BEA WebLogic Server®

WLEC to WebLogic Tuxedo Connector Migration Guide

Version 9.1
Document Revised: December 16, 2005

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Overview of WLEC to WebLogic Tuxedo Connector Migration

How to Modify WLEC Applications for WebLogic Tuxedo Connector

How to Modify the Tuxedo CORBA Simpapp Example

CHAPTER 1

Overview of WLEC to WebLogic Tuxedo Connector Migration

The following sections provide an overview of the requirements and procedures to migrate WLEC applications to the WebLogic Tuxedo Connector:

Guide to this Document

This document introduces the BEA WebLogic Tuxedo Connector™ application development environment. This document provides information on how to migrate WLEC applications to use the WebLogic Tuxedo Connector to interoperate between WebLogic Server and Tuxedo.

The document is organized as follows:

- Chapter 1, "Overview of WLEC to WebLogic Tuxedo Connector Migration," provides information on migration prerequisites, functionality, and key administrative and programming differences between WLEC to WebLogic Tuxedo Connector.
- Chapter 2, "How to Modify WLEC Applications for WebLogic Tuxedo Connector," provides information on how to modify your Tuxedo environment, WebLogic Server environment, and WLEC applications for use with the WebLogic Tuxedo Connector.

- Chapter 3, "How to Modify the Tuxedo CORBA Simpapp Example," provides an example of how to convert a WLEC application to use WebLogic Tuxedo Connector.

Overview

WLEC is a deprecated service in WebLogic Server 8.1. WLEC users should begin plans to migrate applications using WLEC to the WebLogic Tuxedo Connector.

WebLogic Tuxedo Connector provides bi- directional interoperability between WebLogic Server applications and Tuxedo services. The connector allows WebLogic Server clients to invoke Tuxedo services and Tuxedo clients to invoke WebLogic Server Enterprise Java Beans (EJBs) in response to a service request.

WLEC to WebLogic Tuxedo Connector migration requires minor application modification:

- WLEC applications require modification of the portions of application code that use or call environmental objects.
- Existing CORBA C++ server objects do not require server application changes.

Prerequisites

Before you can start migrating your WLEC applications to WebLogic Tuxedo Connector, make sure that you have installed:

- Tuxedo
  - If necessary, migrate your Tuxedo applications to Tuxedo 8.1 or later. For more information, see Upgrading the BEA Tuxedo System to Release 8.1 located at http://e-docs.bea.com/tuxedo/tux81/install/insup.htm or Upgrading the BEA Tuxedo System to Release 9.0 located at http://e-docs.bea.com/tuxedo/tux90/install/insup.htm.
- WebLogic Server
  - If necessary, migrate your WebLogic Server installation to WebLogic Server 9.0. For more information, see WebLogic Platform Upgrade Guide.

Comparing WebLogic Tuxedo Connector and WLEC Functionality

The following table compares the supported functionality in WebLogic Tuxedo Connector and WLEC:

Table 1-1 WebLogic Tuxedo Connector and WLEC Functionality

Feature	WTC	WLEC
Outbound ATMI interoperability from WLS	Yes	No
Inbound ATMI interoperability from Tuxedo	Yes	No
Outbound CORBA interoperability	Yes	Yes
Inbound CORBA interoperability	Yes	No
Supports Tuxedo Buffers	Yes	No
Bi-directional security context propagation	Yes	No
Bi-directional transaction propagation	Yes	No
Bi-directional bridge between JMS and /Q or Tuxedo services	Yes	No
Conversations	Yes	No
VIEWS	Yes	No

Key Differences Between WLEC and WebLogic Tuxedo Connector

The following sections provide information on key administration, configuration, and programming differences between WebLogic Tuxedo Connector and WLEC.

Table 1-2 WLEC and WebLogic Tuxedo Connector Key Differences

Description	WebLogic Tuxedo Connector	WLEC
Connectivity	Uses the Tuxedo /T Domain gateway. The gateway creates a single network link between a WebLogic Server instance and a Tuxedo domain for all method invocations.	Uses a pool of connections and each invocation is sent over a connection obtained from this pool.

Failover Management	Uses Tuxedo domains.	Uses a failover list.
Object Routing	CORBA calls from WebLogic Server applications are propagated to the Tuxedo CORBA environment using the TGIOP/TDOMAINS protocol.	CORBA calls from WebLogic Server applications are propagated over IIOP connection pools using the CORBA API.

CHAPTER 2

How to Modify WLEC Applications for WebLogic Tuxedo Connector

The following sections provide information on the steps required to convert your WLEC applications for use with WebLogic Tuxedo Connector:

- How to Modify Your Tuxedo Environment
- How to Modify your WebLogic Server Environment
- How to Modify WLEC Applications

How to Modify Your Tuxedo Environment

Tuxedo users need to make the following environment changes:

- Create a Tuxedo dmconfig File
- Modify the Tuxedo tuxconfig File

Create a Tuxedo dmconfig File

A new `dmconfig` file must be created to provide connectivity between your Tuxedo and WebLogic Server applications. For more information on how to create Tuxedo domains, see Planning and Configuring CORBA Domains at http://e-docs.bea.com/tuxedo/tux90/add/adcorb.htm.

Modify the Tuxedo tuxconfig File

You will need to modify the `tuxconfig` file so your application will use the Tuxedo /T Domain gateway. Add Tuxedo the domain servers to the `*SERVERS` section of you UBB file.

Example:

```
DMADM SRVGRP=SYS_GRP SRVID=7
GWADM SRVGRP=SYS_GRP SRVID=8
GWTDOMAIN SRVGRP=SYS_GRP SRVID=9
```

Weblogic Tuxedo Connector does not use ISL. If you no longer have other applications that require ISL, you can remove the `ISL` from the `*SERVERS` section.

Example: Comment out the `ISL` section.

```
#      ISL
#           SRVGRP   = SYS_GRP
#           SRVID    = 5
#            CLOPT    = "-A -- -n //lchp15:2468 -d /dev/tcp"
```

How to Modify your WebLogic Server Environment

This section provides information on how to modify your WebLogic Server Environment.

- How to Configure WebLogic Tuxedo Connector
- How to Update the ejb-jar.xml File

How to Configure WebLogic Tuxedo Connector

Note: For more information on how to configure WebLogic Tuxedo Connector, see Configuring WebLogic Tuxedo Connector for Your Applications at http://e-docs.bea.com/wls/docs91/wtc_admin/Install.html.

This section provides basic information on how to create a WTC Service for a migrated WLEC application using the WebLogic Server console. A WTC Service represents configuration information that WebLogic Server uses to create a connection to a Tuxedo application. Typical WTC Service configurations for migrated WLEC applications consist of a local Tuxedo access point, a remote Tuxedo access point, and an imported service.

Use the following steps to create a configuration to administer your application:

1. Create a WTC Service
2. Create a Local Tuxedo Access Point
3. Create a Remote Tuxedo Access Point
4. Create an Imported Service

Create a WTC Service

Use the following steps to create and configure a WTC service using the WebLogic Server Administration Console:

1. In the Administration Console, expand Interoperability and select WTC Servers in the navigation tree.
2. On the WTC Servers page, click New.
3. On the Create a New WTC Server page, enter the name of your WTC Service in the Name field. Example: `mySimpapp`
4. Click OK.
5. Your new WTC Service appears in the WTC Servers list.

Create a Local Tuxedo Access Point

Note: When configuring the Network Address for a local access point, the port number used should be different from any port numbers assigned to other processes. Example: Setting the Network Address to `//mymachine:7001` is not valid if the WebLogic Server listening port is assigned to `//mymachine:7001`.

Use the following steps to configure a local Tuxedo access point:

1. In the Administration Console, expand Interoperability and select WTC Servers.
2. On the WTC Servers page, click the name of a WTC Service, such as mySimpapp, to access the settings page.
3. Click the Local APs tab.
4. Enter the following values for the following fields on the WTC Local Access Points page:

In Access Point, enter a name that uniquely identifies this local Tuxedo access point within a WTC Service configuration. This allows you to create Local Tuxedo Access Point configurations that have the same Access Point ID.

In Access Point Id, enter the connection name used when establishing a session connection to remote Tuxedo access points. The Access Point Id must match the corresponding `DOMAINID` in the `*DM_REMOTE_DOMAINS` section of your Tuxedo `DMCONFIG` file.

In Network Address, enter the network address and port for this local Tuxedo access point. For example: `//123.123.123.123:5678`.

5. Click OK.
6. If you are connecting to a Tuxedo 6.5 domain, do the following:
   a. Click the Connections tab.
   b. Set the Interoperate field to `Yes`.
   c. Click Save.

Create a Remote Tuxedo Access Point

Use the following steps to configure a remote Tuxedo access point:

1. In the Administration Console, expand Interoperability and select WTC Servers.
2. On the WTC Servers page, click the name of a WTC Service, such as mySimpapp.
3. Click the Remote APs tab.
4. Enter the following values for the following fields on the WTC Remote Access Points page:

   In Access Point, enter a name that uniquely identifies this remote Tuxedo access point within a WTC Service configuration. This allows you to create Remote Tuxedo Access Point configurations that have the same Access Point ID.

   In Access Point Id, enter the connection name used to identify a remote Tuxedo access point when establishing a connection to a local Tuxedo access point. The Access Point Id of a remote Tuxedo access point must match the corresponding `DOMAINID` in the `*DM_LOCAL_DOMAINS` section of your Tuxedo `DMCONFIG` file.

   In Local Access Point, enter the name of the local access point for this remote domain.

   In Network Address, enter the network address and port for this remote domain. For example: `//123.123.123.123:1234`

5. Click OK.

Create an Imported Service

Use the following steps to configure an imported service:

1. In the Administration Console, expand Interoperability and select WTC Servers.
2. On the WTC Servers page, click the name of a WTC Service, such as mySimpapp.
3. Click the Imported tab.
4. Enter the following values for the following fields on the WTC Imported Services page:

   In Resource Name, enter a name to identify this imported service configuration. This name allows you create unique Imported Services configurations that have the same Remote Name within a WTC Service.

   Set Local Access Point to the name of the Local Tuxedo Access Point that uses the service.

   In Remote Access Point List, enter a list of Remote Access Point names that offer this imported service.

   In Remote Name, enter *"//domain_id"* where *domain_id* is `DOMAINID` specified in the Tuxedo `UBBCONFIG` file. The maximum length of this unique identifier for CORBA domains is 15 characters and includes the //.

   Example: `//simpappff`

5. Click OK.

How to Update the ejb-jar.xml File

WebLogic Tuxedo Connector uses the Domain gateway to connect WebLogic and Tuxedo applications. IIOP connection pool are not used and the descriptors can be removed from the `ejb-jar.xml` file. The following is an example of code removed from the `wlec/ejb/simpapp` example:

Listing 2-1 IIOP Connection Pool Descriptors for the wlec/ejb/simpapp Example

```
.
.
.
<env-entry>
```

```
        <env-entry-name>IIOPPoolName</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>simplepool</env-entry-value>
      </env-entry>
.
.
.
```

How to Modify WLEC Applications

The following sections provide information on how to modify WLEC applications to interoperate with WebLogic Server and Tuxedo CORBA objects using WebLogic Tuxedo Connector.

- How to Modify WLEC EJBs to Reference CORBA Objects Used by WebLogic Tuxedo Connector
- Transaction Issues

How to Modify WLEC EJBs to Reference CORBA Objects Used by WebLogic Tuxedo Connector

Use the following steps to modify your EJB to use WebLogic Tuxedo Connector to invoke on CORBA objects deployed in Tuxedo:

- Initialize the WTC ORB
- Use the ORB to get the FactoryFinder Object

Initialize the WTC ORB

WLEC uses the `weblogic.jndi.WLInitialContextFactory` to return a context used by the `Tobj_Bootstrap` object.

```
Properties p = new Properties();
p.put(Context.INITIAL_CONTEXT_FACTORY,
     "weblogic.jndi.WLInitialContextFactory");
InitialContext ic = new InitialContext(p);
rootCtx = (Context)ic.lookup("java:comp/env");
```

Replace the WLEC context reference and instantiate the WTC ORB in your Bean. Example:

```
// Initialize the ORB.
String args[] = null;
Properties Prop;
Prop = new Properties();
Prop.put("org.omg.CORBA.ORBClass",
"weblogic.wtc.corba.ORB");

orb = (ORB)new InitialContext().lookup("java:comp/ORB");
```

Use the ORB to get the FactoryFinder Object

Each WLEC connection pool has a `Tobj_Bootstrap` FactoryFinder object used to access the Tuxedo domain. Example:

```
Tobj_Bootstrap myBootstrap =
Tobj_BootstrapFactory.getClientContext("myPool");
org.omg.CORBA.Object myFFObject =
    myBootstrap.resolve_initial_references("FactoryFinder");
```

Remove references to the `Tobj_Bootstrap` Factory Finder object. Use the following method to obtain the FactoryFinder object using the ORB:

```
// String to Object.
org.omg.CORBA.Object fact_finder_oref =
orb.string_to_object("corbaloc:tgiop:simpapp/FactoryFinder");

// Narrow the factory finder.
FactoryFinder fact_finder_ref =
FactoryFinderHelper.narrow(fact_finder_oref);

// Use the factory finder to find the simple factory.
org.omg.CORBA.Object simple_fact_oref =
fact_finder_ref.find_one_factory_by_id(SimpleFactoryHelper.id());
```

Transaction Issues

Note: For more information how to implement JTA transactions, see Programming WebLogic JTA at http://e-docs.bea.com/wls/docs91/jta/index.html.

The following section provides information about how to modify WLEC applications that use transactions.

- WLEC applications using JTA transactions require no changes.
- WLEC applications using CosTransactions need to convert to JTA. If the WLEC client is running within a transaction and needs to invoke a new CosTransaction, the new

transaction is implemented in a new transaction context. To implement the same behavior in JTA, do the following:

- Suspend the original transaction
- Start a new transaction
- Resume the original transaction after the new transaction has been completed.

How to Manage Security Issues Migrating from WLEC to WTC

The following table provides some mapping guidelines for security issues between WLEC and WTC as well as their relationship to Tuxedo.

Table 2-1 Security Mapping Guidelines Migrating from WLEC to WTC

WLEC Security Items	Map to in WTC/WLS	Tuxedo
user name	Access the WTC Servers page and click the Local APs tab. Use the user name in the Access Point ID field.	`DOMAINID` in the `DM_REMOTE_DOMAINS` section of the `DMCONFIG` file
user password	Password pair in the password (rather than one password, you must define one for the local access point and one for the remote access point to form mutual authentication.) You can use `"weblogic.wtc.gwt.genpasswd"` utility to generate the encrypted password pair and then cut and paste to the Console WTC Password page.	Use dmadmin to add the passsword pair to each defined TDomain session.
role	Not supported. WTC depends on impersonating user and uses the impersonated user role defined in Tuxedo.	
application password	The password in the WTC Resources page. Use `"weblogic.wtc.gwt.genpasswd"` utility to generate the encrypted application password.	No special configuration needs.

Table 2-1 Security Mapping Guidelines Migrating from WLEC to WTC

WLEC Security Items	Map to in WTC/WLS	Tuxedo
min encryption level	1. Access the WTC Servers page and click the name of a WTC Service. 2. Click Remote APs tab. 3. Click the Security tab and select the Min Encryption Level required.	MINENCRYPTBITS in the the DM_TDOMAIN section of the DMCONFIG file.
max encryption level	1. Access the WTC Servers page and click the name of a WTC Service. 2. Click Remote APs tab. 3. Click the Security tab and select the Max Encryption Level required.	MAXENCRYPTBITS in the the DM_TDOMAIN section of the DMCONFIG file.
certificate auth	Not supported.	
security context propagation	1. Access the WTC Servers page and click the name of a WTC Service. 2. Click Remote APs tab. 3. Click the Security tab and select Global for the Credential Policy field to propagate user credential to Tuxedo.	ACL="GLOBAL" in the DM_REMOTE_DOMAINS section of the DMCONFIG file.

The following considerations may assist you in understanding how your current WLEC security can map to WTC and Tuxedo security.

- The WLEC user name in the certificate can be used as your Access Point ID in the WTC Local APs page.
- You must configure Access Point ID in the WTC Remote APs page using the remote Tuxedo domain gateway's DOMAINID. (This DOMAINID should be one of the entries in the DM_LOCAL_DOMAINS section in the DMCONFIG file.)
- You must configure the user in both Tuxedo and WTC if you want security context propagation.
- If you do not want security context propagation, do not configure credential-policy. By default, credential-policy is set to "LOCAL" which means no propagation. Also, do not configure ACL_POLICY in Tuxedo. By default ACL_POLICY is set to "LOCAL" which

means do not accept any remote security context received. In this case, if the Tuxedo security level is higher than USER_AUTH, then the DOMAINID for WTC which is configured in the DM_REMOTE_DOMAINS section of the DMCONFIG file is used.

- From the Security tab of the WTC Local APs page, select Domain Password for the Security field. You need to configure 'SECURITY="DM_PW"' in one of the entries in DM_LOCAL_DOMAINS section of the DMCONFIG file for Tuxedo. In this case, password must be configured for for both WTC and the TDomain Gateway and application password is not required.
- If you do not want to set Security to Domain Password, you can set it to Application Password. In this case, you do not have to configure password pair in the WTC Passwords page, but you need to configure App Password and App Password IV fields in the WTC Resources page.

CHAPTER 3

How to Modify the Tuxedo CORBA Simpapp Example

The following section provides an example of how to convert a WLEC application to use WebLogic Tuxedo Connector. This example provides information on the steps required to convert the WebLogic Server 6.1 `examples\wlec\ejb\simpapp` example to work using the WebLogic Tuxedo Connector. A complete migrated FactoryFinder example is available from the BEA dev2dev code library at http://dev2dev.bea.com/code/index.jsp. Review the Prerequisites located at http://e-docs.bea.com/wls/docs91/wlec_migration/Intro.html#Requirements before proceeding.

How to Modify the Tuxedo Environment

This section provides information on how to modify the Tuxedo configuration files to use with WebLogic Tuxedo Connector.

- Create a Domain Configuration
- Test the Tuxedo Environment

Run the Tuxedo CORBA Simpapp Example

You should run the Tuxedo CORBA `simpapp` example to verify your Tuxedo environment and prepare to run the WLEC simpapp application.

Use the following steps to run the Tuxedo example located at `$TUXDIR/samples/corba/simpapp`:

1. Create a working copy of the Tuxedo CORBA `simpapp` example. Copy the Tuxedo CORBA `simpapp` example from your Tuxedo installation and place it in your working `simpapp` directory.
2. Change directories to your working `simpapp` directory.
3. Build and run the example.
   a. Set your Tuxedo environment. Windows users `set %TUXDIR%` in your shell environment. Unix users need to set the Tuxedo environment by running `$TUXDIR/tux.env`.
   b. Make sure the C++ compiler is in your `PATH`.
   c. Set the JAVA_HOME environment variable to the location of your Tuxedo Java JDK.
   d. Set the environment by running the `runme` script. This will create the client stubbs that provide the programming interface for CORBA object operations. A `results` directory is created in your working directory that contains the files used to configure the Tuxedo environment.
   e. Run the Java client.

```
java -DTOBJADDR=%TOBJADDR% -classpath %CLASSPATH% SimpleClient
```

   f. Shutdown the Tuxedo server.

```
tmshutdown -y
```

Modify the UBB Configuration File

In your working Tuxedo `simpapp` directory, use the following steps to modify your `UBB` configuration:

1. Rename the `results/ubb` file in your working directory as `results/ubbdomain`.

2. Edit the `ubbdomain` file using a text editor, such as vi or WordPad.
3. Add Tuxedo gateway servers to the `*SERVERS` section.

   Example: Add the following servers.

```
DMADM SRVGRP=SYS_GRP SRVID=7
GWADM SRVGRP=SYS_GRP SRVID=8
GWTDOMAIN SRVGRP=SYS_GRP SRVID=9
```

4. Save the `ubbdomain` file.

The following code is an example of a modified `ubbdomain` file. Changed sections are marked in **bold**.

Listing 3-1 Modified UBB File

```
*RESOURCES
     IPCKEY    55432
     DOMAINID  simpapp
     MASTER    SITE1
     MODEL     SHM
     LDBAL     N
*MACHINES
     "balto"
     LMID         = SITE1
     APPDIR       = "/tux_apps/corba/simpapp"
     TUXCONFIG    = "/tux_apps/corba/simpapp/results/tuxconfig"
     TUXDIR       = "/my_machine/tux/tuxedo8.1"
     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"
simple_server
     SRVGRP  = APP_GRP
     SRVID   = 1
     RESTART = N

# The ISL handler is not needed for WTC.
# If you do not need it for other WLEC applications,
# it can be removed.
ISL
     SRVGRP  = SYS_GRP
     SRVID   = 5
     CLOPT   = "-A -- -n //mymachine:2468 -d /dev/tcp"
DMADM
     SRVGRP= SYS_GRP
     SRVID= 7
GWADM
     SRVGRP= SYS_GRP
     SRVID= 8
GWTDOMAIN
     SRVGRP= SYS_GRP
```

```
     SRVID= 9
*SERVICES
```

Create a Domain Configuration

In your working Tuxedo `simpapp` directory, use the following steps to create a domain configuration:

1. Create a domain configuration file using a text editor, such as vi or NotePad. The simplest method is to cut and paste the `dmconfig` code example into your editor.
2. Replace all <bracketed> items with information for your environment.

Listing 3-2 dmconfig File

```
*DM_RESOURCES
VERSION=U22
*DM_LOCAL_DOMAINS
TUXDOM      GWGRP=SYS_GRP
            TYPE=TDOMAIN
            DOMAINID="TUXDOM"
            BLOCKTIME=20
            MAXDATALEN=56
            MAXRDOM=89
            DMTLOGDEV="<Path to domain TLOG device>"
            DMTLOGNAME="DMTLOG_TUXDOM"
*DM_REMOTE_DOMAINS
     examples TYPE=TDOMAIN DOMAINID="examples"
*DM_TDOMAIN
     TUXDOM NWADDR="<network address of Tuxedo domain>"
     examples NWADDR="<network address of WTC domain>"
*DM_REMOTE_SERVICES
```

3. Save the file as `dmconfig` in your working `simpapp/results` directory.

Test the Tuxedo Environment

Use the following steps to validate your Tuxedo configuration:

1. In a new shell, change directories to your working `simpapp/results` directory.
2. Set the environment using the `setenv` script for your platform.
3. Load the `ubbdomain` file:

```
tmloadcf -y ubbdomain
```

4. .Load the `dmconfig` file:

```
set
BDMCONFIG=<path_to_your_working_simpapp_example>/simpapp/results/bdm
config
dmloadcf -y dmconfig
```

5. Boot the Tuxedo domain

```
tmboot -y
```

6. Verify the Tuxedo environment.

```
java -DTOBJADDR=%TOBJADDR% -classpath %CLASSPATH% SimpleClient
```

7. Shutdown the Tuxedo server.

```
tmshutdown -y
```

Modify the ejb-jar.xml File

Use a text editor such as Vi or Notepad to remove connection pool descriptors and update the trans-attribute. The following listing provides a code example on how to remove references to the IIOP connection pool descriptors in the WLEC `simpapp` example `ejb-jar.xml`.

- Remove the `env-entry` attribute.
- Set the `trans-attribute` in the `container-transaction` to `Supports`. As the example does not have a transaction, the `container-transaction` can not be `Required`.

Listing 3-3 Example XML Configuration File for a CORBA Server Application

```
.
.
```

```
.
<ejb-jar>
<enterprise-beans>
<session>
     <ejb-name>ejb</ejb-name>
     <home>examples.wlec.ejb.simpapp.ConverterHome</home>
     <remote>examples.wlec.ejb.simpapp.Converter</remote>
<ejb-class>examples.wlec.ejb.simpapp.ConverterBean</ejb-class>
     <session-type>Stateless</session-type>
     <transaction-type>Container</transaction-type>
<!-- Remove or comment out the following statements
     <env-entry>
          <env-entry-name>IIOPPoolName</env-entry-name>
          <env-entry-type>java.lang.String</env-entry-type>
          <env-entry-value>simplepool</env-entry-value>
     </env-entry>
-->
</session>
</enterprise-beans>
<assembly-descriptor>
<container-transaction>
     <method>
     <ejb-name>ejb</ejb-name>
     <method-intf>Remote</method-intf>
     <method-name>*</method-name>
     </method>
     <trans-attribute>Supports</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

Update the build.xml File

A `build.xml` file is presented below to simplify compiling and deploying your migrated application in the Weblogic environment. Use the following example code to replace the contents of the `build.xml` file.

Listing 3-4 Updated build.xml file

```
<project name="wlec-ejb-simpapp" default="all" basedir=".">

<!-- set global properties for this build -->
<property environment="env"/>
<property file="../../../../examples.properties"/>
<property name="build.compiler" value="${compiler}"/>
<property name="source" value="."/>
<property name="build" value="${source}/build"/>
<property name="dist" value="${source}/dist"/>
<property name="ejb_classes" value="Converter.java, ConverterHome.java,
ConverterResult.java,
ProcessingErrorException.java, ConverterBean.java"/>
<property name="ejb_jar" value="wlec_simpapp_corba.jar"/>
<property name="client_classes" value="Converter.java, ConverterHome.java,
ConverterResult.java,
ProcessingErrorException.java, Client.java"/>

<target name="all" depends="clean, init, compile_idl, compile_ejb, jar_ejb,
appc, compile_client"/>

<target name="init">
<!-- Create the time stamp -->
     <tstamp/>
     <!-- Create the build directory structure used by compile
     and copy the deployment descriptors into it-->
     <mkdir dir="${build}"/>
     <mkdir dir="${build}/META-INF"/>
     <mkdir dir="${dist}"/>
     <copy todir="${build}/META-INF">
     <fileset dir="${source}">
     <include name="*.xml"/>
     <exclude name="build.xml"/>
     </fileset>
     </copy>
     </target>
```

```
<!-- Compile IDL stub classes into the build directory (jar preparation) -->
<target name="compile_idl">
     <exec executable="idlj" dir=".">
     <arg line="-td build -pkgPrefix Simple simple -pkgPrefix
     SimpleFactory simple simple.idl" />
     </exec>
     <javac srcdir="${build}" destdir="${build}"
     classpath="${CLASSPATH};${build}"/>
     <delete>
     <fileset dir="${build}">
     <include name="*.java"/>
     </fileset>
     </delete>
     </target>

<!-- Compile ejb classes into the build directory (jar preparation) -->
<target name="compile_ejb">
     <javac srcdir="${source}" destdir="${build}"
     includes="${ejb_classes}"
     classpath="${CLASSPATH};${build}"/>
     </target>

<!-- Make a standard ejb jar file, including XML deployment descriptors -->
<target name="jar_ejb" depends="compile_ejb">
     <jar jarfile="${dist}/std_${ejb_jar}"
     basedir="${build}">
     </jar>
     </target>

<!-- Run appc to create the deployable jar file -->
     <target name="appc" depends="jar_ejb">
<echo message="Generating container classes in ${apps.dir}/${ejb_jar}"/>
     <wlappc debug="${debug}"
     iiop="true"
     source="${dist}/std_${ejb_jar}"
     output="${apps.dir}/${ejb_jar}"
     />
     </target>
```

```
<!-- Compile EJB interfaces & client app into the clientclasses directory
-->
     <target name="compile_client">
     <javac srcdir="${source}"
     destdir="${client.classes.dir}"
     includes="${client_classes}"
     />
     </target>

<target name="run">
<java classname="examples.wlec.ejb.simpapp.Client">
</java>
</target>

     <target name="clean">
     <delete dir="${build}"/>
     <delete dir="${dist}"/>
     </target>
</project>
```

Modify the WLEC ConverterBean

The following listing provides a code example on how to modify the `wlec/ejb/simpapp` example `ConverterBean.java` file to interoperate with Tuxedo using WebLogic Tuxedo Connector.

- All changes are highlighted in bold and look like this: **`new code`**
- Statements that are no longer needed are commented out using **`//`** and look like this: **`// old code`**

Listing 3-5 Modified ConverterBean.java file

```
package examples.wlec.ejb.simpapp;

import javax.ejb.*;
```

```
import java.io.Serializable;
import java.util.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.omg.CORBA.*;
import com.beasys.Tobj.*;
import com.beasys.*;

/*These come from WebLogic Enterprise Simpapp sample */
//import SimpleFactory;
//import SimpleFactoryHelper;
//import Simple;
import simple.SimpleFactory;
import simple.SimpleFactoryHelper;
import simple.Simple;

/**
* <font face="Courier New" size=-1>ConverterBean</font> is a stateless
* SessionBean.
* This bean illustrates:
* <ul>
* <li> Accessing ISL/ISH process and then a WebLogic Enterprise server
* <li> No persistence of state between calls to the SessionBean
* <li> Application-defined exceptions
* </ul>
*
* @author Copyright (c) 1999-2001 by BEA Systems, Inc. All Rights Reserved.
*/
public class ConverterBean implements SessionBean {

static SimpleFactory simple_factory_ref;

// ---------------------------------------------------------------
// private variables
private SessionContext ctx;
private Context        rootCtx;
private ORB orb;

// ---------------------------------------------------------------
// SessionBean implementation

/**
* This method is required by the EJB Specification,
* but is not used by this example.
*
*/
public void ejbActivate()  {}
```

```
/**
* This method is required by the EJB Specification,
* but is not used by this example.
*
*/
public void ejbRemove() {}

/**
* This method is required by the EJB Specification,
* but is not used by this example.
*
*/
public void ejbPassivate() {}

/**
* Sets the session context.
*
* @param ctx               SessionContext context for session
*/

  public void setSessionContext(SessionContext ctx) {
this.ctx = ctx;
}

// Interface exposed to EJBObject

/**
* This method corresponds to the <font face="Courier New" size=-1>create</font>
* method in the home interface <font
*face="CourierNew"size=-1>ConverterHome.java</font>.
* The parameter sets of these two methods are identical. When the client calls the
* <font face="Courier New" size=-1>ConverterHome.create</font> method, the
* container allocates an instance of the EJBean and calls the
* <font face="Courier New" size=-1>ejbCreate</font> method.
*
* @exception               CreateException
*                         if there is an error while initializing the IIOP pool
* @see                     examples.wlec.ejb.simpapp.Converter
*/
public void ejbCreate () throws CreateException {
try {
// try {
// Properties p = new Properties();
// p.put(Context.INITIAL_CONTEXT_FACTORY,
// "weblogic.jndi.WLInitialContextFactory");
// InitialContext ic = new InitialContext(p);
// rootCtx = (Context)ic.lookup("java:comp/env");
//  }
```

```
//catch (NamingException ne) {
//  throw new CreateException("Could not lookup context");
// }

// Initialize the ORB.
String args[] = null;
Properties Prop;
Prop = new Properties();
Prop.put("org.omg.CORBA.ORBClass",
"weblogic.wtc.corba.ORB");

orb = (ORB)new InitialContext().lookup("java:comp/ORB");

initIIOPpool();
}
catch (Exception e) {
throw new CreateException("ejbCreate called: " + e);
}
}

/**
* Converts the string to uppercase.
*
* @param mixed             string input data
* @return                  ConverterResult conversion result
* @exception               examples.wlec.ejb.simpapp.ProcessingErrorException
*                          if there is an error while converting the string
*/
public ConverterResult toUpper(String mixed)
throws ProcessingErrorException
{
return convert("UPPER", mixed);
}

/**
* Converts the string to lowercase.
*
* @param mixed             string input data
* @return                  ConverterResult conversion result
* @exception               examples.wlec.ejb.simpapp.ProcessingErrorException
*                          if there is an error while converting the string
*/
public ConverterResult toLower(String mixed)
throws ProcessingErrorException
{
return convert("LOWER", mixed);
}

protected ConverterResult convert (String changeCase, String mixed)
```

```
throws ProcessingErrorException
{
String result;
try {
// Find the simple object.
Simple simple = simple_factory_ref.find_simple();

if (changeCase.equals("UPPER")) {
// Invoke the to_upper opeation on M3 Simple object
org.omg.CORBA.StringHolder buf = new org.omg.CORBA.StringHolder(mixed);
simple.to_upper(buf);
result = buf.value;
}
else
{
result = simple.to_lower(mixed);
}

}
catch (org.omg.CORBA.SystemException e) {
throw new ProcessingErrorException("Converter error: Corba system exception: "
+ e);
}
catch (Exception e) {
throw new ProcessingErrorException("Converter error: " + e);
}
return new ConverterResult(result);
}

// Private methods

/**
* Returns the WebLogic Enterprise Connectivity pool name.
*
* @return                   String IIOP pool name
*/
// private String getIIOPPoolName() throws ProcessingErrorException {
// try {
// return (String) rootCtx.lookup("IIOPPoolName");
//}
// catch (NamingException ne) {
// throw new ProcessingErrorException ("IIOPPoolName not found in context");
//}
//}

/**
* Initializes an IIOP connection pool.
*/
```

```
private void initIIOPpool() throws Exception {
try {
// Create the bootstrap object,
// Tobj_Bootstrap bootstrap =
// BootstrapFactory.getClientContext(getIIOPPoolName());

// Use the bootstrap object to find the factory finder.
// org.omg.CORBA.Object fact_finder_oref =
// bootstrap.resolve_initial_references("FactoryFinder") ;
org.omg.CORBA.Object fact_finder_oref =
orb.string_to_object("corbaloc:tgiop:simpapp/FactoryFinder");

// Narrow the factory finder.
FactoryFinder fact_finder_ref =
FactoryFinderHelper.narrow(fact_finder_oref);

// Use the factory finder to find the simple factory.
org.omg.CORBA.Object simple_fact_oref =
fact_finder_ref.find_one_factory_by_id(SimpleFactoryHelper.id());

// Narrow the simple factory.
simple_factory_ref =
SimpleFactoryHelper.narrow(simple_fact_oref);

}
catch (org.omg.CosLifeCycle.NoFactory e) {
throw new Exception("Can't find the simple factory: " +e);
}
catch (CannotProceed e) {
throw new Exception("FactoryFinder internal error: " +e);
}
catch (RegistrarNotAvailable e) {
throw new Exception("FactoryFinder Registrar not available: " +e);
}
//catch (InvalidName e) {
//   throw new Exception("Invalid name from resolve_initial_reference(): " +e);
//}
// catch (org.omg.CORBA.BAD_PARAM e) {
//  throw new Exception("Invalid TOBJADDR=//host:port property specified: " +e);
// }
catch (org.omg.CORBA.UserException e) {
throw new Exception("Unexpected CORBA user exception: " +e);
}
catch (org.omg.CORBA.SystemException e) {
throw new Exception("CORBA system exception: " +e);
}
}
```

```
}
```

Configure WebLogic Tuxedo Connector

Use the following steps to configure WebLogic Tuxedo Connector to connect WebLogic Server and the modified WLEC application:

1. Create a WTC Service
2. Create a Local Tuxedo Access Point
3. Create a Remote Tuxedo Access Point
4. Create an Imported Service

Create a WTC Service

Use the following steps to create and configure a WTC service using the WebLogic Server Administration Console:

1. In the Administration Console, expand Interoperability and select WTC Servers in the navigation tree.
2. On the WTC Servers page, click New.
3. On the Create a New WTC Server page, enter **My_WLEC_App** to identify this configuration in the name field.
4. Click OK.
5. Your new WTC Service appears in the WTC Servers list.

Create a Local Tuxedo Access Point

Note: When configuring the Network Address for a local access point, the port number used should be different from any port numbers assigned to other processes. Example: Setting the Network Address to `//mymachine:7001` is not valid if the WebLogic Server listening port is assigned to `//mymachine:7001`.

Use the following steps to configure a local Tuxedo access point:

1. In the Administration Console, expand Interoperability and select WTC Servers.

2. On the WTC Servers page, click the name of a WTC Service to access the settings page.
3. Click the Local APs tab.
4. Enter the following values for the following fields on the WTC Local Access Points page:

   In Access Point, enter **My_Local_WLS_Dom**.

   In Access Point Id, enter **`examples`**.
5. In Network Address, enter the network address and port of the WebLogic Server environment that will host this local domain.

   Example: `//my_WLS_machine:5678`
6. Click OK.

Create a Remote Tuxedo Access Point

Use the following steps to configure a remote Tuxedo access point:

1. In the Administration Console, expand Interoperability and select WTC Servers.
2. On the WTC Servers page, click the name of a WTC Service.
3. Click the Remote APs tab.
4. Enter the following values for the following fields on the WTC Remote Access Points page:

   In Access Point, enter **My_WLEC_Dom**.

   In Access Point Id, enter **TUXDOM**.

   In Local Access Point, enter **My_Local_WLS_Dom**.
5. In Network Address, enter the network address and port of the Tuxedo environment that will host this remote domain.

   Example: `//my_TUX_machine:5678`
6. Click OK.

Create an Imported Service

Use the following steps to configure an imported service:

1. In the Administration Console, expand Interoperability and select WTC Servers.
2. On the WTC Servers page, click the name of a WTC Service.

3. Click the Imported tab.
4. Enter the following values for the following fields on the WTC Imported Services page:

   In Resource Name, enter **//simpapp**.

   In Local Access Point, enter **My_Local_WLS_Dom**.

   In Remote Access Point List, enter **My_WLEC_Dom**.

   In Remote Name, enter *"//domain_id"* where *domain_id* is `DOMAINID` specified in the Tuxedo `UBBCONFIG` file. The maximum length of this unique identifier for CORBA domains is 15 characters and includes the //.

   Example: `//simpappff`
5. Click OK.

Run the simpapp Example

1. Open a new shell and change directories to your working Tuxedo CORBA `simpapp` example.
2. Set environment variables.

   NT\2000 users run the following command: `results\setenv.cmd`

   Unix users run the following command: `results\setenv.sh`
3. Boot the Tuxedo domain

   ```
   tmboot -y
   ```
4. Open a new shell and change directories to your WebLogic Server WLEC `simpapp` example.
5. Set environment variables. Update the following parameters:

Note: NT/2000 users modify and run the setExamplesEnv.cmd. Unix users copy ./config/examples/setExamplesEnv.sh script to your WLEC simpapp directory, then modify and run the setExamplesEnv.sh script.

6. Copy the `simple.idl` file from the Tuxedo CORBA `simpapp` example to your WebLogic Server WLEC `simpapp` example.
7. Build the `wlec_simpapp_corba.jar` file using ant.

   Enter the following command: **`ant`**
8. Use the WLS console to target **My_WLEC_App** to the server.

9. Run the client.

   Enter the following command: **`ant run`**

The Java application generates the following output:

```
Beginning simpapp.Client...

Start of Conversion for: It Works

Converting to lower case: It Works

...Converted: it works

Converting to upper case: It Works
.
..Converted: IT WORKS

Removing Converter

End simpapp.Client...
```

If you have a problem running the example, use the WTC tracing feature. See Monitoring the WebLogic Tuxedo Connector.