BEA WebLogic Server™

WebLogic Tuxedo Connector WLEC Migration Guide

BEA WebLogic Server Version 6.1
Document Date: November 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

WebLogic Tuxedo Connector Administration Guide

Document Date	Software Version
November 2001	BEA WebLogic Server 6.1

Contents

About This Document

1. Overview of WLEC to WebLogic Tuxedo Connector Migration

2. How to Modify WLEC Applications for WebLogic Tuxedo Connector

3. How to Convert the WLEC Simpapp Example for WebLogic Tuxedo Connector

About This Document

This document provides information on how to migrate WLEC applications to interoperate with Tuxedo using WebLogic Tuxedo Connector.

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 Convert the WLEC Simpapp Example for WebLogic Tuxedo Connector," provides an example of how to convert the WLEC simpapp example to interoperate with Tuxedo using the WebLogic Tuxedo Connector.

Audience

This document is written for system administrators and application developers who are interested in building distributed Java applications that interoperate between WebLogic Server and Tuxedo environments. It is assumed that readers are familiar with the WebLogic Server, Tuxedo, C++, Corba, XML and Java programming.

e-docs Web Site

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

How to Print the Document

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

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

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

Related Information

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

For more information about Java applications, refer to the Sun Microsystems, Inc. Java site at http://java.sun.com/.

Contact Us!

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

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

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

Convention	Usage

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String` *`CustomerName`*`;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*:

```
java utils.MulticastTest -n name -a address
     [-p portnumber] [-t timeout] [-s send]
```

Convention	Usage
`|`	Separates mutually exclusive choices in a syntax line. *Example*:

```
java weblogic.deploy [list|deploy|undeploy|update]
     password {application} {source}
```

Convention	Usage
`...`	Indicates one of the following in a command line: ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information

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

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:

- Overview
- Prerequisites
- Comparing WebLogic Tuxedo Connector and WLEC Functionality
- Key Differences Between WLEC and WebLogic Tuxedo Connector

Overview

Note: For more information on release life cycle, see Product Lifecycle Policy located at http://e-docs.bea.com/wls/platforms/eol.html#PLP.

WLEC will be classified a mature product after the release of WebLogic Server 6.1. 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 CORBAC++ server objects do not require server application changes.

Prerequisites

WebLogic Tuxedo Connector CORBA connectivity requires the following application platforms:

- Tuxedo 8.0
- WebLogic Server 6.1 Service Pack 2

Before you can start migrating your WLEC applications to WebLogic Tuxedo Connector, make sure that you have installed Tuxedo 8.0 and WebLogic Server 6.1 Service Pack 2. You may need to perform the following steps:

- If necessary, migrate your Tuxedo applications to Tuxedo 8.0. For more information, see Tuxedo Migration located at http://e-docs.bea.com/tuxedo/tux80/interm/migratn.htm.
- If necessary, migrate your WebLogic Server installation to WebLogic Server 6.1 Service Pack 2. For more information, see WebLogic Server 6.1 Migration located at http://e-docs.bea.com/wls/docs61/migration.html.

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, outbound only
Bi-directional transaction propagation	Yes	No, outbound only
Bi-directional bridge between JMS and /Q or Tuxedo services	yes	No
Conversational API	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.
Administration	Administered as a WebLogic Server Startup class that uses an XML configuration file.	Administered as a WebLogic Server service.
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 to the WLE 5.1 (T-Engine) CORBA and EJB servers over IIOP connection pools using the CORBA API.
Supported Platforms	Supports WebLogic Server 6.1 and BEA Tuxedo version 8.0.	Supports WebLogic Enterprise 5.1 or Tuxedo 8.0 and WebLogic Server versions 5.1 and greater.

2 How to Modify WLEC Applications for WebLogic Tuxedo Connector

The following sections provide an 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 your WLEC Applications

How to Modify Your Tuxedo Environment

Tuxedo users need to make the following environment changes:

- If an existing Tuxedo application is already using Tuxedo /T DOMAINS, then a new domain must be added to the domains configuration file for each connection to a WebLogic Tuxedo Connector instantiation.

- If the existing Tuxedo application does not use domains, then the domain servers must be added to the *TUXCONFIG* of the application. A new *DMCONFIG* must be created with a Tuxedo /T Domain entry corresponding to the WebLogic Tuxedo Connector instantiation.
- WebLogic Tuxedo Connector requires that the Tuxedo domain always have encoding turned on. `MTYPE` should always be unset or set to NULL in the *DMCONFIG* file.
- For more information on Tuxedo domains, see the *BEA TUXEDO Domains Guide*.
- Create a `cns` server process for the CORBA Name Service.

Note: For more information on Tuxedo domains, see the *Overview of the CORBA Name Service* located at http://e-docs.bea.com/tuxedo/tux80/naming/index.htm.

- Create a C++ proxy server. A Tuxedo C++ Corba server that implements the current ILD interface is required to conver the data types and call WebLogic Server EJB generated IDL interfaces.

How to Modify your WebLogic Server Environment

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

- How to Create an XML Configuration File
- How to Deploy your WebLogic Tuxedo Connector Applications
- How to Create Startup and Shutdown Classes

How to Create an XML Configuration File

Note: For more information on how to create an XML configuration file, see Create a WebLogic Tuxedo Connector XML Configuration File in the *WebLogic Tuxedo Connector Administration Guide*.

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

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

How to Configure Connections Between Domains

WebLogic Tuxedo Connector uses the Tuxedo /T Domain gateway to create a single network link between a WebLogic Server instance and a Tuxedo domain for all method invocations.

Use the `ConnectionPolicy` parameter in the `T_DM_LOCAL_TDOMAIN` and `T_DM_REMOTE_TDOMAIN` sections of BDMCONFIG to configure connections between domains. You can select ON_DEMAND, ON_STARTUP, INCOMING_ONLY, and LOCAL. For more information, see Configuring the Connections Between Domains in the *WebLogic Tuxedo Connector Administration Guide*.

How to Configure for Failover

To support failover, you must specify the remote domains responsible for executing a particular service. Specifically, you must specify each remote domain with a T_DM_REMOTE_TDOMAIN section in your XML configuration file. For more information, see Configuring Domains-level Failover and Failback in the *WebLogic Tuxedo Connector Administration Guide*.

How to Configure Authentication of Remote Domains

Domain gateways can be made to authenticate incoming connections requested by remote domains and outgoing connections requested by local domains. You can specify the level of security used by a particular local domain by setting the `SECURITY` parameter in the T_DM_LOCAL_TDOMAIN section of your XML configuration file. There are three levels of password security: `NONE, APP_PW, DM_PW`. For more information see, Authentication of Remote Domains in the *WebLogic Tuxedo Connector Administration Guide*.

How to Configure Access Control Lists

Access Control Lists (ACLs) limit the access to local services within a local domain by restricting the remote domains that can execute these services. Inbound policy from a remote domain is specified using the `AclPolicy` element. Outbound policy towards a remote domain is specified using the `CredentialPolicy` element. This allows WebLogic Server and Tuxedo applications to share the same set of users and the users are able to propagate their credentials from one system to the other. For more information, see How to Configure WebLogic Tuxedo Connector to Provide Security between Tuxedo and WebLogic Server in the *WebLogic Tuxedo Connector Administration Guide*.

How to Configure Link-Level Encryption

You can use encryption across domains to ensure data privacy. Configure this security mechanism by setting the `MINENCRYPTBITS` and `MAXENCRYPTBITS` parameters of the T_DM_LOCAL_TDOMAIN and the T_DM_REMOTE_TDOMAIN sections of your XML configuration file. For more information, see Link-Level Encryption in the *WebLogic Tuxedo Connector Administration Guide*.

How to Deploy your WebLogic Tuxedo Connector Applications

The following sections provide information on how to migrate your WLEC weblogic-ejb-jar.xml file and the build.xml file for use with WebLogic Tuxedo Connector.

- How to Update Your weblogic-ejb-jar.xml File
- How to Update Your build.xml File
- How to Generate a WebLogic Server EJB IDL Interface

How to Update Your weblogic-ejb-jar.xml File

This section will provide information on how to update your `weblogic-ejb-jar.xml` .file.

- remove the connection pool stuff
- add the proper stuff back in

How to Update Your build.xml File

This section will provide information on how to update your build.xml file.

- what ever the steps are

How to Generate a WebLogic Server EJB IDL Interface

Use the `weblogic.ejbc` utility to extract the IDL interface from the WebLogic Server EJB interface.

- what ever the steps are

How to Create Startup and Shutdown Classes

In order to use the WebLogic Tuxedo Connector, you must create a WebLogic Server Startup and Shutdown class. For more information see, Create a StartUp Class for the WebLogic Tuxedo Connector and Create a Shutdown Class for the WebLogic Tuxedo Connector in the *WebLogic Tuxedo Connector Administration Guide*.

How to Modify your WLEC Applications

This section provides information on how to modify WLEC applications to interoperate with WebLogic Server and Tuxedo CORBA objects using WebLogic Tuxedo Connector.

- How to Modify Your WLEC EJB to Reference CORBA Objects Used by WebLogic Tuxedo Connector
- How to Modify WLEC Transactions for WebLogic Tuxedo Connector

How to Modify Your WLEC EJB 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:

- Remove References to Tobj_Bootstrap Object
- Remove References to the FactoryFinder Object
- How to Initialize the WTC ORB
- How to Get the Simple Factory
- How to Get the Simple Object
- How to Invoke on the Simple Object

Remove References to Tobj_Bootstrap Object

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

```
Tobj_Bootstrap myBootstrap =
Tobj_BootstrapFactory.getClientContext("myPool");
```

where

- The `getClientContext()` method returns the `Tobj_Bootstrap` object associated with `mypool`.
- *`myPool`* is the name of the WLEC connection pool for the desired BEA Tuxedo domain.

Remove references to the `Tobj_Bootstrap` object. WebLogic Tuxedo Connector uses the XML configuration file to provide information required to create a path to the Tuxedo service. This pathway is created when the WebLogic Server is started and the WebLogic Tuxedo Connector XML configuration file is loaded.

Remove References to the FactoryFinder Object

Remove references to the FactoryFinder object. This object is obtained using the Tobj_Bootstrap object.

```
org.omg.CORBA.Object myFFObject =
    myBootstrap.resolve_initial_references("FactoryFinder");
FactoryFinder myFactFinder =
    FactoryFinderHelper.narrow(myFFObject);
```

where

- `myBootstrap` is the `Tobj_Bootstrap` object for the desired BEA Tuxedo domain.
- The `resolve_initial_references()` method returns the object reference for the FactoryFinder object.
- The `FactoryFinderHelper` interface provides auxiliary functionality for the `FactoryFinder` interface, notably the `narrow()` method.
- The `narrow()` method casts the object reference to point to a FactoryFinder object.

How to Initialize the WTC ORB

To use CORBA with the WebLogic Tuxedo Connector, you must use the WTC ORB. WLEC uses the `weblogic.jndi.WLInitialContextFactory` to return a context used by the `Tobj_Bootstrap` object. 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 = ORB.init(args, Prop);
```

How to Get the Simple Factory

The WebLogic Tuxedo Connector uses the CosNaming service to get a reference to an object in the remote Tuxedo CORBA domain. This is accomplished by using a `corbaloc:tgiop` or `corbaname:tgiop` object reference. Example:

```
// Get the simple factory.
org.omg.CORBA.Object simple_fact_oref =
orb.string_to_object("corbaname:tgiop:simpapp#simple_factory");


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

Where:

- `simpapp` is the domain id of the Tuxedo domain specified in the Tuxedo UBB.
- `simple_factory` is the name that the object reference was bound to in the CosNaming server in Tuxedo CORBA..
- The *`SimpleFactoryHelper`* interface provides auxiliary functionality for the interface, notably the `narrow()` method.
- The `narrow()` method casts the object reference to point to the object factory.

How to Get the Simple Object

Get the BEA Tuxedo CORBA object using the object's `find()` method. For example, if you are accessing an object named `Simple`, use the following code:

```
// Find the simple object.
Simple simple = simple_factory_ref.find_simple();
```

where the factory provides the `find_simple()` method for finding the `Simple` object.

For information about the FactoryFinder object, see the *CORBA C++ Programming Reference* in the BEA Tuxedo documentation.

How to Invoke on the Simple Object

Perform your task by invoking upon the CORBA object deployed in Tuxedo using the CORBA Java API.

```
// Invoke the to_upper opeation on Simple object
org.omg.CORBA.StringHolder buf =
     new org.omg.CORBA.StringHolder(mixed);
simple.to_upper(buf);
result = buf.value;
```

How to Modify WLEC Transactions for WebLogic Tuxedo Connector

The following sections provide information on how to modify WLEC applications that use transactions to interoperate with Tuxedo using WebLogic Tuxedo Connector.

- Peter Bower's information goes here

CHAPTER

3 How to Convert the WLEC Simpapp Example for WebLogic Tuxedo Connector

The following section provides information on the steps required to convert your WLEC Simpapp example to work using the WebLogic Tuxedo Connector.

- Create the C++ Proxy Server
- Modify the Tuxedo Environment
- Modify the WebLogic Server Environment
- Modify the WLEC ConverterBean
- Run the simpapp Example

Create the C++ Proxy Server

This section will provide information on how to create the proxy server for this example.

- may include generating IDLs

Modify the Tuxedo Environment

This section needs to be updated to include the proxy server.

1. Create a working of the Tuxedo Corba `simpapp` example. Copy the Tuxedo Corba `simpapp` example from you 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 the JAVA_HOME environment variable to the location of your Java JDK.
   b. 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.
   c. Run the Java client.

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

4. Copy the `results\ubb` file to your working directory and rename the new file `ubbdomain`.
5. Edit the `ubbdomain` file using a text editor, such as vi or WordPad.
6. Add a server for the COSNaming Service to the `*SERVERS` section.

```
cns
     SRVGRP=SYS_GRP
     SRVID=6
      CLOPT="-A --"
```

7. Add the Tuxedo gateway servers to the `*SERVERS` section.

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

8. Save the `ubbdomain` file.
9. Create a `domconfig` file using a text editor, such as vi or WordPad. Replace all <bracketed> items with information for your environment.

```
*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
```

10. Load the ubbdomain file

```
tmloadcf -y ubbdomain
```

11. .Load the dom1config file

```
set BDMCONFIG=d:\mydomain\simpapp\bdmconfig
dmloadcf -y dom1config
```

12. Boot the Tuxedo domain

```
tmboot -y
```

13. Verify the Tuxedo environment.

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

14. Shutdown the Tuxedo Environment.

```
tmshutdown -y
```

Modify the WebLogic Server Environment

1. Create an XML configuration file. For more information, see “How to Create an XML Configuration File” on page 2-2.

Listing 3-1 Example XML Configuration File for a CORBA Server Application

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

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

<!--Java and XML-->
     <WTC_CONFIG>
     <BDMCONFIG>
          <T_DM_LOCAL_TDOMAIN AccessPoint="examples">
               <WlsClusterName>Coolio</WlsClusterName>
               <AccessPointId>examples</AccessPointId>
               <Type>TDOMAIN</Type>
               <Security>NONE</Security>
               <ConnectionPolicy>ON_DEMAND</ConnectionPolicy>
               <BlockTime>30</BlockTime>
               <NWAddr>//localhost:20304</NWAddr>
          </T_DM_LOCAL_TDOMAIN>
          <T_DM_REMOTE_TDOMAIN AccessPoint="TUXDOM">
               <LocalAccessPoint>examples</LocalAccessPoint>
               <AccessPointId>TUXDOM</AccessPointId>
               <Type>TDOMAIN</Type>
               <NWAddr>//localhost:20305</NWAddr>
          </T_DM_REMOTE_TDOMAIN>
          <T_DM_IMPORT
               ResourceName="//simpapp"
               LocalAccessPoint="examples"
               RemoteAccessPointList="TUXDOM">
               <TranTime>600</TranTime>
          </T_DM_IMPORT>
</BDMCONFIG>
</WTC_CONFIG>
```

2. Validate the `bdmconfig.xml file` using the `WTCValidateCF` utility:

   `java weblogic.wtc.gwt.WTCValidateCF bdmconfig.xml`

3. Modify your `weblogic-ejb-jar.xml` file.
4. Modify your `build.xml` file.
5. Boot your WLS examples server
6. Configure a WTC StartUp Class. You need to provide a classname and the fully qualified name of the `bdmconfig.xml` file.

Set the classname to:
`weblogic.wtc.gwt.WTCStartup`

Set the arguments field to:
`BDMCONFIG=path_name_of_your_bdmconfig.xml_file`

7. Configure a WTC ShutDown Class

   Set the classname to:
   `weblogic.wtc.gwt.WTCShutdown`

8. Shutdown your WLS examples server.

Modify the WLEC ConverterBean

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

- Statements you need are highlighed in bold and look like this: **`new code`**
- Statements you need to remove marked Strikethrough and look like this : ~~remove these statements~~

Listing 3-2 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;

/**
* <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;

// -------------------------------------------------------------------
// 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="Courier New"
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");
}
```

~~//initIIOPpool();~~
initWTCorb();

```
}
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") ;~~

~~// 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);~~
~~}~~
~~}~~
/**
* Initializes WTC ORB.
*/

public void initWTCorb() throws ProcessingErrorException {
try {
// Initialize the ORB.
String args[] = null;
Properties Prop;
Prop = new Properties();
Prop.put("org.omg.CORBA.ORBClass",
"weblogic.wtc.corba.ORB");

ORB orb = ORB.init(args, Prop);

// Get the simple factory.
org.omg.CORBA.Object simple_fact_oref =
orb.string_to_object("corbaname:tgiop:simpapp#simple_factory");

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

}
catch (Exception e) {
throw new ProcessingErrorException("Can't call TUXEDO CORBA server: " +e);
}
}
```

```
}
```

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 modify the tux.env file located at $TUXDIR.
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:
   - Set the APPLICATIONS to the location of the applications directory in your WLS domain
   - Set the CLIENT_CLASSES to build

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. Build the `new_wtc_toupper.jar` file using ant.

   Enter the following command: **`ant`**
7. Boot your WLS application server
8. Check to see that the new_wtc_toupper.jar is deployed. Use the WLS console or check the `config.xml` file. Manually deploy the EJB if needed.
9. Run the client.

```
java -classpath %CLASSPATH%;%CLIENT_CLASSES% examples.wlec.ejb.simpapp.Client
```

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...
Use TraceLevel
```

If you have a problem running the simpappcns example, use the WTC tracing feature. Add the TraceLevel parameter to the StartUp Class Argument list. A TraceLevel value of 100000 traces all calls for WebLogic Tuxedo Connector components.

Example:

```
BDMCONFIG=path_to_my_bdmconfig.xml_file,TraceLevel=100000
```