Oracle® Fusion Middleware

Administering WebLogic Tuxedo Connector for Oracle WebLogic Server

14*c* (14.1.2.0.0)
F61298-01
December 2024

Oracle Fusion Middleware Administering WebLogic Tuxedo Connector for Oracle WebLogic Server, 14*c* (14.1.2.0.0)

F61298-01

Contents

3 Oracle WebLogic Tuxedo Connector Administration

4 Controlling Oracle WebLogic Tuxedo Connector Connections and Services

5 Administration of CORBA Applications

6 How to Manage Oracle WebLogic Tuxedo Connector in a Clustered Environment

7 How to Configure the Oracle Tuxedo Queuing Bridge

8 Connecting WebLogic Integration and Tuxedo Applications

9 Troubleshooting The WebLogic Tuxedo Connector

Index

Preface

This document provides information on how to configure and administer the Oracle WebLogic Tuxedo Connector to interoperate between Oracle WebLogic Server and Oracle Tuxedo.

Audience

This document is written for WebLogic Server administrators and application developers who understand Oracle WebLogic Tuxedo Connector application environment.

It is assumed that readers are familiar with Web technologies and the operating system and platform where WebLogic Server is installed.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

The Oracle 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 Java site at `http://www.oracle.com/technetwork/java/index.html`.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction to Oracle WebLogic Tuxedo Connector

This chapter summarizes the concepts and functionality of Oracle WebLogic Tuxedo Connector for this release of WebLogic Server.
This chapter includes the following sections:

Oracle WebLogic Tuxedo Connector Overview

The Oracle 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 Jakarta Enterprise Beans (EJBs) in response to a service request.

> **Note:**
>
> The Oracle WebLogic Tuxedo Connector uses an interoperability protocol that provides interoperability across many combinations of WebLogic Server and Tuxedo versions. However, Oracle recommends that you use WebLogic Server and Tuxedo versions that are currently supported with error correction, in order to receive full support and bug fixes for the interoperability provided by WebLogic Tuxedo Connector.

Key Functionality and Administrative Features

The Oracle 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 Oracle WebLogic Tuxedo Connector tBridge functionality provides Tuxedo /Q and JMS advanced messaging services.

The Oracle 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 Oracle WebLogic Tuxedo Connector includes the following key administration features:

- Simple implementation. The Oracle WebLogic Tuxedo Connector does not require modification of existing Tuxedo application code.
  - Existing Tuxedo clients call WebLogic Server EJBs through the Oracle WebLogic Tuxedo Connector.
  - New or modified WebLogic Server clients call Tuxedo services through Oracle 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

Oracle 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.

How Oracle WebLogic Tuxedo Connector Differs from Jolt

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

- Oracle WebLogic Tuxedo Connector offers a similar but different API than Jolt.
- Jolt enables the development of generic Jakarta clients and other Web server applications that the Oracle 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 Oracle WebLogic Tuxedo Connector when a generic Jakarta client or other Web server application is required and WebLogic Server is not part of the solution.

Platform Support

See the Oracle Fusion Middleware Supported System Configurations page on the Oracle Technology Network for more information on object references.

New and Changed Features in This Release

See *What's New in Oracle WebLogic Server* for a comprehensive listing of the new WebLogic Server features introduced in this release.

2

Configuring Oracle WebLogic Tuxedo Connector

This chapter describes how to configure the Oracle WebLogic Tuxedo Connector. This chapter includes the following sections:

- Summary of Environment Changes and Considerations
- Configuring Oracle 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 Oracle Tuxedo and Oracle WebLogic Server environments before you can start using the Oracle WebLogic Tuxedo Connector.

- Oracle Tuxedo Changes
- WebLogic Server Changes

Oracle Tuxedo Changes

Tuxedo users need to make the following environment changes:

- If an existing Tuxedo application is already using Tuxedo `/T DOMAINS`, then a new domain must be added to the domains configuration file for each connection to an Oracle 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 Oracle WebLogic Tuxedo Connector instantiation.
- Oracle WebLogic Tuxedo Connector requires that the Oracle Tuxedo domain always have encoding turned on. `MTYPE` should always be unset, or set to NULL, or set to a value different from the `MTYPE` in the `DM_LOCAL_DOMAINS` section in the `DMCONFIG` file.

See Using the Tuxedo Domains Component.

WebLogic Server Changes

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

- Administration and Programming
- WebLogic Server Threads

Administration and Programming

WebLogic Server users need to make the following environment changes:

- Create Jakarta clients or servers. See *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server*.
- Configure the Oracle WebLogic Tuxedo Connector using the WebLogic Remote Console, command-line interface, or WLST. See Configuring Oracle WebLogic Tuxedo Connector for Your Applications.
- If the Oracle 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. See User Authentication .

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 Oracle WebLogic Tuxedo Connector, there is no Oracle 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.

> **Note:**
>
> AWTC server uses three threads plus one thread for every local access point defined.

Configuring Oracle WebLogic Tuxedo Connector for Your Applications

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

- Oracle WebLogic Tuxedo Connector MBean Classes
- Configuring Oracle WebLogic Tuxedo Connector Using the Remote Console
- Configuring Oracle WebLogic Tuxedo Connector Using the Command-Line Interface
- Set the WebLogic Server Environment
- How to Set Oracle WebLogic Tuxedo Connector Properties
- System Level Debug Settings
- Oracle WebLogic Tuxedo Connector Configuration Guidelines

Oracle WebLogic Tuxedo Connector MBean Classes

The Oracle 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`. Table 2-1 lists the MBean types used to configure Oracle WebLogic Tuxedo Connector:

Table 2-1 MBean Types Used to Configure Oracle WebLogic Tuxedo Connector

MBean Type	Description
WTCServer	Parent MBean containing the interoperability attributes required for a connection between WebLogic Server and Tuxedo.
WTCLocalTuxDom	Provides configuration information to connect available remote Tuxedo domains to a WTC server. You must configure at least one local Tuxedo access point. **Note:** Because of dynamic configuration, you can create and deploy an empty WTC server.
WTCRemoteTuxDom	Provides configuration information to connect a WTC server to available remote Tuxedo domains. You may configure multiple remote domains.
WTCExport	Provides information on services exported by a local Tuxedo access point.
WTCImport	Provides information on services imported and available on remote domains.
WTCResources	Specifies global field table classes, view table classes, and application passwords for domains. Support for MBSTRING is provided using `RemoteMBEncoding` and `MBEncodingMapFile` attributes.
WTCPassword	Specifies the configuration information for inter-domain authentication.
WTCtBridgeGlobal	Specifies global configuration information for the transfer of messages between WebLogic Server and Tuxedo.
WTCtBridgeRedirect	Specifies the source, target, direction, and transport of messages between WebLogic Server and Tuxedo.

For more information on the Oracle WebLogic Server management and the `config.xml` file, see *MBean Reference for Oracle WebLogic Server*.

Configuring Oracle WebLogic Tuxedo Connector Using the Remote Console

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

1. In the **Edit Tree**, go to **Interoperability**, then **WTC Servers**.
2. Create or modify the WTC server you want to configure.

Table 2-2 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 2-2 Oracle WebLogic Tuxedo Connector Configuration Tasks

Task #	Task	Description
1	Create a WTC server.	Set the Name attribute.
2	Create local access points.	Set the attributes that describe your local Tuxedo access point on the General, Connections, and Security pages. You must configure at least one local Tuxedo access point. **Note:** Because of dynamic configuration, you can create and deploy an empty WTC server.

Table 2-2 (Cont.) Oracle WebLogic Tuxedo Connector Configuration Tasks

Task #	Task	Description
3	Create remote access points.	Set the attributes that describe your remote Tuxedo domains on the Remote APs page.
4	Create exported services.	Set the attributes that describe your exported WebLogic Server services on the Exported page.
5	Create imported services.	Set the attributes that describe your imported Tuxedo services on the Imported page.
6	Create password configurations.	Set the attributes that describe your passwords on the Passwords page.
7	Create resources.	Set the attributes that describe your Oracle WebLogic Tuxedo Connector resources on the Resources page.
8	Create queuing bridge connections.	Set the global configuration information for the transfer of messages between WebLogic Server and Tuxedo.
9	Create Tuxedo queuing bridge redirections.	Sets the attributes used to specify the source, target, direction, and transport of a message between WebLogic Server and Tuxedo
10	Target WTC servers.	Select a target server instance for your WTC server.

Configuring Oracle WebLogic Tuxedo Connector Using the Command-Line Interface

The command-line interface provides a way to create and manage Oracle WebLogic Tuxedo Connector connections. See *Understanding the WebLogic Scripting Tool*.

Set the WebLogic Server Environment

You need to set the environment of your WebLogic Server application by running the `setExamplesEnv` script located at *`ORACLE_HOME`*`/user_projects/domains/<wls_examples>`, where *`ORACLE_HOME`* is the directory you specified as the Oracle Home when you installed Oracle WebLogic and <*wls_examples*> is the directory name that you provided for the WLS Code 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 *`ORACLE_HOME`*`/user_projects/domains/<wl_server>` 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.

How to Set Oracle 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

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

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

where *mykey* is the key value.

See Configuring a Password Configuration.

Set encoding

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

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

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

where *codesetname* is the name of a supported codeset used by a remote Tuxedo domain. See Supported Encodings at `http://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html` for list of supported base and extended coding sets.

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

Example:

- The Supported Encoding list includes `EUC_JP`
- The remote domain is supported by a Solaris operating system using `eucJP`

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

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

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 00 ....strings.....
+++++ END +++++

Outbound UDATA: buffer type (String, null)
+++++ User Data(12) +++++
00 00 00 07 53 54 52 49 4E 47 53 00 ....STRINGS.
+++++ END +++++
```

- Enable IPv4 for SDP transport

Enable IPv4 for SDP transport

To use Socket Direct Protocol (SDP), set the system property `-Djava.net.preferIPv4Stack=true` in the `JAVA_OPTIONS` variable in your server start script.

For detailed information on how to configure WTC to use SDP to interoperate with Tuxedo, see the *Oracle Tuxedo/Oracle Exalogic Environment Deployment Guide*, 11*g*R1PS2.

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
  ```

Oracle WebLogic Tuxedo Connector Configuration Guidelines

Use the following guidelines when configuring Oracle WebLogic Tuxedo Connector:

- You may have more than one WTC server in your configuration.
- You cannot target two or more WTC servers to the same server instance. A server instance can only be a target for one WTC server.
- Some configuration changes implemented in a WTC server after a target server instance is selected will not be updated in the target server instance. You must remove the WTC server from the server instance and then add the updated WTC server to the target server instance.

  For example, changes to tBridge requires you to undeploy and then deploy the WTC server to make configuration changes effective. However, some configuration changes, such as `KeepAlive`, `KeepAliveWait` and `RetryInterval`, take effect when you activate the change.

3

Oracle WebLogic Tuxedo Connector Administration

This chapter describes how to configure Oracle WebLogic Tuxedo Connector and establish connectivity, and provide security between WebLogic Server applications and Tuxedo environments. Oracle WebLogic Tuxedo Connector uses attributes that are analogous to the interoperability attributes required for the communication between Tuxedo access points.

For more information on the Oracle WebLogic Server management, including the Oracle WebLogic Tuxedo Connector, see the *MBean Reference for Oracle WebLogic Server*.

This chapter includes the following sections:

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 on the Connections page of the local Tuxedo access points and remote Tuxedo access points configurations of your WTC server.

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 3-1 Access Point Connection Policy Settings

If the Local Access Point Setting is...	And the Remote Access Point Setting is	Then the Actual Connection Policy Is...
ON_DEMAND	ON_DEMAND	ON_DEMAND
ON_DEMAND	ON_STARTUP	ON_STARTUP
ON_DEMAND	INCOMING_ONLY	INCOMING_ONLY
ON_DEMAND	LOCAL	ON_DEMAND
ON_STARTUP	ON_DEMAND	ON_DEMAND
ON_STARTUP	ON_STARTUP	ON_STARTUP
ON_STARTUP	INCOMING_ONLY	INCOMING_ONLY
ON_STARTUP	LOCAL	ON_STARTUP
INCOMING_ONLY	ON_DEMAND	ON_DEMAND
INCOMING_ONLY	ON_STARTUP	ON_STARTUP
INCOMING_ONLY	INCOMING_ONLY	INCOMING_ONLY
INCOMING_ONLY	LOCAL	INCOMING_ONLY

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 3-2 Interaction of Local and Remote Access Point Connection Policies

If the Local System's Effective Connection Policy Is...	And the Remote System's Effective Connection Policy Is...	Then the Settings of the Parameters at the Remote Domain Are...
ON_DEMAND	ON_DEMAND	ON_DEMAND from either
ON_DEMAND	ON_STARTUP	ON_STARTUP when both are up
ON_DEMAND	INCOMING_ONLY	ON_DEMAND from local
ON_STARTUP	ON_DEMAND	ON_STARTUP when both are up
ON_STARTUP	ON_STARTUP	ON_STARTUP when both are up
ON_STARTUP	INCOMING_ONLY	ON_STARTUP when both are up
INCOMING_ONLY	ON_DEMAND	ON_DEMAND from remote
INCOMING_ONLY	ON_STARTUP	ON_STARTUP when both are up
INCOMING_ONLY	INCOMING_ONLY	manual connect only when both are up

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

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 on the Connections page of your local Tuxedo access point to `On Startup`.

- How to Configure RetryInterval
- How to Configure MaxRetries

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

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.

Use this only when `ConnectionPolicy` is set to `On Startup`. For other connection policies, retry processing is disabled.

Table 3-3 Example Settings of MaxRetries and RetryInterval Parameters

If you set...	Then...

`ConnectionPolicy: On Startup`
`RetryInterval: 30`
`MaxRetries: 3`
	The access point makes 3 attempts to establish a connection, at 30 seconds intervals, before quitting.

`ConnectionPolicy: On Startup` `MaxRetries: 0`	The access point attempts to establish a connection at initialization time but does not retry if the first attempt fails.
`ConnectionPolicy: On Startup` `RetryInterval: 30`	The access point attempts to establish a connection every 30 seconds until a connection is established.

How to Request Connections for Client Demands (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.

> **Note:**
>
> If the `ConnectionPolicy` is not specified for the local access point, the Oracle WebLogic Tuxedo Connector uses a `ConnectionPolicy` of `On Demand`.

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

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.

> **Note:**
>
> A `ConnectionPolicy` of `LOCAL` is not valid for local access points.

Configuring Failover and Failback

Oracle 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.

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

- How to Configure Link-level Failover

Prerequisite to Using Failover and Failback

To use failover/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 server:

- 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 server would include two remote Tuxedo access point configurations and two imported services configurations in your WTC server. The WTC server defined in the `config.xml` file would contain the following:

```
<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>myLoclTuxDom</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>
          <local-access-point>TDOM2</local-access-point>
          <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>
           <name>imp0</name>
           <resource-name>TOUPPER</resource-name>
           <local-access-point>TDOM2</local-access-point>
           <remote-access-point-list>TDOM1,TDOM3</remote-access-point-list>
           <remote-name>TOUPPER</remote-name>
      </wtc-import>
      <wtc-import>
           <name>imp1</name>
           <resource-name>TOUPPER</resource-name>
           <local-access-point>TDOM2</local-access-point>
           <name>2ndImportedResources</name>
           <remote-access-point-list>TDOM3,TDOM1</remote-access-point-list>
           <remote-name>TOUPPER</remote-name>
      </wtc-import>
</wtc-server>
```

- How Failback Works

How Failback Works

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 `WTCRemoteTuxDomMBean` 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

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.

Example 3-1 Link-level Failover Configuration

```
<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

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 on the Security page of the local Tuxedo access point configuration of your WTC server. 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 server. 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 server. You use the `weblogic.wtc.gwt.genpasswd` utility to create encrypted local and remote passwords.

The `Security` attribute on the Security page of the local Tuxedo access point of your WTC server 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 server does not match the security type of the *DM_LOCAL_DOMAINS or if the passwords do not match, the connection fails.

See the following sections:

- Configuring a Password Configuration
- Generating Encrypted Passwords
- Usage
- Examples

Configuring a Password Configuration

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.

See How to Set Oracle WebLogic Tuxedo Connector Properties.

- Using AES Encrypted Passwords

Using AES Encrypted Passwords

This release of WebLogic Server provides the ability to optionally support 256-bit AES encryption of passwords by setting the `-Daes` flag in the `weblogic.wtc.gwt.genpasswd` utility (see Generating Encrypted Passwords). This is equivalent to setting the `GWT_SNP_MINCRYPT` environment variable to `AES` for a GWTDOMAIN process in Tuxedo.

> **Note:**
>
> For AES support in Tuxedo 10 for GWTDOMAIN, passwords in the `tpusr` file remain encrypted using their original encryption method. `tpusr` files are normally generated in a Tuxedo native domain and then copied to a location where WTC (via the WLS domain) can access it.

Generating Encrypted Passwords

To generate 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 werver.
- The password configuration of your WTC server does not store clear text passwords.
- The key value is a WebLogic Server property.

  `-Dweblogic.wtc.PasswordKey=mykey`

  `PasswordKey` is the attribute used to assign the key

  `mykey` is the key value
  - The `PasswordKey` attribute can only be assigned for one key value. This key value is used for all Oracle 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:

```
$ java weblogic.wtc.gwt.genpasswd
Usage: genpasswd [-Daes] Key <LocalPassword|RemotePassword|AppPassword> <local|remote|
application>
```

Call the utility with a key value, password to encrypt, and the type of password. To generate 256-bit AES encoded password and password IV, include the `-Daes` flag.

Example:

```
$ 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 the password configuration of your WTC server.

```
Local Password   : my_password
Local Password IV: my_passwordIV
```

where

- Cut and paste the string of characters represented by *my_password* into the Password field.
- Cut and paste the string of characters represented by *my_passwordIV* into the PasswordIV field.

Examples

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

- Local Passwords
- Remote Passwords
- App Passwords

Local Passwords

The following example uses `key1` to encrypt `LocalPassword1` as the password of the local access point.

```
$ java weblogic.wtc.gwt.genpasswd key1 LocalPassword1 local
Local Password : FMTCg5Vi1mTGFds1U4GKIQQj7s2uTlg/ldBfy6Kb+yY=
Local Password IV : NAGikshMiTE=
```

Your Password attributes are:

```
Local Password: FMTCg5Vi1mTGFds1U4GKIQQj7s2uTlg/ldBfy6Kb+yY=
Local Password IV: NAGikshMiTE=
```

The following example uses AES encryption to encode `LocalPassword1` as the password of the local access point.

```
$ java weblogic.wtc.gwt.genpasswd -Daes LocalPassword1 local
Local Password   : 71Im/Y4VcuhInuN1My5wWRjIneu+KPR0aN1WBliwIq4=
Local Password IV: a9qAjBpYExA=
```

Your Password attributes are:

```
Local Password: 71Im/Y4VcuhInuN1My5wWRjIneu+KPR0aN1WBliwIq4=
Local Password IV: a9qAjBpYExA=
```

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 : ohYHxzhYHP0=
```

Your Password attributes are:

```
Remote Password: A/DgdJYOJunFUFJa62YmPgsHan8pC02zPT0T7EigaVg=
Remote Password IV: ohYHxzhYHP0=
```

App Passwords

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

```
$ java weblogic.wtc.gwt.genpasswd mykey test123 application
App Password : uou2MALQEZgNqt8abNKiC9ADN5gHDLviqO+Xt/VjakE=
App Password IV : eQuKjOaPfCw=
```

Your Resources attributes are:

```
Application Password: uou2MALQEZgNqt8abNKiC9ADN5gHDLviqO+Xt/VjakE=
Application Password IV: eQuKjOaPfCw=
```

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`

See the following sections:

- ACL Policy is LOCAL
- ACL Policy is GLOBAL
- Remote Access Point Credential Policy is GLOBAL
- Remote Access Point Credential Policy is LOCAL
- User Authentication for Tuxedo 6.5

ACL Policy is LOCAL

If the Oracle 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.

ACL Policy is GLOBAL

If the Oracle 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.

When `CredentialPolicy` is set to `GLOBAL` for WTC, then WLS user credential is propagated from WTC to the remote Tuxedo domain. If a remote Tuxedo domain is also configured with `ACL_POLICY` set to `GLOBAL`, then it will accept the WLS user credential and use it to access Tuxedo services. If a remote Tuxedo domain is configured with `ACL_POLICY` to `LOCAL`, then it will discard the received WLS user credential and use WTC `DOMAINID` to access Tuxedo services.

Remote Access Point Credential Policy is LOCAL

When `CredentialPolicy` is set to `LOCAL` for WTC, then WLS user credential is not propagated to a remote Tuxedo domain. The remote Tuxedo access point sets the identity of a service request received from the WTC domain to be the principal name specified in the local principal name for the remote Tuxedo domain.

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 Oracle WebLogic Tuxedo Connector, you will need to upgrade to Tuxedo 7.1 or higher.

How to Configure Oracle WebLogic Tuxedo Connector to Provide Security between Oracle Tuxedo and Oracle 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 Oracle 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 on the Security page of the local Tuxedo access point of your WTC server 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 page of the remote Tuxedo access point of your WTC server 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. See Anonymous Users.
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. See How to Enable User-Level Authentication.

- Using the Resources TpUsrFile attribute

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 server 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 Remote Console to administer the security information from a single system. Requires Tuxedo 8.1 and higher. Use the following steps to configure Oracle 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

Detailed information on how to implement single point security administration, see Implementing Single Point Security Administration. See *Understanding Security for Oracle WebLogic Server*.

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

Set the `security` attribute on the Security page of the local Tuxedo access point of your WTC server 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 page of the remote Tuxedo access point of your WTC server 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. See Anonymous Users.
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

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

For additional information, see How to Create a Custom AppKey Plug-in in *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server*.

- 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 on the Security page of the local Tuxedo access point of your WTC server 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 page of the remote Tuxedo access point of your WTC server 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. See Anonymous Users.
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 server 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 server starts.

Anonymous Users

The Allow Anonymous attribute on the Security page 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 on the Security page in the local Tuxedo access points and remote Tuxedo access points configurations of your WTC server.

Secure Socket Level Encryption

This release of WebLogic Server supports Secure Socket Level (SSL) encryption for securing communications between domains, including use of trusted certificates and private keys. You configure this security mechanism by setting attributes in the SSL Configuration, KeyStore Configuration, and the `MINENCRYPTBITS` and `MAXENCRYPTBITS` attributes on the Security page in the local Tuxedo access points and remote Tuxedo access points configurations of your WTC server.

4

Controlling Oracle WebLogic Tuxedo Connector Connections and Services

This chapter describes how to control connectivity and services between WebLogic Server applications and Tuxedo environments. Oracle WebLogic Tuxedo Connector uses attributes that are analogous to the interoperability attributes required for the communication between Tuxedo access points.
This chapter includes the following sections:

Dynamic Administration of Connections

You can dynamically list, start, and stop individual connections using the WebLogic Remote Console or WLST scripting language. Refer to the following sections for how to start and stop WTC server connections using the available tools.

Using WebLogic Scripting Tool (WLST)

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 *Understanding the WebLogic Scripting Tool*.

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,

```
cmo.startConnection('WLSDOM','TUXDOM')
```

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

```
cmo.startConnection('WLSDOM')
```

Stopping Connections

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

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

```
cmo.stopConnection('WLSDOM','TUXDOM')
```

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

```
cmo.stopConnection('WLSDOM')
```

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

Example 4-1 Dynamically List, Start, and Stop Connections

```
java weblogic.WLST
wls:/offline> connect('weblogic','weblogic')
wls:/mydomain/serverConifg> 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=cmo.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
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.

Example 4-2 Modifying Configuration Attributes

```
java weblogic.WLST
wls:/offline> connect('weblogic','weblogic')
wls:/mydomain/serverConifg> edit()
wls:/mydomain/edit> startEdit()
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

wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms/TDOM2> save()
wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms/TDOM2> activate(block="true")
wls:/mydomain/edit/WTCServers/myWTC/LocalTuxDoms/TDOM2> disconnect()
wls:/offline> exit()
```

Suspend/Resume WTC Services

Using the WebLogic Remote Console or WLST, an administrator can suspend and resume a service on a specific WTC server. When an imported service is suspended on a WTC server, then all the JATMI client requests sent to the WTC server for that service are returned immediately by WTC throwing a TPException.TPENOENT. The service will not become available until the service is explicitly resumed.

For service requests from a Tuxedo client to WTC that are targeted to a suspended exported service, the service request is returned with TPENOENT without ever invoking the actual services. Any service requests already received and in processing will continue to process and is not affected by the suspend operation.

See Suspend/Resume WTC Services Dynamically

Refer to the following sections for how to suspend and resume WTC services using the available tools.

- Using WebLogic Scripting Tool (WLST)
- Suspend/Resume WTC Services Dynamically

Using WebLogic Scripting Tool (WLST)

WLST allows you to suspend and resume Oracle WebLogic Tuxedo Connector services through the WTCRuntimeMBean. You can also check the status of the service.

- Checking Status of WTC Services
- Suspending WTC Services
- Resuming WTC Services

Checking Status of WTC Services

To determine the status of a service, specify the `SvcName`, `LDOM`, or `RDomList` in the arguments. For example,

```
int WTCService.getServiceStatus(String SvcName)
```

In this case, the code returns a status of all the imported and exported services with the name `SvcName` for the targeted WTC server. If there is more than one imported service or exported service with the same resource name `'SvcName'`, then if at least one service is available, the status will return `AVAILABLE`. If there is more than one imported service or exported service with the same resource name `'svcName'` and some services are suspended and some are unavailable, the status returns a `SUSPENDED` value. If all services are unavailable, the status returns an `UNAVAILABLE` value. `TPException.TPENOENT` is thrown when no match is found.

The legal values of the returned status are as shown in Table 4-1.:

Table 4-1 Status Values for a Service

Status Values	Description
WTCServiceStatus.SUSPENDED	The service is suspended administratively.
WTCServiceStatus.AVAILABLE	The service is not suspended, and is accessible
WTCServiceStatus.UNAVAILABLE	The service is not suspended, but is not accessible because there is no connection available to remote Tuxedo GWTDomain gateway that provides this service.

Suspending WTC Services

You can suspend any imported or exported service advertised by a WTC server. Any service suspended administratively will become available only when either WTC server is redeployed, WLS server is rebooted, or the service is resumed administratively.

To suspend an available service, specify the `SvcName`, `LDOM`, or `RDomList` in the arguments. For example,

```
Void WTCRuntimeMBean.suspendService(String SvcName, boolean isImported)
```

This case suspends all the Import or Export services with the specified name. If `isImported` is true, then only imported services are suspended; if it is false, then only exported services are suspended. `TPException.TPENOENT` is thrown if nothing is found.

Resuming WTC Services

You can resume any imported or exported service advertised by a WTC server that has a status of suspended. Any service suspended administratively will become available only when either WTC server is redeployed, WLS server is rebooted, or the service is resumed administratively.

To resume a suspended service, specify the `SvcName`, `LDOM`, or `RDomList` in the arguments. For example,

```
void WTCRuntimeMBean.resumeService(String SvcName)
```

This example resumes all the Import and Export services with `SvcName` configured for the targeted WTC server. `TPException.TPENOENT` is thrown if no match is found.

Suspend/Resume WTC Services Dynamically

Using the WebLogic Remote Console or WLST, an administrator can suspend and resume a service on a specific WTC server. When an imported service is suspended on a WTC server, then all the JATMI client requests sent to the WTC server for that service are returned immediately by WTC throwing a TPException.TPENOENT. The service will not become available until the service is explicitly resumed.

The dynamic status only affect imported service. When there is at least one TDomain session available or possibly available, then the imported service will become available. It will become suspended only when no TDomain session is available. When connection policy resolution for a `WTCRemoteTuxDom` is `ON_DEMAND` then the TDomain session is always available even though it does not exist. When a connection policy resolution for `WTCRemoteTuxDom` is `INCOMING_ONLY` or `ON_STARTUP`, then the TDomain session becomes available only when the connection is made and the TDomain session exists.

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

Example 4-3 Dynamically Suspend and Resume Services

```
java weblogic.WLST
wls:/offline> connect('weblogic','weblogic','t3://localhost:7001')
wls:/mydomain/serverConfig> serverRuntime()
wls:/mydomain/serverRuntime> cd('WTCRuntime/WTCService')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> ls()
-r-- Name WTCService
-r-- ServiceStatus weblogic.wtc.gwt.DServiceInfo[weblogic.wtc.gwt.DServiceInfo@1947a96]
-r-- Type WTCRuntime
-r-x getServiceStatus Integer :
                 String(java.lang.String),String(java.lang.String),String(java.lang.String)
-r-x getServiceStatus Integer : String(localAccessPoint),String(svcName)
-r-x getServiceStatus Integer : String(localAccessPoint),String(svcName),Boolean(isImport)
-r-x getServiceStatus Integer : String(svcName)
-r-x getServiceStatus Integer : String(svcName),Boolean(isImport)
-r-x listConnectionsConfigured weblogic.wtc.gwt.DSessConnInfo[] :
-r-x resumeService Void : String(localAccessPoint),String(remoteAccessPointList),String(svcName)
-r-x resumeService Void : String(localAccessPoint),String(svcName)
-r-x resumeService Void : String(localAccessPoint),String(svcName),Boolean(isImport)
-r-x resumeService Void : String(svcName)
-r-x resumeService Void : String(svcName),Boolean(isImport)
-r-x startConnection Void : String(LDomAccessPointId)
-r-x startConnection Void : String(LDomAccessPointId),String(RDomAccessPointId)
```

```
-r-x stopConnection Void : String(LDomAccessPointId)
-r-x stopConnection Void : String(LDomAccessPointId),String(RDomAccessPointId)
-r-x suspendService Void : String(localAccessPoint),String(remoteAccessPointList),String(svcName)
-r-x suspendService Void : String(localAccessPoint),String(svcName)
-r-x suspendService Void : String(localAccessPoint),String(svcName),Boolean(isImport)
-r-x suspendService Void : String(svcName)
-r-x suspendService Void : String(svcName),Boolean(isImport)
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> status=cmo.getServiceStatus()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print status[0].getServiceName()
TOUPPER
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print
         weblogic.wtc.gwt.WTCServiceStatus.svcTypeToString(status[0].getServiceType())
IMPORT
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print
         weblogic.wtc.gwt.WTCServiceStatus.statusToString(status[0].getStatus())
AVAILABLE
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> cmo.suspendService('TOUPPER')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> status=cmo.getServiceStatus()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print
         weblogic.wtc.gwt.WTCServiceStatus.statusToString(status[0].getStatus())
SUSPENDED
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> cmo.resumeService('TOUPPER')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> status=cmo.getServiceStatus()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print
         weblogic.wtc.gwt.WTCServiceStatus.statusToString(status[0].getStatus())
AVAILABLE
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print status[0].getServiceName()
TOUPPER
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> cmo.suspendService('TDOM1','TOUPPER')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> status=cmo.getServiceStatus()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print status[0].getServiceName()
TOUPPER
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print
         weblogic.wtc.gwt.WTCServiceStatus.statusToString(status[0].getStatus())
SUSPENDED
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> cmo.resumeService('TDOM1','TOUPPER')
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> status=cmo.getServiceStatus()
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print status[0].getServiceName()
TOUPPER
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> print
         weblogic.wtc.gwt.WTCServiceStatus.statusToString(status[0].getStatus())
AVAILABLE
wls:/mydomain/serverRuntime/WTCRuntime/WTCService>
wls:/mydomain/serverRuntime/WTCRuntime/WTCService> disconnect()
wls:/offline> exit()
```

5

Administration of CORBA Applications

This chapter provides information on how to administer and configure the Oracle WebLogic Tuxedo Connector to support Tuxedo CORBA clients and services.

For additional information, see Tuxedo CORBA.

This chapter includes the following sections:

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

How to Configure Oracle WebLogic Tuxedo Connector for CORBA Service Applications

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

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 `UBBCONFIG` file of the remote Tuxedo domain where the object is deployed. The maximum length of this unique identifier for CORBA domains is 15 characters including the `//`.
    - Set `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.

See *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server*.

See Configuring Oracle WebLogic Tuxedo Connector for Your Applications.

- Example WTC Server and Tuxedo UBB Files

Example WTC Server and Tuxedo UBB Files

The following WTC server (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.

Example 5-1 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>
          <remote-name>//simpapp</remote-name>
     </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 server.

Example 5-2 Example Tuxedo UBB File for a CORBA Server Application

```
*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 Oracle WebLogic Tuxedo Connector for Inbound RMI-IIOP

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

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

Configuring Your WTC Server for Inbound RMI-IIOP

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.

See Configuring Oracle WebLogic Tuxedo Connector for Your Applications.

Administering the Tuxedo Application Environment

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

1. Set the TOBJADDR for your environment, for 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, for 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:

```
*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 server.

4. Load your modified domain configuration file using `dmloadcf`.

See Tuxedo Administration Topics.

- Guidelines About Using Your Server Name as an Object Reference

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:tgiop:examplesServer/NameService`

How to Configure Oracle WebLogic Tuxedo Connector for Outbound RMI-IIOP

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 server:

1. Configure a local Tuxedo access point.
   - 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.
2. Configure imported services.
   - Set `Resource Name` to `//domain_id` where `domain_id` is `DOMAINID` specified in the Tuxedo `UBBCONFIG` file of the remote Tuxedo domain where the object is deployed. The maximum length of this unique identifier for CORBA domains is 15 characters including the `//`.
   - Set `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.

See *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server* for information on how to develop applications that use RMI-IIOP to call a Tuxedo service using a WebLogic Server EJB.

See Configuring Oracle WebLogic Tuxedo Connector for Your Applications.

Example Outbound RMI-IIOP Configuration

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

Example 5-3 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>myLoclTuxDom</name>
           <scurity>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>
.
.
.
```

6

How to Manage Oracle WebLogic Tuxedo Connector in a Clustered Environment

This chapter provides information on how to administer and configure the Oracle WebLogic Tuxedo Connector for use in a clustered environment.

For additional information, see *Administering Clusters for Oracle WebLogic Server*.

This chapter includes the following sections:

Oracle WebLogic Tuxedo Connector Guidelines for Clustered Environments

Use the following guidelines when deploying Oracle 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 Oracle WebLogic Tuxedo Connector that includes an Imported Services page that defines any imported services required. If one server in the cluster does not have a Oracle WebLogic Tuxedo Connector deployed, the Jakarta Enterprise 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.
- Oracle WebLogic Tuxedo Connector does not support inbound TGIOP in clustered environments.
- How to Configure for Clustered Nodes

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

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

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

See Communications in a Cluster in *Administering Clusters for Oracle WebLogic Server*. Oracle WebLogic Tuxedo Connector also provides domain-level failover and failback capabilities. See Configuring Failover and Failback.

- Example Clustered Oracle WebLogic Tuxedo Connector Configuration

Example Clustered Oracle WebLogic Tuxedo Connector Configuration

The following configuration provides an example of Oracle 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 server that contains the same service (TOUPPER) in as an imported service.

Example 6-1 Example Clustered Oracle 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-provider>
<sec:cert-path-builder>WebLogicCertPathProvider</sec:cert-path-builder>
     <sec:user-lockout-manager></sec:user-lockout-manager>
     <sec:security-dd-model>Advanced</sec:security-dd-model>
<sec:combined-role-mapping-enabled>false</sec:combined-role-mapping-enabled>
     <sec:name>myrealm</sec:name>
   </realm>
   <default-realm>myrealm</default-realm>
<credential-encrypted>{3DES}O0Qw7QBG3+cmemXbtKhHPJL2QLw7tqSYkoWqBtU17W+IoPebpoNai/T3SdtxBOwVHOJJPi
        /sA8JMJ9MAM4i3KqVgd26A311z</credential-encrypted>
   <web-app-files-case-insensitive>os</web-app-files-case-insensitive>
<compatibility-connection-filters-enabled>true</compatibility-connection-filters-enabled>
   <node-manager-username>weblogic</node-manager-username>
<node-manager-password-encrypted>{3DES}37KMzVTzxZ9VFxCFSVGWzA==</node-manager-password-encrypted>
     <enforce-strict-url-pattern>false</enforce-strict-url-pattern>
   </security-configuration>
   <security>
     <realm>wl_default_realm</realm>
```

```
    <password-policy>wl_default_password_policy</password-policy>
  </security>
  <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>
        <block-time>30000</block-time>
        <nw-addr>//mymachine:20401</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>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>EJBLSleep</resource-name>
        <local-access-point>WDOM1</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>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>
```

```
        <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>
  </wtc-server>
  <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>
<filter-dispatched-requests-enabled>true</filter-dispatched-requests-enabled>
      <rtexprvalue-jsp-param-name>true</rtexprvalue-jsp-param-name>
<jsp-compiler-backwards-compatible>true</jsp-compiler-backwards-compatible>
    </web-app-container>
    <admin-server-name>wtcAServer</admin-server-name>
</domain>
```

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
- Fail Over

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 Oracle 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
```

See Tuxedo Load Balancing.

Fail Over

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
```

See Specifying Domains Failover and Failback on Tuxedo.

7

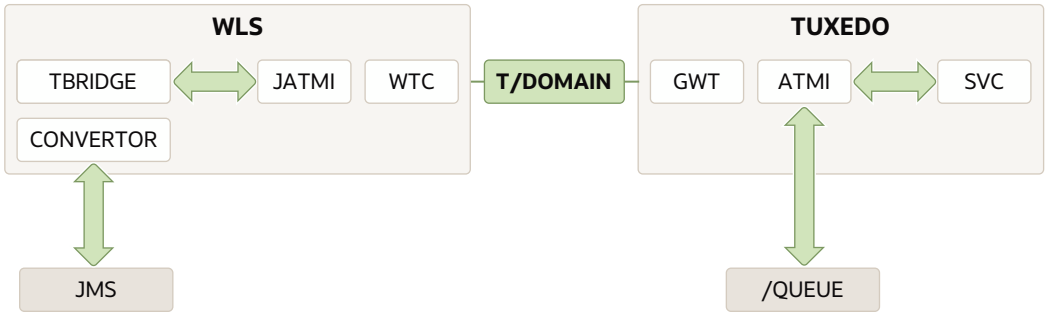
How to Configure the Oracle Tuxedo Queuing Bridge

This chapter provides information on the Tuxedo queuing bridge functionality and configuration.
This chapter includes the following sections:

Overview of the Tuxedo Queuing Bridge

The Tuxedo queuing bridge is a part of the Oracle WebLogic Tuxedo Connector that provides a bi-directional JMS interface for your WebLogic Server applications to 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.

Figure 7-1 Interaction between WebLogic Server and Tuxedo with queuing bridge



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 server.
- The Tuxedo queuing bridge uses Jakarta Messaging (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.

See the following sections:

How Tuxedo Queuing Bridge connects JMS with Tuxedo

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

> **Note:**
>
> Messages remain on the JMS queue until they have been acknowledged.

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.
   a. BytesMessage, TextMessage, XML are converted respectively to TypedCArray, TypedString, and TypedFML32. XML/FML translation is performed according to the `TranslateFML` attribute.
   b. Translation errors are sent to the `wlsServerErrorDestination` queue and the message is acknowledged in the JMS session.
   c. 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 TDomain gateway.
   a. Messages with a redirect set to `JmsQ2TuxQ` use JATMI `tpenqueue` to deliver the message to a Tuxedo queue.
   b. 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

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

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

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. See, FML32 Considerations in *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server*.

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 server. These attributes contain the necessary information to establish a connection to Tuxedo.

A complete Tuxedo queuing bridge configuration requires the following:

- Configure one queuing bridge
- Configure one or more redirections

See the following sections:

- Dynamically Adding/Modifying Tuxedo Queuing Bridge
- Tuxedo Queuing Bridge Instantiate
- Starting the Tuxedo Queuing Bridge
- Error Logging

Dynamically Adding/Modifying Tuxedo Queuing Bridge

Typically, after a complete Tuxedo queuing bridge configuration exists and WTC is deployed, the Tuxedo queuing bridge is activated. After the Tuxedo queuing bridge is active, no additions or modifications to the configuration can occur without shutting the WTC queuing bridge down.

If WTC is activated and the Tuxedo queuing bridge is deactivated, the following additions and modifications are possible.

- Add WTC queuing bridge if there is no WTC queuing bridge configuration
- Modify WTC queuing bridge
- Delete WTC queuing bridge
- Add WTC redirections
- Modify WTC redirections
- Delete WTC redirections

After completing the WTC queuing bridge configuration changes, you must activate the changes.

> **Note:**
>
> Dynamically adding/modifying Tuxedo queuing bridge is only possible if the queuing bridge is not activated. WTC can be activated, but the queuing bridge must not be activated. To shut down the WTC queuing bridge, you should deactivate the WTC server.

Tuxedo Queuing Bridge Instantiate

If, and only if, a complete WTC queuing bridgequeuing bridge configuration is available at the time of activation will WTC start the Tuxedo queuing bridge.

If the WTC queuing bridge configuration is incomplete, then no Tuxedo queuing bridge instance is available.

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 server are configured and the WTC server 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

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

Tuxedo Queuing Bridge Connectivity

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 server 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.

> **Note:**
>
> JMS message types: MapMessage, ObjectMessage, StreamMessage are not valid in Oracle 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.

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.

```
<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>
     <translate-fml>NO</translate-fml>
</wtc-tbridge-redirect>
```

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

- The `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.

Table 7-1 provides information on JmsQtoTuxQ message mapping.

Table 7-1 JmsQ to TuxQ Message Mapping

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

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.

```
<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>
     <target-name>weblogic.jms.Tux2JmsQueue</target-name>
     <translate-fml>NO</translate-fml>
</wtc-tbridge-redirect>
```

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

- The `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.

Table 7-2 provides information on `TuxQ2JmsQ` message mapping:

Table 7-2 TuxQ2JmsQ Message Mapping

From: Oracle WebLogic Tuxedo Connector JATMI (Tuxedo)	To: JMS Message Type
TypedCArray	BytesMessage
TypedString (translateFML = NO)	TextMessage
TypedFML32 (translateFML = FLAT)	TextMessage

Table 7-2 (Cont.) TuxQ2JmsQ Message Mapping

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

Example JmsQ2TuxS Configuration

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:

- The `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`.

Table 7-3 provides information on the JMSQ2TuxX message mapping:

Table 7-3 JMSQ2TuxX Message Mapping

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

> **Note:**
>
> There may be scenarios where a reply from Tuxedo is returned and the `translateFML` parameter has no effect. Translation may occur automatically.

See Using FML with WebLogic Tuxedo Connector in *Developing Oracle WebLogic Tuxedo Connector Applications for Oracle WebLogic Server*.

Priority Mapping

Oracle 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.

```
JmstoTux- 0:1 | 1:12 | 2:23 | 3:34 | 4:45 | 5:56 | 6:67 | 7:78 | 8:89 | 9:100
TuxtoJms- 0:1-10|1:11-20|2: 21-30|3: 31-40|4: 41-50|5:51-60|6: 61-70|7:71-80|8:81-90|9:91-100
```

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.

See the following sections:

- WLS Error Destination
- Unsupported Message Types
- Tuxedo Error Queue
- Limitations

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.

8

Connecting WebLogic Integration and Tuxedo Applications

This chapter describes how the Oracle WebLogic Tuxedo Connector Tuxedo queuing bridge provides the necessary infrastructure for WebLogic Integration users to integrate Tuxedo applications into their business workflows.

For additional information, see BEA WebLogic Integration.

This chapter includes the following sections:

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 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.
- Process TypedFML32 response buffers using defined Business Operations.

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.

9

Troubleshooting The WebLogic Tuxedo Connector

This chapter provides WebLogic Tuxedo Connector troubleshooting information. This chapter includes the following sections:

- 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 (Deprecated)
- Enable Debug Mode
- Enable a User Data Dump

Set Trace Levels (Deprecated)

Because `TraceLevel` is no longer supported, use system debugging. By default all the debug tracing is off. See System Level Debug Settings for information on how to develop applications that use RMI-IIOP to call a Tuxedo service using a WebLogic Server EJB.

See How to Set Oracle WebLogic Tuxedo Connector Properties.

Enable Debug Mode

Use the following Remote Console procedure to specify that trace information is written to the log file.

1. In the **Edit Tree**, go to **Environment**, then **Servers**: *myServer*.
2. Go to the **Debug**: **All** page, select the desired **DebugWTC** options, and then click **Save**.
3. Go to **Logging**: **General**, and click **Show Advanced Properties**.
4. Set the **Stdout Severity Level** to **Info**.
5. Click **Save**.

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 `simpserv` example, I get the following error:

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

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

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 and later, WebLogic Tuxedo Connector sessions are managed using a WTC server.

- A WebLogic Tuxedo Connector session is started when a configured WTC server is assigned to a selected server instance.
- A WebLogic Tuxedo Connector session is ended by removing a WTC server from the WebLogic server instance or when you shut down the WebLogic server instance.

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 server and the WTC server is assigned to a selected server

How do I Assign a WTC Server to a Server Instance?

The WebLogic Remote Console displays an exception when I try to assign my WTC server to a server instance. What should I do?

Make sure you have a valid WTC server configured. Each WTC server must have one or more local Tuxedo access points configured before it can be assigned to a server instance. 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 Oracle 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 Section 3.5, User Authentication.
- 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 Oracle WebLogic Tuxedo Connector Properties.
- Check the WebLogic Tuxedo Connector configuration against the Tuxedo remote domain. The remote domain must match the name of a remote domain configured in Oracle 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?

See *Upgrading Oracle WebLogic Server*.

Index