establish connectivity and provide security between WebLogic Server applications and Tuxedo environments. WebLogic Tuxedo Connector uses attributes that are analogous to the interoperability attributes required for the communication between Tuxedo access points.

The following sections provide WebLogic Tuxedo Connector configuration information:

- Configuring the Connections Between Access Points
- Dynamic Administration of Connections
- Configuring Failover and Failback
- Configuring for TypedMBString Support
- Authentication of Remote Access Points
- User Authentication
- How to Configure WebLogic Tuxedo Connector to Provide Security between Tuxedo and WebLogic Server
- Link-Level Encryption
Configuring the Connections Between Access Points

Several options can specify the conditions under which an access point tries to establish a connection with a remote access point. Specify these conditions using the ConnectionPolicy attribute in the Connections tab of the Local Tuxedo Access Points and Remote Tuxedo Access Points configurations of your WTC Service. 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 imported services associated with each remote access point.

The WTC local access point has three connection policies: **ON_DEMAND**, **INCOMING_ONLY**, and **ON_STARTUP**. The default is **ON_DEMAND**.

The WTC remote access point has four connection policies: **ON_DEMAND**, **INCOMING_ONLY**, **ON_STARTUP**, and **LOCAL**. The default is **LOCAL**. When you specify **LOCAL** for the remote access point connection policy setting, the local access point connection policy is used. The remote access point connection policy takes precedence over the local access point connection policy.

The local access point connection policy works as a backup for remote access point connection. At the WTC startup, WTC processes through all the remote access point definitions and decides the actual connection policy similar to the following table.

<table>
<thead>
<tr>
<th>Local Access Point Setting</th>
<th>Remote Access Point Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>ON_DEMAND</strong></td>
</tr>
<tr>
<td><strong>ON_DEMAND</strong></td>
<td><strong>ON_DEMAND</strong></td>
</tr>
<tr>
<td><strong>ON_STARTUP</strong></td>
<td><strong>ON_DEMAND</strong></td>
</tr>
<tr>
<td><strong>INCOMING_ONLY</strong></td>
<td><strong>ON_DEMAND</strong></td>
</tr>
</tbody>
</table>
The following information clarifies the interaction between the connection policy for the local access point, the connection policy for the remote access point, and the settings of these parameters at the remote domain.

<table>
<thead>
<tr>
<th>Local System’s Effective Connection Policy</th>
<th>Remote System’s Effective Connection Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON_DEMAND</td>
<td>ON_DEMAND when both are up</td>
</tr>
<tr>
<td>ON_DEMAND</td>
<td>ON_STARTUP when both are up</td>
</tr>
<tr>
<td>ON_STARTUP</td>
<td>ON_DEMAND from either</td>
</tr>
<tr>
<td>ON_STARTUP</td>
<td>ON_STARTUP when both are up</td>
</tr>
<tr>
<td>INCOMING_ONLY</td>
<td>ON_DEMAND when both are up</td>
</tr>
<tr>
<td>INCOMING_ONLY</td>
<td>INCOMING_ONLY from remote</td>
</tr>
<tr>
<td>INCOMING_ONLY</td>
<td>manual connect only when both are up</td>
</tr>
</tbody>
</table>

**How to Request a Connection at Boot Time (On Startup)**

A policy of On Startup means that an access point attempts to establish a connection with its remote 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, set the ConnectionPolicy attribute in the Connections tab of your local Tuxedo access point to On Startup.

**How to Configure RetryInterval**

You can control the frequency of automatic connection attempts by specifying the interval (in seconds) during which the access point should wait before trying to establish a connection again. The minimum value is 0; the default value is 60, and maximum value is 2147483647.

**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 an access point tries to establish connections to remote 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 access point automatically.

- If you set `MaxRetries` to a number, the access point tries to establish a connection the specified number of times before quitting.

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

### Table 3-1 Example Settings of MaxRetries and RetryInterval Parameters

<table>
<thead>
<tr>
<th>If you set ...</th>
<th>Then ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ConnectionPolicy: On Startup</strong>&lt;br&gt;RetryInterval: 30&lt;br&gt;MaxRetries: 3</td>
<td>The access point makes 3 attempts to establish a connection, at 30 seconds intervals, before quitting.</td>
</tr>
<tr>
<td><strong>ConnectionPolicy: On Startup</strong>&lt;br&gt;MaxRetries: 0</td>
<td>The access point attempts to establish a connection at initialization time but does not retry if the first attempt fails.</td>
</tr>
<tr>
<td><strong>ConnectionPolicy: On Startup</strong>&lt;br&gt;RetryInterval: 30</td>
<td>The access point attempts to establish a connection every 30 seconds until a connection is established.</td>
</tr>
</tbody>
</table>

### How to Request Connections for Client Demands (On Demand)

**Note:** If the `ConnectionPolicy` is not specified for the local access point, 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 start connection command.

### Accepting Incoming Connections (Incoming Only)

A connection policy of **Incoming Only** means that an access point does not establish a connection to remote access points upon starting. The access point is available for incoming connection requests from remote access points.

### How to use LOCAL Connection Policy

**Note:** A `ConnectionPolicy` of **LOCAL** is not valid for local access points.
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 access point ConnectionPolicy is not defined, the system uses the setting specified by the associated local access point.

Dynamic Administration of Connections

You can dynamically list, start, and stop individual connections using the WLST scripting language. The listConnectionsConfigured() attribute lists the configured connections, startConnection() attribute allows you to start an individual connection, and stopConnection() attribute allows you to stop individual connections. For information on how to administer individual connections dynamically, refer to the WebLogic Server Scripting Tool at http://e-docs.bea.com/wls/docs90/config_scripting/index.html.

- Listing Connections

Using the WebLogic Scripting Tool (WLST), you can dynamically list the connections for a domain with the listConnectionsConfigured() attribute. When you run

cmo.listConnectionsConfigured(), a reference to an array of DSessConnInfo structures is returned. It is convenient to save this in a local WLST variable, such as

wls:/mydomain/serverRuntime/WTCRuntime/WTCService>
r=cmo.listConnectionsConfigured()

Each DSessConnInfo instance has a local access point ID, remote access point ID, and status (boolean, true = connected, false = not connected). For example,

wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].getLocalAccessPointId() WLSDOM
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].getRemoteAccessPointId() TUXDOM
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].isConnected() 0

- Starting Connections

Using the WebLogic Scripting Tool (WLST), you can dynamically start individual connections for an access point with the startConnection() attribute.

To start a connection between a local and a remote access point, specify the access point IDs in the arguments. For example,
To start a connection between a local and all associated remote access points, specify the local access point ID in the argument. For example,

```java
com.startConnection('WLSDOM', 'TUXDOM')
```

**Stopping Connections**

Using the WebLogic Scripting Tool (WLST), you can dynamically stop individual connections for an access point with the `stopConnection()` attribute.

To stop a connection between a local and a remote access point, specify the access point IDs in the arguments. For example,

```java
com.stopConnection('WLSDOM', 'TUXDOM')
```

To stop all connections involving a given local access point, specify the local access point ID in the argument. For example,

```java
com.stopConnection('WLSDOM')
```

The following code list is an example of dynamically listing, starting and stopping connections using WLST.

### Listing 3-1  Dynamically List, Start, and Stop Connections

```java
java weblogic.WLST
wls:/offline> connect('weblogic', 'weblogic')
wls:/mydomain/serverConfig> cd('WTCServers')
wls:/mydomain/serverConfig/WTCServers> cd('myWTC')
wls:/mydomain/serverConfig/WTCServers/myWTC> cd('LocalTuxDoms')
wls:/mydomain/serverConfig/WTCServers/myWTC/LocalTuxDoms> ls()
  dr--  TDOM2
wls:/mydomain/serverConfig/WTCServers/myWTC/LocalTuxDoms> cd('..\..\..')
wls:/mydomain/serverConfig> serverRuntime()
wls:/mydomain/serverRuntime> cd('WTCRuntime')
wls:/mydomain/serverRuntime/WTCRuntime> cd('WTCService')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> r=com.listConnectionsConfigured()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].getLocalAccessPointId()
  TDOM2
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].getRemoteAccessPointId()
  TDOM1
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].isConnected()
  0
```
Dynamic Administration of Connections

```plaintext
wls:/mydomain/serverRuntime/WTCRuntime/WTCService>
cmo.startConnection('TDOM2','TDOM1')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService>
r=cmo.listConnectionsConfigured()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].isConnected()
1
wls:/mydomain/serverRuntime/WTCRuntime/WTCService>
cmo.stopConnection('TDOM2','TDOM1')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService>
r=cmo.listConnectionsConfigured()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print r[0].isConnected()
0
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> disconnect()
wls:/offline> exit()
```

- **Modifying Configuration Attributes**

  Using the WebLogic Scripting Tool (WLST), you can dynamically modify a configuration attribute.

  The following code listing is an example that modifies the `setInteroperate()` attribute.

**Listing 3-2  Modifying Configuration Attributes**

```plaintext
java weblogic.WLST
wls:/offline> connect('weblogic','weblogic')
```

```plaintext
wls:/mydomain/serverConifg> edit()
```

```plaintext
wls:/mydomain/edit> startEdit()
```

```plaintext
wls:/mydomain/edit> cd("WTCServers/myWTC")
wls:/mydomain/edit/WTCServers/myWTC> cd("LocalTuxDoms")
wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms> cd("TDOM2")
wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms/TDOM2>
cmo.setInteroperate("Yes")
wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms/TDOM2> validate()
wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms/TDOM2> showChanges()
```

Changes that are in memory and saved to disc but not yet activated are:

- **MBean Changed** : mydomain:Name=TDOM2,Type=WTCLocalTuxDom,WTCServer=myWTC
- **Operation Invoked** : modify
- **Attribute Modified** : Interoperate
- **Attributes Old Value** : No
- **Attributes New Value** : Yes
- **Server Restart Required** : false
Configuring Failover and Failback

**Note:** In the Tuxedo T/Domain, there is a limit of two (2) backup remote access points. The WebLogic Tuxedo Connector has no limit to the number of backup access points allowed to be configured for a service.

WebLogic Tuxedo Connector provides a failover mechanism that transfers requests to alternate remote access points when a failure is detected with a primary remote access point. It also provides failback to the primary remote access point when that access point is restored. This level of failover/failback depends on connection status. The access point must be configured with a connection policy of **On Startup** or **Incoming Only** to enable failover/failback.

Prerequisite to Using Failover and Failback

To use 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 failback as it operates on the assumption that the remote access point 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 access points that you have specified with the Tuxedo **RDOM** parameter.

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

How to Configure Failover

To support failover, you must specify the remote access points responsible for executing a particular service. You must specify the following in your WTC Service:

- Create Remote Tuxedo Access Points configurations for each remote access point.
- Create Imported Services configurations that specify the service provided by each remote access point.
Suppose a service, TOUPPER, is available from two remote access points: TDOM1 and TDOM3. Your WTC Service would include two Remote Tuxedo Access Point configurations and two Imported Services configurations in your WTC Service. The WTC Service defined in the config.xml file would contain the following:

```xml
<wtc-server>
  <name>WTCsimpapp</name>
  <wtc-local-tux-dom>
    <access-point>TDOM2</access-point>
    <access-point-id>TDOM2</access-point-id>
    <connection-policy>ON_DEMAND</connection-policy>
    <interoperate>no</interoperate>
    <nw-addr>123.123.123.123:5678</nw-addr>
    <name>myLoc1TuxDom</name>
    <security>NONE</security>
  </wtc-local-tux-dom>
  <wtc-remote-tux-dom>
    <access-point>TDOM1</access-point>
    <access-point-id>TDOM1</access-point-id>
    <local-access-point>TDOM2</local-access-point>
    <nw-addr>123.123.123.123:1234</nw-addr>
    <name>myRTuxDom</name>
  </wtc-remote-tux-dom>
  <wtc-remote-tux-dom>
    <access-point>TDOM3</access-point>
    <access-point-id>TDOM3</access-point-id>
    <nw-addr>234.234.234.234:5555</nw-addr>
    <name>2ndRemoteTuxDom</name>
  </wtc-remote-tux-dom>
  <wtc-export>
    <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
    <local-access-point>TDOM2</local-access-point>
    <name>myExportedResources</name>
    <resource-name>TOLOWER</resource-name>
  </wtc-export>
  <wtc-import>
    <local-access-point>TDOM2</local-access-point>
    <name>myImportedResources</name>
    <remote-access-point-list>TDOM1</remote-access-point-list>
    <remote-name>TOUPPER</remote-name>
  </wtc-import>
  <wtc-import>
    <local-access-point>TDOM2</local-access-point>
    <name>2ndImportedResources</name>
    <remote-access-point-list>TDOM3</remote-access-point-list>
    <remote-name>TOUPPER</remote-name>
  </wtc-import>
</wtc-server>
```
**How to Configure Failback**

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

- Automatic retries (On Startup only)
- Incoming connections

**How to Configure Link-level Failover**

To support link-level failover, you must specify the correct failover sequence information in the comma separated syntax `<nw-addr>` XML tag in the `WTCTRemoteTuxDomMBean` and `WTCLocalTuxDomMBean` definitions. The order of the network addresses determines the order of preference for failover.

**Note:** The value of the XML tag is checked for correct syntax. If the syntax is not correct, the `InvalidAttributeException` is thrown.

The semantic of the link-level failover is late binding, which means the existence and availability is not checked when the MBean is created. This is to allow users to add the machine to DNS after the WTC configuration is created, but before the TDomain session connection is created.

The correct syntax in `config.xml` will be as follow using comma separated syntax for the `<nw-addr>` XML tag.

```
<nw-addr>//host1:4001</nw-addr>  --> only one host, no link-level failover
<nw-addr>//host1:4001,//host2:4001</nw-addr>  --> can failover to host2
<nw-addr>//host1:4001,//host2:4001,//host3:4001</nw-addr>  --> can failover from host 1 to host2, and if host2 still not available then failover to host3
```

**Sample Link-level Failover Configuration**

The following example configures a WTC local access point named `WDOM`, and one TDomain session with name `TDOM`. This TDomain session also defines a remote access point named `DOM1`. The TDomain session in this case is a session between `WDOM` and `TDOM`. The local access point will try to listen on end point `"//pluto:4100"` first; if fails to create a listening endpoint, the session attempts to create a listening endpoint on `"//saturn:4101"`. If WTC migrated from `pluto` to `saturn`, then the remote access point `DOM1` is able to contact `WDOM` using `"//saturn:4101"`. 
If the remote access point DOM1 migrates from host mercury to host mars, the WDOM can contact DOM1 at "/mars:4001".

The order of network address specified in the list provides order preference. For WDOM, "/pluto:4100" is the first choice for creating a listening endpoint and "/saturn:4101" is the second choice. For remote access point DOM1, "/mercury:4001" is the first choice to create a connection from WDOM to DOM1 and "/mars:4001" is the second choice.

Listing 3-3  Link-level Failover Configuration

```xml
<wtc-server>
  <name>myWTCserver</name>
  ....
  <wtc-local-tux-dom>
    <name>WDOM</name>
    <access-point>WDOM</access-point>
    <access-point-id>WDOM</access-point-id>
    <nw-addr>//pluto:4100,//saturn:4101</nw-addr>
  </wtc-local-tux-dom>
  <wtc-remote-tux-dom>
    <name>TDOM</name>
    <access-point>DOM1</access-point>
    <access-point-id>DOM1</access-point-id>
    <local-access-point>WDOM</local-access-point>
    <nw-addr>//mercury:4001,//mars:4001</nw-addr>
  </wtc-remote-tux-dom>
  ....
</wtc-server>
```

Configuring for TypedMBString Support

To configure WTC to support MBSTRING buffers, you must specify the encoding you want to use in the RemoteMBEncoding attribute of the WTCResources definition. This attribute is optional and if it is not specified or is invalid, Java’s default encoding is used.

TypedMBString uses the conversion function java.lang.String class for converting between Unicode and an external encoding. TypedMBString uses a map file to map the encoding names between Java and GNU iconv, which is used by the C language API of MBSTRING. The map file is mbencmap, which is a text-based file in $WL_HOME/server/lib directory as a default. The map file creates a HashMap with each "user_name java_name" pair. You can customize the map file.
An encoding map file contains one or more lines with the following syntax.

```
<user_name> <java_name1>,<java_name2>,[java_name3,...]
```

By specifying multiple `java_names` in a line, multiple Java encoding names are mapped to a single `user_name`. The `user_name` always maps to the first `java_name` in the line.

### Authentication of Remote Access Points

**Note:** Tuxedo 6.5 users should set the `Interoperate` parameter to `Yes`.

Domain gateways can be made to authenticate incoming connections requested by remote access points and outgoing connections requested by local access points. Application administrators can define when security should be enforced for incoming connections from remote access points. You can specify the level of security used by a particular local access point by setting the `Security` attribute in the Security tab of the local Tuxedo access point configuration of your WTC Service. There are three levels of password security:

- **NONE**—incoming connections from remote access points are not authenticated.

- **Application Password**—incoming connections from remote access points are authenticated using the application password defined in the Resource configuration of your WTC Service. You use the `weblogic.wtc.gwt.genpasswd` utility to create encrypted application passwords.

- **Domain Password**—this feature enforces security between two or more access points. Connections between the local and remote access points are authenticated using password pairs defined in the Password configuration of your WTC Service. You use the `weblogic.wtc.gwt.genpasswd` utility to create encrypted local and remote passwords.

The `Security` attribute in the Security tab of the local Tuxedo access point of your WTC Service must match the `SECURITY` attribute 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 access point and the remote access point.

- If the security type of the local Tuxedo access point in your WTC Service does not match the security type of the `*DM_LOCAL_DOMAINS` or if the passwords do not match, the connection fails.
Configuring a Password Configuration

Note: For more information on how to assign a PasswordKey, see “How to Set WebLogic Tuxedo Connector Properties” on page 2-7.

The /Domain architecture with SECURITY=DM_PW requires a password for each connection principal. Each TDomain session between two TDomain gateways has two distinctive connection principals associated with it; by default, they are represented by Domain IDs. The default Session Authentication with DM_PW requires both sides configure two secrets for both connection principals so they can authenticate each other. The following example provides configurations for both WTC and Tuxedo.

- WTC is configured with:
  - local access point WDOM1 with DOMAIN ID WDOM1
  - remote access point TDOM1 with DOMAIN ID TDOM1
  - security set to DM_PW

- Tuxedo is configured with:
  - its own local access point TDOM1 with DOMAIN ID TDOM1
  - remote access point WDOM1 with DOMAIN ID WDOM1
  - security is set to DM_PW

Then WTC needs to configure a password pair for TDOMAIN session (WDOM1, TDOM1). For example, the password pair is represent as (pWDOM1, pTDOM1) for the TDomain Session (WDOM1, TDOM1). Then Tuxedo TDOMAIN needs to configure a password pair for TDOMAIN session (TDOM1, WDOM1). The password pair should be (pTDOM1, pWDOM1) in this case.

Generating Encrypted Passwords

Use weblogic.wtc.gwt.genpasswd to generate encrypted passwords for Local Password, Remote Password, and App Password attributes. The utility uses a key to encrypt a password that is copied into the Password or Resources configuration of your WTC Service.

- The Password configuration of your WTC Service does not store clear text passwords.
- The key value is a WebLogic Server property.
  - PasswordKey is the attribute used to assign the key.
    -Dweblogic.wtc.PasswordKey=mykey
where: \textit{mykey} is the key value

- The \texttt{PasswordKey} attribute can only be assigned one key value. This key value is used for all WebLogic Tuxedo Connector passwords generated (local, remote, and application passwords) for use with a specific WebLogic Server.

**Usage**

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

Example:

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

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

Call the utility with a key value, password to encrypt, and the type of password.

Example:

```bash
$ java weblogic.wtc.gwt.genpasswd Key1 LocalPassword1 local
```

The utility will respond with the encoded password and password IV. Cut and paste the results into the appropriate fields in Password configuration of your WTC Service.

Local Password : my\_password

Local Password IV: my\_passwordIV

where

- Cut and paste the string of characters represented by \texttt{my\_password} into the Password field.
- Cut and paste the string of characters represented by \texttt{my\_passwordIV} into the PasswordIV field.

**Examples**

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

**Local Passwords**

The following example uses \texttt{key1} to encrypt “LocalPassword1” as the password of the local access point.
### Remote Passwords

The following example uses `mykey` to encrypt “RemotePassword1” as the password for the remote access point.

```
$ java weblogic.wtc.gwt.genpasswd mykey RemotePassword1 remote
Remote Password : A/DgdJYOJunFUFJa62YMpGshAn8pC02zPT0T7EigaVg=
Remote Password IV : ohYHzhYHPO=
```

Your Password attributes are:

- Remote Password: `A/DgdJYOJunFUFJa62YMpGshAn8pC02zPT0T7EigaVg=`
- Remote Password IV: `ohYHzhYHPO=`

### App Passwords

The following example uses `mykey` to encrypt “test123” as the application password.

```
$ java weblogic.wtc.gwt.genpasswd mykey test123 application
App Password : uou2MALQE2gNqt8abNKiC9ADN5gHDlvqO+Xt/VjakE=
App Password IV : eQuKj0aPfCw=
```

Your Resources attributes are:

- Application Password: `uou2MALQE2gNqt8abNKiC9ADN5gHDlvqO+Xt/VjakE=`
- Application Password IV: `eQuKj0aPfCw=`

### User Authentication

Access Control Lists (ACLs) limit the access to local services within a local access point by restricting the remote Tuxedo access point that can execute these services. Inbound policy from a remote Tuxedo access point is specified using the `AclPolicy` attribute. Outbound policy towards a remote Tuxedo domain is specified using the `CredentialPolicy` attribute. 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 AclPolicy and CredentialPolicy are:

- LOCAL
- GLOBAL

**ACL Policy is LOCAL**

If the WebLogic Tuxedo Connector ACL Policy is set to LOCAL, access to local services does not depend on the remote user credentials. The Tuxedo remote access point ID is authenticated as a local WebLogic Server user. To allow WebLogic Tuxedo Connector to authenticate a DOMAIND in as a local user, use the WebLogic Server Console to complete the following steps:

1. Click on the Security node.
2. Click on Realms.
3. Select your default security Realm.
4. Click on Users.
5. Click the Configure a new User text link.
6. In the General tab, do the following:
   a. Add the Tuxedo DOMAIND in the Name field.
   b. Enter and validate a password.
   c. Click apply.

**ACL Policy is GLOBAL**

If the WebLogic Tuxedo Connector ACL Policy is GLOBAL, access to local services depends on the remote user credentials.

**Remote Access Point Credential Policy is GLOBAL**

If a remote domain is running with the CredentialPolicy set to GLOBAL, the request has the credentials of the remote user, thus the ability to access the local service depends on this credential.
Remote Access Point Credential Policy is LOCAL

If a remote domain is running with the CredentialPolicy set to LOCAL, the result depends on the access rights given to the user with remote access point ID as its name.

User Authentication for Tuxedo 6.5

Tuxedo 6.5 users should set the Interoperate parameter to Yes. The AclPolicy and CredentialPolicy elements are ignored and the Tuxedo remote access point ID is authenticated as a local WebLogic Server user. If you require User Security features and use the WebLogic Tuxedo Connector, you will need to upgrade to Tuxedo 7.1 or higher.

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

The following sections provide information on how to configure WebLogic Tuxedo provide user security information to Tuxedo:

- TpUsrFile Plug-in
- LDAP Plug-in
- Custom Plug-in
- Anonymous Users

TpUsrFile Plug-in

The TpUsrFile plug-in provides traditional Tuxedo TpUserFile functionality for users who do not need single point security administration or custom security authentication. Use the following steps to configure WebLogic Tuxedo Connector to provide security between Tuxedo and WebLogic Server applications using the TpUsrFile plug-in AppKey Generator:

- Configuring the Local Tuxedo Access Point for the TpUsrFile Plug-in
- Configure the Remote Tuxedo Access Point for the TpUsrFile Plug-in

Configuring the Local Tuxedo Access Point for the TpUsrFile Plug-in

Set the security attribute in the Security tab of the local Tuxedo access point of your WTC Service to match the SECURITY parameter of the *DM_LOCAL_DOMAINS section of the Tuxedo domain configuration file.
Configure the Remote Tuxedo Access Point for the TpUsrFile Plug-in

Configure the Security tab of the remote Tuxedo access point of your WTC Service to establish an inbound and outbound Access Control List (ACL) policy.

Perform the following steps to prepare the WebLogic Server environment:

1. Set the AclPolicy attribute to **GLOBAL**.
2. Set the CredentialPolicy attribute to **GLOBAL**.
3. Set the Allow Anonymous attribute for your environment. If you select to allow anonymous users to access Tuxedo, you must set the value of the Default AppKey to be used by anonymous users. For more information on anonymous users, see “Anonymous Users” on page 3-21.
4. Select TpUsrFile from the AppKey Generator dropdown box.
5. Set the value of the Tp Usr File attribute to the full path to the user password file.

   You must have a copy of the Tuxedo tpusr file in your WebLogic Server environment. Copy the tpusr file from TUXEDO to the WebLogic Server application environment or generate your own tpusr file. 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/tux90/sec/secadm.htm#1239966.

Using the Resources TpUsrFile attribute

The location of the TpUsrFile can be specified from your remote Tuxedo access point configurations or from your Resources configuration. You may find it convenient assign the value of the TpUsrFile attribute globally at the WTC Service level, rather than by assigning it individually on all of your remote Tuxedo access point configurations. Use the following guidelines to help you determine where to best configure the TpUsrFile attribute:

- All TpUsrFile attribute values are ignored if the TpUsrFile Plug-in is not selected as the AppKey Generator, regardless of location.
- If the Resources configuration does not have TpUsrFile attribute values, the TpUsrFile attribute value must be specified in the remote Tuxedo access point configurations. The cached user record information is ignored.
- If the Resources and remote Tuxedo access point configurations contain TpUsrFile attribute values, the attribute values in the remote Tuxedo access points are used. The cached user record information is ignored.
If the remote Tuxedo access point configurations do not have TpUsrFile attribute values, the TpUsrFile attribute value must be specified in the Resources configuration. The cached user record is used, which improves system performance. However, this restricts the user to have the same identity in all remote Tuxedo access points.

**LDAP Plug-in**

The LDAP plug-in provides single point security administration that allows you to maintain user security information in a WebLogic Server embedded LDAP server and use the WebLogic Server Console to administer the security information from a single system. Requires Tuxedo 8.1 and higher. Use the following steps to configure WebLogic Tuxedo Connector to provide security between Tuxedo and WebLogic Server applications using the LDAP Plug-in AppKey Generator:

- Implementing Single Point Security Administration
- Configure the Local Tuxedo Access Point for the LDAP Plug-in
- Configure the Remote Tuxedo Access Point for the LDAP Plug-in

**Implementing Single Point Security Administration**


**Configure the Local Tuxedo Access Point for the LDAP Plug-in**

Set the `security` attribute in the Security tab of the local Tuxedo access point of your WTC Service to match the SECURITY parameter of the *DM_LOCAL_DOMAINS* section of the Tuxedo domain configuration file.

**Configure the Remote Tuxedo Access Point for the LDAP Plug-in**

Configure the Security tab of the remote Tuxedo access point of your WTC Service to establish an inbound and outbound Access Control List (ACL) policy.

Perform the following steps to prepare the WebLogic Server environment:

1. Set the AclPolicy attribute to `GLOBAL`.
2. Set the CredentialPolicy attribute to `GLOBAL`.
3. Set the Allow Anonymous attribute for your environment. If you select to allow anonymous users to access Tuxedo, you must set the value of the Default AppKey to be used by anonymous users. For more information on anonymous users, see “Anonymous Users” on page 3-21.

4. Select LDAP from the AppKey Generator dropdown box.

5. If necessary, set the value of the Tuxedo UID Keyword attribute and Tuxedo GID attribute. Default values are provided. These keywords for the Tuxedo user ID (UID) is used to extract the Tuxedo UID and GID in the user record of the embedded LDAP database.

Custom Plug-in

Note: For information on how to create a Custom Plug-in, see How to Create a Custom AppKey Plug-in athttp://e-docs.bea.com/wls/docs90/wtc_atmi/CustomAppKey.html.

The Custom plug-in provides the ability for you to create customized security authentication. Use the following steps to configure WebLogic Tuxedo Connector to provide security between Tuxedo and WebLogic Server applications using the Custom Plug-in AppKey Generator:

- Configure the Local Tuxedo Access Point for the Custom Plug-in
- Configure the Remote Tuxedo Access Point for the Custom Plug-in

Configure the Local Tuxedo Access Point for the Custom Plug-in

Set the security attribute in the Security tab of the local Tuxedo access point of your WTC Service to match the SECURITY parameter of the *DM_LOCAL_DOMAINS section of the Tuxedo domain configuration file.

Configure the Remote Tuxedo Access Point for the Custom Plug-in

Configure the Security tab of the remote Tuxedo access point of your WTC Service to establish an inbound and outbound Access Control List (ACL) policy.

Perform the following steps to prepare the WebLogic Server environment:

1. Set the AclPolicy attribute to GLOBAL.
2. Set the CredentialPolicy attribute to GLOBAL.
3. Set the Allow Anonymous attribute for your environment. If you select to allow anonymous users to access Tuxedo, you must set the value of the Default AppKey to be used by anonymous users. For more information on anonymous users, see “Anonymous Users” on page 3-21.

4. Select Custom from the AppKey Generator dropdown box.

5. Set the value of the Custom AppKey Class attribute to the full pathname to your Custom AppKey generator class. This class is loaded when the WTC Service is started.

6. Set the value of the Custom AppKey Param attribute to the optional parameters that you may require to use your Custom AppKey class when it is initialized when the WTC Service starts.

Anonymous Users

The Allow Anonymous attribute on the Security tab of a remote Tuxedo access point specifies whether the anonymous user is allowed to access Tuxedo. If the anonymous user is allowed to access Tuxedo, the value of the Default AppKey attribute is used for TpUsrFile and LDAP AppKey plug-ins. The TpUsrFile and LDAP plug-ins do not allow users that are not defined in user database to access Tuxedo unless the Allow Anonymous attribute is enabled. Interaction with the Custom AppKey plug-in depends on the design of the Custom AppKey generator.

The default value of the Default AppKey is -1. If you wish to use this value, you must make sure that your Tuxedo environment has a user assigned to that key value. You should avoid assigning the Default AppKey value to 0. In some systems, this specifies the user as root.

Anonymous Users and CORBA Services

It is important to understand the differences between how ATMI services and CORBA services authenticate an anonymous user. ATMI services rely on the Default AppKey value sent with the message. Corba services use the default WebLogic Server anonymous user name <anonymous> to identify the user credential defined in the Tuxedo tpusr file. CORBA users must configure the anonymous user using one of the following methods to become an authenticated user:

- Add <anonymous> to the Tuxedo tpusr file.
- Define anonymous as a user in the WebLogic Authentication provider. You do this by setting the following argument when starting a WebLogic Server instance:
  -Dweblogic.security.anonymousUserName=anonymous
**Link-Level Encryption**

You can use encryption 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` attributes of the Security tab in the local Tuxedo access points and remote Tuxedo access points configurations of your WTC Service.

**Note:** Encryption requires appropriate licensing. For more information on license requirements, see “Licensing” on page 1-3.
Administration of CORBA Applications

Note: For more information on CORBA applications, see Tuxedo CORBA at http://e-docs.bea.com/tuxedo/tux90/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 WTC Service, see “Configuring WebLogic Tuxedo Connector for Your Applications” on page 2-2.

This section provides information on how to configure a WTC Service to support a call to a Tuxedo CORBA server from a WebLogic Server EJB. Use the following steps to configure your WTC Service:

1. Configure Local Tuxedo Access Points WebLogic Server applications.
2. Configure Remote Tuxedo Access Points for your Tuxedo CORBA domain.
3. Configure Imported Services.
- Set Resource Name to “//domain_id” where domain_id is DOMAINID specified in the Tuxedo[UBB_CONFIG] 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 Local Access Point to the value of the Local Access Point attribute of your Remote Tuxedo Access Point.

- Set the Remote Access Point List to the value of the Access Point Id attribute of the Remote Tuxedo Access Point.

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/docs90/wtc_atmi/index.html.

**Example WTC Service and Tuxedo UBB Files**

The following WTC Service (represented by the WTCServer MBean in the config.xml file) provides an example of how to configure an Imported Services configuration for a TUXEDO CORBA server.

**Listing 4-1 Example WTCServer MBean for a CORBA Server Application**

```
<wtc-server>
  <name>WTCsimpappCNS</name>
  <wtc-local-tux-dom>
    <access-point>examples</access-point>
    <access-point-id>examples</access-point-id>
    <connection-policy>ON_DEMAND</connection-policy>
    <nw-addr>//123.123.123.123:5678</nw-addr>
    <name>myLoclTuxDom</name>
    <security>NONE</security>
  </wtc-local-tux-dom>
  <wtc-remote-tux-dom>
    <access-point>TUXDOM</access-point>
    <access-point-id>TUXDOM</access-point-id>
    <local-access-point>examples</local-access-point>
    <nw-addr>//123.123.123.123:1234</nw-addr>
    <name>myRTuxDom</name>
  </wtc-remote-tux-dom>
  <wtc-import>
    <local-access-point>examples</local-access-point>
    <name>myImportedResources</name>
    <remote-access-point-list>TUXDOM</remote-access-point-list>
  </wtc-import>
</wtc-server>
```
The following example Tuxedo UBB configuration file has a `DOMAINID` name of `simpapp`. The `DOMAINID` name is used in the `Resource Name` attribute of the Imported Services configuration of your WTC Service.

**Listing 4-2 Example Tuxedo UBB File for a CORBA Server Application**

```plaintext
*RESOURCES
  IPCKEY  55432
  DOMAINID simpapp
  MASTER  SITE1
  MODEL   SHM
  LDBAL   N
*MACHINES
  *YODA*
    LMID=SITE1
    APPDIR="your APPDIR"
    TUXCONFIG="APPDIR\tuxconfig"
    TUXDIR="your TUXDIR"
    MAXWSCLIENTS=10
*GROUPS
  SYS_GRP
    LMID=SITE1
    GRPNO=1
  APP_GRP
    LMID=SITE1
    GRPNO=2
*SERVERS
  DEFAULT:
    RESTART=Y
    MAXGEN=5
    TMSYSEVT
    SRVGRP=SYS_GRP
```
SRVID=1
TMFFNAME
SRVGRP=SYS_GRP
SRVID=2
CLOPT="-A -- -N -M"
TMFFNAME
SRVGRP=SYS_GRP
SRVID=3
CLOPT="-A -- -N"
TMFFNAME
SRVGRP=SYS_GRP
SRVID=4
CLOPT="-A -- -F"
ISL
SRVGRP=SYS_GRP
SRVID=5
CLOPT="-A -- -n </your tux machine:2468>"
CNS
SRVGRP=SYS_GRP
SRVID=6
CLOPT="-A -- *
DMADM SRVGRP=SYS_GRP SRVID=7
GWADM SRVGRP=SYS_GRP SRVID=8
GWTDOMAIN SRVGRP=SYS_GRP SRVID=9
simple_server
SRVGRP=APP_GRP
SRVID=1
RESTART = N

*SERVICES

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 WTC Service to enable Tuxedo CORBA objects to invoke upon EJBs deployed in WebLogic Server using the RMI-IIOP API.
How to Administer and Configure WebLogic Tuxedo Connector for Inbound RMI-IIOP

- Configuring Your WTC Service for Inbound RMI-IIOP
- Administering the Tuxedo Application Environment

Configuring Your WTC Service for Inbound RMI-IIOP

**Note:** For more information on how to configure your WTC Service, see “Configuring WebLogic Tuxedo Connector for Your Applications” on page 2-2.

Configure Local Tuxedo Access Points and Remote Tuxedo Access Points as needed for your environment. No special administration steps are required to enable Tuxedo CORBA objects to invoke upon EJBs deployed in WebLogic Server using the RMI-IIOP API.

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/tux90/intern/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 your_bind_name
   ```

   where

   ```
   your_bind_name
   ```

   is the CosNaming service object name from your Tuxedo application.

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

**Listing 4-3**  ior.txt File for iiopejb.stateless.server.tux Tuxedo Client Example

```
corbaloc:tgiop:myServer/NameService
```

where
myServer is your server name.

3. Modify the *DM_REMOTE_SERVICES of your Tuxedo domain configuration file. Replace your WebLogic Server service name, formerly the DOMAINID, with the name of your WebLogic Server.

Listing 4-4  Domain Configuration File

```plaintext
*DM_RESOURCES
  VERSION=U22

*DM_LOCAL_DOMAINS
  TDOM1 GWGRP=SYS_GRP
  TYPE=TDOMAIN
  DOMAINID="TDOM1"
  BLOCKTIME=20
  MAXDATALEN=56
  MAXRDOM=89

*DM_REMOTE_DOMAINS
  TDOM2 TYPE=TDOMAIN
  DOMAINID="TDOM2"

*DM_TDOMAIN
  TDOM1 NWADDR="/123.123.123.123:1234"
  TDOM2 NWADDR="/234.234.234.234:5678"

*DM_REMOTE_SERVICES
  "/myServer"
```

where

myServer is the server name that is running the WTC Service.

4. Load your modified domain configuration file using dmloadcf.
Guidelines About Using Your Server Name as an Object Reference

This section provides guidelines you need to remember when creating server names that are used as object references.

- The maximum field length Tuxedo accepts in the DM_REMOTE_SERVICES section is 15 characters including the //. For example: If your server name is examplesServer, your DM_REMOTE_SERVICES object reference is //examplesServe.

- If you require multiple servers, the server names must be unique in the first 13 characters.

- You can use the complete name of your server name in the ior.txt file if it exceeds 13 characters. For example: corbaloc:tgio:examplesServer/NameService

How to Configure WebLogic Tuxedo Connector for Outbound RMI-IIOP

Note: For more information on how to configure your WTC Service, see “Configuring WebLogic Tuxedo Connector for Your Applications” on page 2-2.

This section provides information on how to enable WebLogic Server EJBs to invoke upon Tuxedo CORBA objects using the RMI-IIOP API. Use the following steps to modify your WTC Service:

1. Configure a Local Tuxedo Access Point.

2. Configure Remote Tuxedo Access Point. Outbound RMI-IIOP requires two additional elements: Federation URL and Federation Name.

   - Set Federation URL 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 Federation Name to the symbolic name of the federation point.

3. Configure Imported Services.

   - Set Resource Name to “/domain_id” where domain_id is DOMAINID specified in the TuxedoUBBCONFIG 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 Local Access Point to the value of the Local Access Point attribute of your Remote Tuxedo Access Point.
Set Remote Access Point List to the value of the Access Point Id attribute of your Remote Tuxedo Access Point.

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/docs90/wtc_atmi/index.html.

**Example Outbound RMI-IIOP Configuration**

The following WTCServer MBean in the config.xml file provides an example of a configured WTC Service for outbound RMI-IIOP.

**Listing 4-5  Example WTCServer MBean for Outbound RMI-IIOP**

```
<wtc-server>
  <name>WTCtrader</name>
  <wtc-local-tux-dom>
    <access-point>TDOM2</access-point>
    <access-point-id>TDOM2</access-point-id>
    <connection-policy>ON_DEMAND</connection-policy>
    <nw-addr>//123.123.123.123:5678</nw-addr>
    <name>myLocTuxDom</name>
    <security>NONE</security>
  </wtc-local-tux-dom>
  <wtc-remote-tux-dom>
    <access-point>TDOM1</access-point>
    <access-point-id>TDOM1</access-point-id>
    <federation-name>tuxedo.corba.remote</federation-name>
    <federation-url>corbaloc:tgiop:simpapp/NameService</federation-url>
    <local-access-point>TDOM2</local-access-point>
    <nw-addr>//123.123.123.123:1234</nw-addr>
    <name>myRTuxDom</name>
  </wtc-remote-tux-dom>
  <wtc-import>
    <local-access-point>TDOM2</local-access-point>
    <name>myImportedResources</name>
    <remote-access-point-list>TDOM1</remote-access-point-list>
    <remote-name>//simpapp</remote-name>
  </wtc-import>
</wtc-server>
```
How to Manage WebLogic Tuxedo Connector in a Clustered Environment

Note: For more information on WebLogic Server Clusters, see Using WebLogic Server Clusters at http://e-docs.bea.com/wls/docs90/cluster/index.html.

The following sections provide information on how to administer and configure the WebLogic Tuxedo Connector for use in a clustered environment:

- WebLogic Tuxedo Connector Guidelines for Clustered Environments
- How to Configure OutBound Requests to Tuxedo Domains
- How to Configure Inbound Requests from Tuxedo Domains

WebLogic Tuxedo Connector Guidelines for Clustered Environments

Use the following guidelines when deploying WebLogic Tuxedo Connector in a clustered environment:

- Because the binding is not replicated in other servers in a cluster, all the WebLogic Servers in the cluster must have a configured WebLogic Tuxedo Connector that includes an Imported Services tab that defines any imported services required. If one server in the cluster does not have a WebLogic Tuxedo Connector deployed, the Enterprise Java Bean (EJB) or Message Driven Bean (MDB) won't be able to find a Tuxedo Connection Factory for that connection.
• The administrator is responsible for the correct configuration of the TUXEDO DMCONFIG to allow proper load balancing and fail over of inbound calls to clustered nodes.

• WebLogic Tuxedo Connector does not support inbound TGIOP in clustered environments.

How to Configure for Clustered Nodes

Configuring WTC servers for a clustered WebLogic Server (WLS) environment is the same as configuring WTC for a non-clustered WLS environment. Configure a WTC server for each node in a cluster that you intend to deploy a JATMI-based EJB. Then target each WTC server to their intended WebLogic Server. There should only be one WTC server per WebLogic Server node.

Limitations for Clustered Nodes

For every WebLogic Server that has a JATMI-based EJB deployed, you must configure it with a WTC server. The high availability depends on the WebLogic Server cluster’s own HA ability. There is no special capability to failover/failback among the WTC servers.

How to Configure OutBound Requests to Tuxedo Domains

Note: For more information on WebLogic Server Clusters, see Cluster Features and Infrastructure at http://e-docs.bea.com/wls/docs90/cluster/features.html. WebLogic Tuxedo Connector also provides domain-level failover and failback capabilities. For more information, see “Configuring Failover and Failback” on page 3-8.

The load balancing and failover of the outbound requests from WebLogic Server depend on the WebLogic Server EJB and MDB.

Example Clustered WebLogic Tuxedo Connector Configuration

The following configuration provides an example of WebLogic Tuxedo Connector in a clustered environment. The cluster consists of an administration server (wtcAServer) and three managed servers (wtcMServer1, wtcMServer2, wtcMServer3). Each managed server has a configured WTC Service that contains the same service (TOUPPER) in as an imported service.

Listing 5-1   Example Clustered WebLogic Tuxedo Connector Configuration

```
<name>mydomain</name>
<security-configuration>
```
<name>mydomain</name>
<realm>
<sec:authentication-provider xsi:type="wls:default-authenticatorType"></sec:authentication-provider>
<sec:authentication-provider xsi:type="wls:default-identity-asserterType">
<sec:active-type>AuthenticatedUser</sec:active-type>
</sec:authentication-provider>
<sec:role-mapper xsi:type="wls:default-role-mapperType"></sec:role-mapper>
<sec:authorizer xsi:type="wls:default-authorizerType"></sec:authorizer>
<sec:adjudicator xsi:type="wls:default-adjudicatorType"></sec:adjudicator>
<sec:credential-mapper xsi:type="wls:default-credential-mapperType"></sec:credential-mapper>
<sec:cert-path-provider xsi:type="wls:web-logic-cert-path-providerType">
<sec:cert-path-builder>WebLogicCertPathProvider</sec:cert-path-builder>
</sec:cert-path-provider>
<security-configuration xsi:type="wls:security-dd-modelType">
<sec:name>myrealm</sec:name>
</security-configuration>
</realm>
<default-realm>myrealm</default-realm>
<credential-encrypted>{3DES}O0Qw7QBG3+cmemXbtKhJRJL2QLw7tqSYkoWqBtU7W+IoPebpoNai/T3SdxtCBwVH0JJpi/sA8JMMJ4M4i3KqVgd26A31lz</credential-encrypted>
<web-app-files-case-insensitive>os</web-app-files-case-insensitive>
<node-manager-username>weblogic</node-manager-username>
<node-manager-password-encrypted>{3DES}37KmzVTzxxS9VFxCFSVGWhA==</node-manager-password-encrypted>
<enforce-strict-url-pattern>false</enforce-strict-url-pattern>
<security-configuration>
<realm>WL_default_realm</realm>
<password-policy>WL_default_password_policy</password-policy>
</security-configuration>
</secure>
<wtc-server>
<name>WTCServer1</name>
<target>wtcMServer1</target>
<wtc-local-tux-dom>
<name>ltd0</name>
<access-point>WDOM1</access-point>
<access-point-id>WDOM1</access-point-id>
<security>NONE</security>
<connection-policy>ON_STARTUP</connection-policy>
<brick-time>30000</brick-time>
<nw-addr>//mymachine:20401</nw-addr>
</wtc-local-tux-dom>
</wtc-local-tux-dom>
<wtc-remote-tux-dom>
  <name>rtd0</name>
  <access-point>TDOM1</access-point>
  <access-point-id>TDOM1</access-point-id>
  <local-access-point>WDOM1</local-access-point>
  <nw-addr>//123.123.123.123:20301</nw-addr>
</wtc-remote-tux-dom>
<wtc-remote-tux-dom>
  <name>rtd1</name>
  <access-point>TDOM2</access-point>
  <access-point-id>TDOM2</access-point-id>
  <local-access-point>WDOM1</local-access-point>
  <nw-addr>//123.123.123.123:20302</nw-addr>
</wtc-remote-tux-dom>
<wtc-export>
  <name>exp0</name>
  <resource-name>TOLOWER</resource-name>
  <local-access-point>WDOM1</local-access-point>
  <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
  <remote-name>TOLOWER</remote-name>
</wtc-export>
<wtc-export>
  <name>exp1</name>
  <resource-name>EJBSleep</resource-name>
  <local-access-point>WDOM1</local-access-point>
  <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
  <remote-name>EJBSleep</remote-name>
</wtc-export>
<wtc-import>
  <name>imp0</name>
  <resource-name>TOUPPER</resource-name>
  <local-access-point>WDOM1</local-access-point>
  <remote-access-point-list>TDOM2,TDOM1</remote-access-point-list>
</wtc-import>
<wtc-import>
  <name>imp1</name>
  <resource-name>LSleep</resource-name>
  <local-access-point>WDOM1</local-access-point>
  <remote-access-point-list>TDOM2,TDOM1</remote-access-point-list>
</wtc-import>
</wtc-server>
<wtc-server>
  <name>WTCServer2</name>
  <target>wtcMServer2</target>
  <wtc-local-tux-dom>
    <name>ltd0</name>
    <access-point>WDOM2</access-point>
    <access-point-id>WDOM2</access-point-id>
  </wtc-local-tux-dom>
<security>NONE</security>
<connection-policy>ON_STARTUP</connection-policy>
<block-time>30000</block-time>
<nw-addr>//mymachine:20402</nw-addr>
</wtc-local-tux-dom>
<wtc-remote-tux-dom>
  <name>rtd0</name>
  <access-point>TDOM1</access-point>
  <access-point-id>TDOM1</access-point-id>
  <local-access-point>WDOM2</local-access-point>
  <nw-addr>//123.123.123.123:20301</nw-addr>
</wtc-remote-tux-dom>
<wtc-remote-tux-dom>
  <name>rtd1</name>
  <access-point>TDOM2</access-point>
  <access-point-id>TDOM2</access-point-id>
  <local-access-point>WDOM2</local-access-point>
  <nw-addr>//123.123.123.123:20302</nw-addr>
</wtc-remote-tux-dom>
<wtc-export>
  <name>exp0</name>
  <resource-name>TOLOWER</resource-name>
  <local-access-point>WDOM2</local-access-point>
  <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
  <remote-name>TOLOWER</remote-name>
</wtc-export>
<wtc-export>
  <name>exp1</name>
  <resource-name>EJBLSleep</resource-name>
  <local-access-point>WDOM2</local-access-point>
  <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
  <remote-name>EJBLSleep</remote-name>
</wtc-export>
<wtc-import>
  <name>imp0</name>
  <resource-name>TOUPPER</resource-name>
  <local-access-point>WDOM2</local-access-point>
  <remote-access-point-list>TDOM2,TDOM1</remote-access-point-list>
</wtc-import>
<wtc-import>
  <name>imp1</name>
  <resource-name>LSleep</resource-name>
  <local-access-point>WDOM2</local-access-point>
  <remote-access-point-list>TDOM2,TDOM1</remote-access-point-list>
</wtc-import>
</wtc-server>
<wtc-server>
  <name>WTCServer3</name>
  <target>wtcMServer3</target>
<wtc-local-tux-dom>
  <name>ltd0</name>
  <access-point>WDOM3</access-point>
  <access-point-id>WDOM3</access-point-id>
  <security>NONE</security>
  <connection-policy>ON_STARTUP</connection-policy>
  <block-time>30000</block-time>
  <nw-addr>//mymachine:20403</nw-addr>
</wtc-local-tux-dom>

<wtc-remote-tux-dom>
  <name>rtd0</name>
  <access-point>TDOM1</access-point>
  <access-point-id>TDOM1</access-point-id>
  <local-access-point>WDOM3</local-access-point>
  <nw-addr>://123.123.123.123:20301</nw-addr>
</wtc-remote-tux-dom>

<wtc-remote-tux-dom>
  <name>rtd1</name>
  <access-point>TDOM2</access-point>
  <access-point-id>TDOM2</access-point-id>
  <local-access-point>WDOM3</local-access-point>
  <nw-addr>://123.123.123.123:20302</nw-addr>
</wtc-remote-tux-dom>

<wtc-export>
  <name>exp0</name>
  <resource-name>TOLOWER</resource-name>
  <local-access-point>WDOM3</local-access-point>
  <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
  <remote-name>TOLOWER</remote-name>
</wtc-export>

<wtc-export>
  <name>exp1</name>
  <resource-name>EJBLSleep</resource-name>
  <local-access-point>WDOM3</local-access-point>
  <ejb-name>tuxedo.services.TOLOWERHome</ejb-name>
  <remote-name>EJBLSleep</remote-name>
</wtc-export>

<wtc-import>
  <name>imp0</name>
  <resource-name>TOUPPER</resource-name>
  <local-access-point>WDOM3</local-access-point>
  <remote-access-point-list>TDOM2,TDOM1</remote-access-point-list>
</wtc-import>

<wtc-import>
  <name>imp1</name>
  <resource-name>LSleep</resource-name>
  <local-access-point>WDOM3</local-access-point>
  <remote-access-point-list>TDOM2,TDOM1</remote-access-point-list>
</wtc-import>
How to Configure OutBound Requests to Tuxedo Domains

```xml
<server>
  <name>wtcAServer</name>
  <native-io-enabled>true</native-io-enabled>
  <ssl>
    <name>wtcAServer</name>
    <identity-and-trust-locations>FilesOrKeyStoreProviders</identity-and-trust-locations>
  </ssl>
  <listen-port>5472</listen-port>
  <tunneling-enabled>true</tunneling-enabled>
</server>

<server>
  <name>wtcMServer1</name>
  <native-io-enabled>true</native-io-enabled>
  <ssl>
    <name>wtcMServer1</name>
    <identity-and-trust-locations>FilesOrKeyStoreProviders</identity-and-trust-locations>
  </ssl>
  <listen-port>7701</listen-port>
  <cluster>wtcCluster</cluster>
  <listen-address>mymachine</listen-address>
  <tunneling-enabled>true</tunneling-enabled>
  <jta-migratable-target>
    <user-preferred-server>wtcMServer1</user-preferred-server>
    <cluster>wtcCluster</cluster>
  </jta-migratable-target>
</server>

<server>
  <name>wtcMServer2</name>
  <native-io-enabled>true</native-io-enabled>
  <ssl>
    <name>wtcMServer2</name>
    <identity-and-trust-locations>FilesOrKeyStoreProviders</identity-and-trust-locations>
  </ssl>
  <listen-port>7702</listen-port>
  <cluster>wtcCluster</cluster>
  <listen-address>mymachine</listen-address>
  <tunneling-enabled>true</tunneling-enabled>
  <jta-migratable-target>
    <user-preferred-server>wtcMServer2</user-preferred-server>
    <cluster>wtcCluster</cluster>
  </jta-migratable-target>
</server>

<server>
  <name>wtcMServer3</name>
  <native-io-enabled>true</native-io-enabled>
```
<ssl>
  <name>wtcMServer3</name>
  <identity-and-trust-locations>FilesOrKeyStoreProviders</identity-and-trust-locations>
</ssl>
  <listen-port>7703</listen-port>
  <cluster>wtcCluster</cluster>
  <listen-address>mymachine</listen-address>
  <tunneling-enabled>true</tunneling-enabled>
  <jta-migratable-target>
    <user-preferred-server>wtcMServer3</user-preferred-server>
    <cluster>wtcCluster</cluster>
  </jta-migratable-target>
</server>

<cluster>
  <name>wtcCluster</name>
  <multicast-address>239.0.0.20</multicast-address>
  <multicast-port>7700</multicast-port>
  <multicast-ttl>1</multicast-ttl>
</cluster>

<configuration-version>9.0.0.0</configuration-version>

<file-realm>
  <name>wl_default_file_realm</name>
</file-realm>

<realm>
  <name>wl_default_realm</name>
  <file-realm>wl_default_file_realm</file-realm>
</realm>

<password-policy>
  <name>wl_default_password_policy</name>
</password-policy>

<migratable-target>
  <name>wtcMServer1 (migratable)</name>
  <user-preferred-server>wtcMServer1</user-preferred-server>
  <cluster>wtcCluster</cluster>
</migratable-target>

<migratable-target>
  <name>wtcMServer2 (migratable)</name>
  <user-preferred-server>wtcMServer2</user-preferred-server>
  <cluster>wtcCluster</cluster>
</migratable-target>

<migratable-target>
  <name>wtcMServer3 (migratable)</name>
  <user-preferred-server>wtcMServer3</user-preferred-server>
  <cluster>wtcCluster</cluster>
</migratable-target>

<web-app-container>
  <relogin-enabled>true</relogin-enabled>
  <allow-all-roles>true</allow-all-roles>
</web-app-container>
How to Configure Inbound Requests from Tuxedo Domains

Load balancing and failover of inbound requests from Tuxedo depend on the Tuxedo domain DMCONFIG configuration.

Load Balancing

Note: For more information on load balancing for the Tuxedo environment, see Tuxedo Load Balancing.

The following is a sample Tuxedo DMCONFIG that load balances from Tuxedo to clustered WTC. This configuration has three nodes in a WebLogic Server cluster. Each node has a single properly configured WebLogic Tuxedo Connector instance that provides an exported service that is accessible to the Tuxedo client.

```
*DM_IMPORT
TOUPPER LDOM=tuxedo_dom RDOM=WDOM1 LOAD=50
TOUPPER LDOM=tuxedo_dom RDOM=WDOM2 LOAD=50
TOUPPER LDOM=tuxedo_dom RDOM=WDOM3 LOAD=50
```

Fail Over

Notes: For more information on failover with Tuxedo Domains, see Specifying Domains Failover and Failback on Tuxedo.

The following is a sample Tuxedo DMCONFIG that uses a more sophisticated configuration that load balances between the WebLogic Server nodes as well as illustrate Tuxedo failover capability. The Tuxedo domain must be configured with a Connection Policy of On Startup or Incoming Only to enable Domains-level failover/failback.

```
*DM_IMPORT
```
TOUPPER LDOM=tuxedo_dom RDOM=WDOM1, WDOM2, WDOM3 LOAD=50
TOUPPER LDOM=tuxedo_dom RDOM=WDOM2, WDOM3, WDOM1 LOAD=50
TOUPPER LDOM=tuxedo_dom RDOM=WDOM3, WDOM1, WDOM2 LOAD=50
CHAPTER 6

How to Configure the Tuxedo Queuing Bridge

The following sections provide information on the Tuxedo Queuing Bridge functionality and configuration.

- Overview of the Tuxedo Queuing Bridge
- Configuring the Tuxedo Queuing Bridge
- Tuxedo Queuing Bridge Connectivity
- Example Connection Type Configurations
- Priority Mapping
- Error Queues

Overview of the Tuxedo Queuing Bridge

The Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge:

- Connectivity is determined by the configuration of the attributes in the Tuxedo Queuing Bridge and Redirections of your WTC Service.
- The Tuxedo Queuing Bridge uses Java Messaging Service (JMS) to provide an interface to a Tuxedo /Q or a Tuxedo service.
- The Tuxedo Queuing Bridge provides simple translation between XML and FML32 to provide connectivity to existing Tuxedo systems.

**How Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge Converter.

3. The Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge 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, Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge processes the next available unacknowledged message.

How Tuxedo Queuing Bridge connects Tuxedo to JMS

Note: Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge to a JMS queue using the TuxQ2JmsQ redirect.
1. Tuxedo Queuing Bridge polls the Tuxedo queue for available messages.

2. A Tuxedo service places a message on a Tuxedo queue.

3. Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge 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.

**Tuxedo Queuing Bridge Limitations**

The Tuxedo Queuing Bridge 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.

- Tuxedo Queuing Bridge 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.

**Configuring the Tuxedo Queuing Bridge**

Tuxedo Queuing Bridge connectivity is determined by configuring the attributes in the Tuxedo Queuing Bridge and Redirections of your WTC Service. These attributes contain the necessary information to establish a connection to Tuxedo.

**Starting the Tuxedo Queuing Bridge**

The Tuxedo Queuing Bridge is started as part of the WebLogic Server application environment if the Tuxedo Queuing Bridge and Redirections of your WTC Service are configured and the WTC Service is deployed to a target server. Any configuration condition that prevents the Tuxedo Queuing Bridge from starting results in an error being logged.
Error Logging

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

Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge, a log entry is generated indicating this is an unsupported type and the message is discarded.

The Tuxedo Queuing Bridge 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 by the Tuxedo Queuing Bridge and Redirections configurations of your WTC Service and provides a one-to-one connection between the identified points. Three types of connections can be configured. 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 an example configuration in the config.xml file for reading from a JMS queue and sending to Tuxedo /Q.

```xml
<wtc-tbridge-redirect>
  <direction>JmsQ2TuxQ</direction>
  <name>redir0</name>
  <reply-q>RPLYQ</reply-q>
  <source-name>weblogic.jms.Jms2TuxQueue</source-name>
  <target-access-point>TDOM2</target-access-point>
  <target-name>STRING</target-name>
  <target-qspace>QSPACE</target-qspace>
</wtc-tbridge-redirect>
```
The following section describes the components of the JmsQ2TuxQ configuration:

- **Direction** connection type is `JmsQ2TuxQ`.
- **Source Name** specifies the name of the JMS queue to read is `weblogic.jms.Jms2TuxQueue`. The Tuxedo Queuing Bridge establishes a JMS client session to this queue using `CLIENT_ACKNOWLEDGE` semantics.
- **Target Access Point** specifies the name of the access point is `TDOM2`.
- **Target Qspace** specifies the name of the Qspace is `Qspace`.
- **Target Name** specifies the name of the queue is `STRING`.
- **ReplyQ** specifies the name of a JMS reply queue is `RPLYQ`. Use of this queue causes `tpenqueue` to provide `TMFORWARD` functionality.
- **TranslateFML** set to NO specifies that no data translation is provided by the Tuxedo Queuing Bridge.

The following table provides information on JmsQtoTuxQ message mapping:

<table>
<thead>
<tr>
<th>From: JMS Message Type</th>
<th>To: WebLogic Tuxedo Connector JATMI (Tuxedo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesMessage</td>
<td>TypedCArray</td>
</tr>
<tr>
<td>TextMessage (translateFML = NONE)</td>
<td>TypedString</td>
</tr>
<tr>
<td>TextMessage (translateFML = FLAT)</td>
<td>TypedFML32</td>
</tr>
</tbody>
</table>

**Example TuxQ2JmsQ Configuration**

The following section provides an example configuration in the `config.xml` file for reading from a Tuxedo /Q and sending to a JMS queue.

```xml
<wtc-tbridge-redirect>
  <direction>TuxQ2JmsQ</direction>
  <name>redir1</name>
  <source-access-point>TDOM2</source-access-point>
  <source-name>STRING</source-name>
  <source-qspace>QSPACE</source-qspace>
</wtc-tbridge-redirect>
```
The following section describes the components of the `TuxQ2JmsQ` configuration:

- **Direction** connection type is `TuxQ2JmsQ`.
- **Target Name** specifies the name of the JMS queue to read is `weblogic.jms.Tux2JmsQueue`.
- **Source Access Point** specifies the name of the access point is `TDOM2`.
- **Source Qspace** specifies the name of the Qspace is `Qspace`.
- **Source Name** specifies the name of the queue is `STRING`.
- **TranslateFML** set to NO specifies that no data translation is provided by the Tuxedo Queuing Bridge.
- **TranslateFML** set to Flat specifies that the data is translated from FML to XML by the Tuxedo Queuing Bridge.

The following table provides information on TuxQ2JmsQ message mapping:

<table>
<thead>
<tr>
<th>From: WebLogic Tuxedo Connector JATMI (Tuxedo)</th>
<th>To: JMS Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypedCArray</td>
<td>BytesMessage</td>
</tr>
<tr>
<td>TypedString (translateFML = NO)</td>
<td>TextMessage</td>
</tr>
<tr>
<td>TypedFML32 (translateFML = FLAT)</td>
<td>TextMessage</td>
</tr>
<tr>
<td>TypedFML (translateFML = FLAT)</td>
<td>TextMessage</td>
</tr>
<tr>
<td>TypedXML</td>
<td>TextMessage</td>
</tr>
</tbody>
</table>

**Example JmsQ2TuxS Configuration**

**Note:** For more information on XML/FML conversion, see Using FML with WebLogic Tuxedo Connector at [http://e-docs.bea.com/wls/docs90/wtc_atmi/XML_FML.html](http://e-docs.bea.com/wls/docs90/wtc_atmi/XML_FML.html).
The following section provides an example configuration in the config.xml file for reading from a JMS queue, calling a Tuxedo service, and then writing the results back to a JMS queue.

```
<wtc-tbridge-redirect>
  <direction>JmsQ2TuxS</direction>
  <name>redir0</name>
  <replyq>weblogic.jms.Tux2JmsQueue</replyq>
  <source-name>weblogic.jms.Jms2TuxQueue</source-name>
  <target-access-point>TDOM2</target-access-point>
  <target-name>TOUPPER</target-name>
  <translate-fml>FLAT</translate-fml>
</wtc-tbridge-redirect>
```

The following section describes the components of the JmsQ2TuxS configuration:

- **Direction** connection type is `JmsQ2TuxS`.
- **Source Name** specifies the name of the JMS queue to read is `weblogic.jms.Jms2TuxQueue`.
- **Target Access Point** specifies the name of the access point is `TDOM2`.
- **Target Name** specifies the name of the queue is `TOUPPER`.
- **ReplyQ** specifies the name of the JMS reply queue is `weblogic.jms.Tux2JmsQueue`.
- **TranslateFML** set to `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 `TOUPPER`. 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:

<table>
<thead>
<tr>
<th>JMS Message Type</th>
<th>WebLogic Tuxedo Connector JATMI (Tuxedo)</th>
<th>JMS Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesMessage</td>
<td>TypedCArray</td>
<td>BytesMessage</td>
</tr>
<tr>
<td>TextMessage</td>
<td>TypedString</td>
<td>TextMessage</td>
</tr>
<tr>
<td>(translateFML = NONE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TextMessage</td>
<td>TypedFML32</td>
<td>TextMessage</td>
</tr>
<tr>
<td>(translateFML = FLAT)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following section provides an example configuration in the config.xml file for reading from a JMS queue, calling a Tuxedo service, and then writing the results back to a JMS queue.

```
<wtc-tbridge-redirect>
  <direction>JmsQ2TuxS</direction>
  <name>redir0</name>
  <replyq>weblogic.jms.Tux2JmsQueue</replyq>
  <source-name>weblogic.jms.Jms2TuxQueue</source-name>
  <target-access-point>TDOM2</target-access-point>
  <target-name>TOUPPER</target-name>
  <translate-fml>FLAT</translate-fml>
</wtc-tbridge-redirect>
```

The following section describes the components of the JmsQ2TuxS configuration:

- **Direction** connection type is `JmsQ2TuxS`.
- **Source Name** specifies the name of the JMS queue to read is `weblogic.jms.Jms2TuxQueue`.
- **Target Access Point** specifies the name of the access point is `TDOM2`.
- **Target Name** specifies the name of the queue is `TOUPPER`.
- **ReplyQ** specifies the name of the JMS reply queue is `weblogic.jms.Tux2JmsQueue`.
- **TranslateFML** set to `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 `TOUPPER`. 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:

<table>
<thead>
<tr>
<th>JMS Message Type</th>
<th>WebLogic Tuxedo Connector JATMI (Tuxedo)</th>
<th>JMS Message Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BytesMessage</td>
<td>TypedCArray</td>
<td>BytesMessage</td>
</tr>
<tr>
<td>TextMessage</td>
<td>TypedString</td>
<td>TextMessage</td>
</tr>
<tr>
<td>(translateFML = NONE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TextMessage</td>
<td>TypedFML32</td>
<td>TextMessage</td>
</tr>
<tr>
<td>(translateFML = FLAT)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Priority Mapping

WebLogic Tuxedo Connector supports multiple Tuxedo Queuing Bridge 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 Tuxedo Queuing Bridge redirect instance.

Use Priority Mapping to map priorities between the JMS and Tuxedo.

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

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

- JmstoTux
- TuxtoJms

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

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

\[
\begin{align*}
\text{JmstoTux} & : 0:1 | 1:12 | 2:23 \mid 3:34 \mid 4:45 \mid 5:56 \mid 6:67 \mid 7:78 \mid 8:89 \mid 9:100 \\
\text{TuxtoJms} & : 1-10:0 | 11-20:1 | 21-30:2 \mid 31-40:3 \mid 41-50:4 \mid 51-60:5 \mid 61-70:6 \mid 71-80:7 \mid 81-90:8 \mid 91-100:9
\end{align*}
\]

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 Tuxedo Queuing Bridge 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.
WLS Error Destination

The WLS Error Destination 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 Tuxedo Queuing Bridge does not redirect the message to the error queue.

Tuxedo Error Queue

The Tuxedo Error Queue is the failure queue for the JATMI primitive tpdequeue during a TuxQ2JmsQ redirect.

Limitations

The Tuxedo Queuing Bridge error queues have the following limitations:

- Tuxedo Error Destination 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 drop messages on error. All messages are received or none are received.

- Information about the error is only available in the server log.
Connecting WebLogic Integration and Tuxedo Applications

Note: For more information on how to integrate applications, see BEA WebLogic Integration at http://e-docs.bea.com/wli/docs81/index.html.

The WebLogic Tuxedo Connector Tuxedo Queuing Bridge provides the necessary infrastructure for WebLogic Integration users to integrate Tuxedo applications into their business workflows. The following sections discuss WebLogic Integration - Tuxedo interoperability using the WebLogic Tuxedo Connector.

- Synchronous WebLogic Integration-to-Tuxedo Connectivity
- Synchronous Non-Blocking WebLogic Integration-to-Tuxedo Connectivity
- Asynchronous WebLogic Integration-to-Tuxedo Connectivity
- Asynchronous Tuxedo /Q-to-WebLogic Integration Connectivity
- Bi-directional Asynchronous Tuxedo-to-WebLogic Integration Connectivity

Synchronous WebLogic Integration-to-Tuxedo Connectivity

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

- Defining WebLogic Integration Business Operations.
- Invoking an eLink Adapter.
- Defining WebLogic Integration Exception Handlers.
Defining Business Operations
Define WebLogic Integration 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 Integration process flow:

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

Define Exception handlers
Define WebLogic Integration Exception handlers to process exceptions.

Synchronous Non-Blocking WebLogic Integration-to-Tuxedo Connectivity
WebLogic Integration 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 Integration writes a message to JMS queue.
- Once the message is on the JMS queue then Tuxedo Queuing Bridge 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 Integration event node waits on the response queue for a response message.
Asynchronous WebLogic Integration-to-Tuxedo Connectivity

WebLogic Integration sends a guaranteed asynchronous message to a Tuxedo /Q:

- 1:1 relationship between JMS queue and Tuxedo /Q.
- WebLogic Integration writes a message to JMS queue.
- Once the message is on the JMS queue then Tuxedo Queuing Bridge 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 Integration Connectivity

Tuxedo /Q sends a guaranteed asynchronous message to WebLogic Integration:

- 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 Tuxedo Queuing Bridge 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 Integration Connectivity

Tuxedo executes a blocking invocation of a WebLogic Integration process flow. Use two asynchronous instances to connect from JMS to Tuxedo /Q and from Tuxedo /Q back to JMS.
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. To record log information you must:

- Set Trace Levels
- Enable Debug Mode
- Enable a User Data Dump

**Set Trace Levels**

*Note:* For more information about setting WebLogic Server properties, see “How to Set WebLogic Tuxedo Connector Properties” on page 2-7.

To enable tracing, update the `JAVA_OPTIONS` variable in your server start script to the desired level.

Example:

```
JAVA_OPTIONS=-Dweblogic.wtc.TraceLevel=100000
```
Use the following values to set the TraceLevel:

<table>
<thead>
<tr>
<th>Value</th>
<th>Components Traced</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>TBRIDGE_IO</td>
<td>Tuxedo Queuing Bridge input and output</td>
</tr>
<tr>
<td>15000</td>
<td>TBRIDGE_EX</td>
<td>more Tuxedo Queuing Bridge information</td>
</tr>
<tr>
<td>20000</td>
<td>GWT_IO</td>
<td>Gateway input and output, including the ATMI verbs</td>
</tr>
<tr>
<td>25000</td>
<td>GWT_EX</td>
<td>more Gateway information</td>
</tr>
<tr>
<td>50000</td>
<td>JAMTI_IO</td>
<td>JAMTI input and output, including low-level JAMTI calls</td>
</tr>
<tr>
<td>55000</td>
<td>JAMTI_EX</td>
<td>more JAMTI information</td>
</tr>
<tr>
<td>60000</td>
<td>CORBA_IO</td>
<td>CORBA input and output</td>
</tr>
<tr>
<td>65000</td>
<td>CORBA_EX</td>
<td>more CORBA information</td>
</tr>
<tr>
<td>100000</td>
<td>All Components</td>
<td>information on all WebLogic Tuxedo Connector components</td>
</tr>
</tbody>
</table>

**Enable Debug Mode**

Use the following procedure to specify that trace information is written to the log file:

1. Click the Server node in the left pane.
2. Select your server in the left pane.
3. Select the Logging tab.
4. In the General tab:
   a. Check **Debug to Stdout**
   b. Set **Stdout severity threshold** to **Info**.
Enable a User Data Dump

To enable dumping of user data, add the following line to the java.weblogic.Server command.

```
JAVA_OPTIONS=-Dweblogic.debug.DebugWTCUData=true
```

Enabling this causes user data to be dumped after the connection is connected. If no other debugging properties are enabled, then this will be the only WTC information dumped, except normal WTC error/informational messages. The dump is available in the WLS server log file.

The dump has the following format.

- For outbound messages
  
  Outbound UDATA: buffer type (<type>, <subtype>)
  +++++ User Data(size) +++++
  ......

- For inbound messages

  Inbound UDATA: buffer type (<type>, <subtype>)
  +++++ User Data(size) +++++
  ......

Frequently Asked Questions

This section provides solutions to common user questions.

What does this EJB Deployment Message Mean?

When I build the simpser 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.
How do I Start the Connector?

Releases prior to WebLogic Server 7.0 used a WebLogic Server Startup class to start a WebLogic Tuxedo Connector session and a WebLogic Server Shutdown class to end a session. In WebLogic Server 8.1, WebLogic Tuxedo Connector sessions are managed using a WTC Service.

- A WebLogic Tuxedo Connector session is started when a configured WTC Service is assigned to a selected server.

- A WebLogic Tuxedo Connector session is ended by removing a WTC Service from the WebLogic server or when you shutdown the WebLogic server.

How do I Start the Tuxedo Queuing Bridge?

The Tuxedo Queuing Bridge is started if the Tuxedo Queuing Bridge and Redirections configurations exist in your WTC Service and the WTC Service is assigned to a selected server.

How do I Assign a WTC Service to a Server?

The console displays an exception when I try to assign my WTC Service to a server. What should I do?

Make sure you have a valid WTC Service configured. Each WTC Service must have 1 or more Local Tuxedo Access Points configured before it can be assigned to a server. Your server log will display the following:

<!-- Apr 22, 2002 4:21:35 PM EDT > Error > WTC > 180101 > At least one local domain has to be defined. -->

How do I Resolve Connection Problems?

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

- Make sure you have started your Tuxedo server.

- Set the $TraceLevel and enable Debug mode. Repeat the connectivity test and check the WebLogic Tuxedo Connector and Tuxedo log files for error messages.

- Avoid using machine names or localhost. Always use an IP address when specifying a network location.
• Check your `AclPolicy` and `CredentialPolicy` attributes. If your `AclPolicy` is `LOCAL`, you must register the remote domain `DOMAINID` as a WebLogic Server user. For more information, see “User Authentication” on page 3-15.

• If you are migrating from WebLogic Server 6.x and your applications use security, you need to set `PasswordKey` as a WebLogic Server property. For more information, see “How to Set WebLogic Tuxedo Connector Properties” on page 2-7.

• 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 your Remote Tuxedo Access Point `Access Point Id` attribute.

• Request assistance from BEA Customer Support.

How do I Migrate from Previous Releases?

You must make some changes in your WebLogic Tuxedo Connector 6.x applications (including WebLogic Tuxedo Connector 1.0) to use them with WebLogic Server 9.0. For more information, see Upgrading WebLogic Application Environments at http://e-docs.bea.com/wls/docs90/../../common/docs90/upgrade/index.html.