# BEA WebLogic Enterprise

## CORBA, J2EE, and Tuxedo Interoperability and Coexistence

WebLogic Enterprise 5.1
Document Edition 5.1
May 2000

## Copyright



## Restricted Rights Legend



## Trademarks or Service Marks



**CORBA, J2EE, and Tuxedo Interoperability and Coexistence**



| Document Edition | Date | Software Version |
|---|---|---|
| 5.1 | May 2000 | BEA WebLogic Enterprise 5.1 |

# Contents


## About This Document

What You Need to Know ..... vii
e-docs Web Site ..... viii
How to Print the Document ..... viii
Related Information ..... viii
Contact Us! ..... ix
Documentation Conventions ..... ix

## 1. Introduction

Interoperability Among the CORBA, J2EE, and Tuxedo Programming Models ..... 1-1
BEA Clients and Servers ..... 1-2
T-Engine Server Interoperability ..... 1-3
Java Enterprise Tuxedo (JET) Software ..... 1-5
Transactions and Security ..... 1-6
T-Engine Client and Server Interoperability ..... 1-6
Transactions and Security ..... 1-8
A Note About BEA Jolt ..... 1-9
RMI Clients and the WebLogic RMI-on-IIOP Protocol ..... 1-9
J-Engine and T-Engine Interoperability ..... 1-10
Third-party ORB Interoperability ..... 1-11
T-Engine Interdomain Interoperability ..... 1-13
WebLogic Enterprise and Tuxedo Domains Interoperability ..... 1-15
Overview of the Interoperability Sample Applications ..... 1-17

## 2. EJB-to-CORBA/Java Simpapp Sample Application


How the EJB-to-CORBA/Java Simpapp Sample Application Works ..... 2-2
Software Prerequisites ..... 2-3
Implementing the Bridge Object to Invoke a CORBA/Java Object ..... 2-3
The OMG IDL Code for the EJB-to-CORBA/Java Simpapp Interfaces ... 2-5
Building and Running the EJB-to-CORBA/Java Simpapp Sample Application ..... 2-6
Verifying the Settings of the Environment Variables ..... 2-7
Verifying the Environment Variables ..... 2-8
Changing the Environment Variables ..... 2-9
Copying the Files for the Java Simpapp Sample Application into a Work Directory ..... 2-10
Files in the Work Directory ..... 2-11
EJB Simpapp Files ..... 2-11
CORBA/Java Simpapp files ..... 2-12
Utility Files ..... 2-12
Changing the Protection Attribute on the Files for the EJB-to-CORBA/Java Simpapp Sample Application ..... 2-13
Executing the runme Command ..... 2-14
Running the Sample Application ..... 2-15
Processes and Files Generated by the EJB-to-CORBA/Java Simpapp Sample Application ..... 2-16
Processes Started ..... 2-16
Files Generated in the corbaj Directory ..... 2-17
Files Generated in the ejb_corbaj Directory ..... 2-19
Files Generated in the results Directory ..... 2-19
Stopping the EJB-to-CORBA/Java Simpapp Sample Application ..... 2-21


## 3. CORBA/C++-to-EJB Simpapp Sample Application


How the CORBA/C++-to-EJB Simpapp Sample Application Works ..... 3-2
Software Prerequisites ..... 3-3
Implementing the Bridge Object to Invoke an EJB ..... 3-3
The OMG IDL Code for the CORBA/C++-to-EJB Simpapp Interfaces ..... 3-5
Building and Running the CORBA/C++-to-EJB Simpapp Sample Application ..... 3-6
Verifying the Settings of the Environment Variables ..... 3-7

Verifying the Environment Variables ..... 3-8
Changing the Environment Variables ..... 3-9
Copying the Files for the CORBA/C++-to-EJB Simpapp Sample Application into a Work Directory ..... 3-10
Files in the Work Directory ..... 3-11
CORBA/C++ Client Files ..... 3-11
EJB Server Files ..... 3-12
Utility Files ..... 3-12
Changing the Protection Attribute on the Files for the CORBA/C++-to-EJB Simpapp Sample Application ..... 3-13
Executing the runme Command ..... 3-14
Running the Sample Application ..... 3-15
Processes and Files Generated by the CORBA/C++-to-EJB Simpapp Sample Application ..... 3-16
Processes Started ..... 3-16
Files Generated in the cpp Directory ..... 3-17
File Generated in the cpp_ejb Directory ..... 3-19
Files Generated in the results Directory ..... 3-19
Stopping the CORBA/C++-to-EJB Simpapp Sample Application ..... 3-21

## 4. CORBA/Java-to-Tuxedo Simpapp Sample Application

How the CORBA/Java-to-Tuxedo Simpapp Sample Application Works ..... 4-2
Key Application Components ..... 4-2
Application Flow ..... 4-3
OMG IDL Code for the CORBA/Java-to-Tuxedo Simpapp Interfaces ..... 4-3
Software Prerequisites ..... 4-5
Example Code ..... 4-5
Building and Running the CORBA/Java-to-Tuxedo Simpapp Sample Application ..... 4-6
Step 1: Verify the Settings of Environment Variables ..... 4-7
Required Environment Variables ..... 4-7
Optional Environment Variables ..... 4-7
Verifying the Environment Variables ..... 4-8
Changing the Environment Variables ..... 4-9
Step 2: Copy the Files into a Work Directory ..... 4-10
Copying the Files ..... 4-10

Files Copied to the Working Directory ..... 4-11
Step 3: Change the Protection Attribute on the Files ..... 4-13
Step 4: Run the CORBA/Java-to-Tuxedo Simpapp Sample Application 4-13
Executing the runme Command ..... 4-13
Running the Sample Application Manually ..... 4-15
Server Processes Started by the Sample Application ..... 4-16
Files Generated by the Sample Application ..... 4-16
Stopping the CORBA/Java-to-Tuxedo Simpapp Sample Application ..... 4-20

## 5. EJB-to-Tuxedo Simpapp Sample Application

How the EJB-to-Tuxedo Simpapp Sample Application Works ..... 5-2
Key Application Components ..... 5-2
Application Flow ..... 5-3
Software Prerequisites ..... 5-3
Example Code ..... 5-3
Building and Running the EJB-to-Tuxedo Simpapp Sample Application ..... 5-5
Step 1: Verify the Settings of Environment Variables ..... 5-5
Required Environment Variables ..... 5-5
Optional Environment Variables ..... 5-6
Verifying the Environment Variables ..... 5-7
Changing the Environment Variables ..... 5-8
Step 2: Copy the Files into a Work Directory ..... 5-8
Copying the Files ..... 5-9
Files Copied to the Working Directory ..... 5-9
Step 3: Change the Protection Attribute on the Files for the EJB-to-Tuxedo Simpapp Sample Application ..... 5-12
Step 4: Run the EJB-to-Tuxedo Simpapp Sample Application ..... 5-12
Executing the runme Command ..... 5-12
Running the Sample Application Manually ..... 5-14
Server Processes Started by the Sample Application ..... 5-15
Files Generated by the Sample Application ..... 5-15
Stopping the EJB-to-Tuxedo Simpapp Sample Application ..... 5-18

## Index

# About This Document

This document describes how to build and run the suite of sample applications, which show how Enterprise JavaBeans and CORBA objects can coexist in the same BEA WebLogic Enterprise™ application.

This document includes the following topics:

- Chapter 1, “Introduction,” provides a high-level overview of the interoperability and coexistence capabilities in the WebLogic Enterprise system among the CORBA, J2EE, and Tuxedo® programming models. This chapter also describes the set of interoperability sample applications provided with the WebLogic Enterprise software.
- Chapter 2, “EJB-to-CORBA/Java Simpapp Sample Application,” describes how to build and run the EJB-CORBA/Java Simpapp sample application.
- Chapter 3, “CORBA/C++-to-EJB Simpapp Sample Application,” describes how to build and run the CORBA/C++-EJB Simpapp sample application.
- Chapter 4, “CORBA/Java-to-Tuxedo Simpapp Sample Application,” describes how to build and run the CORBA/Java-to-Tuxedo Simpapp sample application.
- Chapter 5, “EJB-to-Tuxedo Simpapp Sample Application,” describes how to build and run the EJB-to-Tuxedo Simpapp sample application.

## What You Need to Know

This document is intended for programmers who are interested in creating secure, scalable, transaction-based server applications. It assumes you are knowledgeable with CORBA, Enterprise JavaBeans, and the C++ and Java programming languages.

# e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

# How to Print the Document

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

A PDF version of this document is available on the WebLogic Enterprise 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 Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have Adobe Acrobat Reader installed, you can download it for free from the Adobe Web site at http://www.adobe.com/.

# Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA Tuxedo, distributed object computing, transaction processing, C++ programming, and Java programming, see the *Bibliography* in the WebLogic Enterprise online documentation.

# Contact Us!

Your feedback on the BEA WebLogic Enterprise 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 Systems, Inc. professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.1 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, contact BEA Customer Support through BEA WebSUPPORT at 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 | Item |
|---|---|
| **boldface text** | Indicates terms defined in the glossary. |
| Ctrl+Tab | Indicates that you must press two or more keys simultaneously. |

| Convention | Item |
|---|---|
| *italics* | Indicates emphasis or book titles. |
| `monospace text` | Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.<br>*Examples*:<br>`#include <iostream.h> void main ( ) the pointer psz`<br>`chmod u+w *`<br>`\tux\data\ap`<br>`.doc`<br>`tux.doc`<br>`BITMAP`<br>`float` |
| **`monospace boldface text`** | Identifies significant words in code.<br>*Example*:<br>`void` **`commit`** `( )` |
| *`monospace italic text`* | Identifies variables in code.<br>*Example*:<br>`String` *`expr`* |
| UPPERCASE TEXT | Indicates device names, environment variables, and logical operators.<br>*Example*s:<br>LPT1<br>SIGNON<br>OR |
| `{ }` | Indicates a set of choices in a syntax line. The braces themselves should never be typed. |
| `[ ]` | Indicates optional items in a syntax line. The brackets themselves should never be typed.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`<br>`[-l` *`file-list`*`]...` |

| Convention | Item |
|---|---|
| \| | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |
| . . . | Indicates one of the following in a command line:<br>■ That an argument can be repeated several times in a command line<br>■ That the statement omits additional optional arguments<br>■ That you can enter additional parameters, values, or other information<br>The ellipsis itself should never be typed.<br>*Example*:<br>`buildobjclient [-v] [-o name ] [-f file-list]...`<br>`[-l file-list]...` |
| .<br>.<br>. | Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed. |

CHAPTER

# 1 Introduction

This topic includes the following sections:

- Interoperability Among the CORBA, J2EE, and Tuxedo Programming Models

  This topic describes the interoperability and coexistence capabilities in the WebLogic Enterprise (WLE) system among the CORBA, J2EE, and Tuxedo programming models.

- Overview of the Interoperability Sample Applications

  This topic describes the interoperability sample applications provided with the WebLogic Enterprise software. The sample applications provide client and server programmers with information about the basic concepts of combining Enterprise JavaBeans (EJBs) and CORBA objects in the same WebLogic Enterprise application.

This chapter does not discuss specific interoperability or coexistence details with regards to WebLogic Server or WebLogic Enterprise Connectivity.

## Interoperability Among the CORBA, J2EE, and Tuxedo Programming Models

The key interoperability features are presented in the following categories:

- T-Engine Server Interoperability
- T-Engine Client and Server Interoperability
- J-Engine and T-Engine Interoperability

- Third-party ORB Interoperability
- T-Engine Interdomain Interoperability
- WebLogic Enterprise and Tuxedo Domains Interoperability

First, a summary description of BEA clients, servers, the T-Engine, and the J-Engine follows.

# BEA Clients and Servers

Note the following definitions:

- **BEA client**

  A BEA client can be any of the following entities, which exist outside the BEA domain and must use a listener/handler as a gateway to the domain:

  - Jolt® client application (via the Jolt listener/handler)
  - Tuxedo /WS client application (via the Tuxedo /WS listener/handler)
  - WebLogic Enterprise CORBA client application (via the IIOP listener/handler)
  - ActiveX client application (via the IIOP listener/handler)
  - RMI client application (via the IIOP listener/handler)

  Note that BEA clients invoking other BEA clients is not supported.

- **BEA server**

  A BEA server can fall into one of two general categories:

  - Tuxedo engine, or **T-Engine**, servers. T-Engine servers include Tuxedo services, CORBA objects, and EJBs that run on the Tuxedo-based WebLogic Enterprise infrastructure. These servers run within the administrative unit of a WebLogic Enterprise domain, and are configured via a `UBBCONFIG` file.
  - Java engine, or **J-Engine** servers. J-Engine servers include EJBs, Servlets, and Java Server Pages (JSPs) that run on the WebLogic Server-based infrastructure.

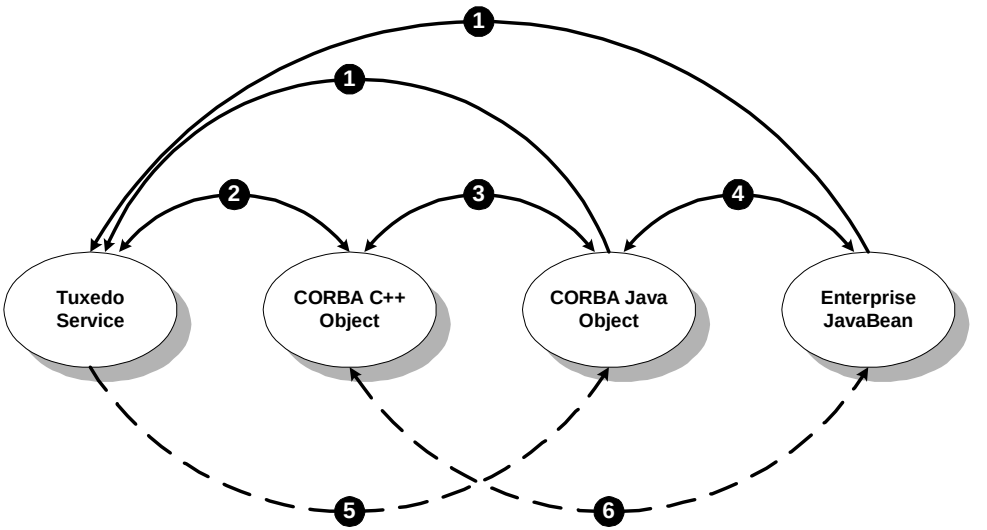
# T-Engine Server Interoperability

This section describes the interoperability among the following T-Engine server components:

- Tuxedo service
- CORBA C++ object
- CORBA Java object
- Enterprise JavaBean

Figure 1-1 shows the direct interoperability support among the various T-Engine server applications. The numbered callouts in the figure are explained in the text that follows the figure. In this figure, the solid black arrows show the direct invocation paths that are supported. The dotted arrows show indirect invocation paths that are supported; for example, a Tuxedo service can invoke a CORBA Java object using either a CORBA C++ proxy object, or a C++ client stub file compiled from the OMG IDL for that Java object.

**Figure 1-1 T-Engine Server Interoperability**



Note the following details about the preceding figure:

1. *CORBA Java object or Enterprise JavaBean invoking a Tuxedo service*

   WebLogic Enterprise provides the Java Enterprise Tuxedo (JET) API that you can use to have either a CORBA Java object or an EJB invoke a Tuxedo service running in the WebLogic Enterprise domain. An example of a CORBA Java object invoking a Tuxedo service using JET is described in Chapter 4, "CORBA/Java-to-Tuxedo Simpapp Sample Application," and an example of an EJB application invoking a Tuxedo service using JET is described in Chapter 5, "EJB-to-Tuxedo Simpapp Sample Application." (Note that RMI server applications can also run in the WebLogic Enterprise T-Engine domain, and they can also use JET to invoke Tuxedo services in that domain.)

   For considerations about using the JET software, see "Java Enterprise Tuxedo (JET) Software" on page 1-5.

2. *Tuxedo service invoking a CORBA C++ object and vice versa*

   A C++ object can include ATMI calls to Tuxedo services. See the Wrapper University sample application, available from the *Guide to the University Sample Applications*, for an example application that shows this feature.

   A Tuxedo service can invoke a CORBA C++ object using the compiled C++ client stub file for that object. (One way to do this is to implement the Tuxedo service as a C-callable C++ function that invokes the client stub file for the C++ object. If you use this approach, note that you need to link in the C++ ORB libraries when you build the Tuxedo service.)

3. *CORBA C++ object invoking a CORBA Java object and vice versa*

   CORBA C++ and CORBA Java objects that run in the same WebLogic Enterprise domain can invoke each other directly. For information about invoking across WebLogic Enterprise domains, see the section "T-Engine Interdomain Interoperability" on page 1-13.

4. *CORBA Java object invoking an EJB and vice versa*

   In the WebLogic Enterprise environment, a CORBA Java object can invoke methods on an EJB directly. See Chapter 3, "CORBA/C++-to-EJB Simpapp Sample Application," for an example application that includes a CORBA Java object that invokes an EJB.

5. *Tuxedo service invoking a CORBA Java object*

   A Tuxedo service can invoke a CORBA Java object by compiling the Java object's OMG IDL file with the `idl` command, which produces a C++ client stub file that the Tuxedo service can invoke, using an approach similar to the one described in point 2.

6. *EJB invoking a CORBA C++ object and vice versa*

   Chapter 2, "EJB-to-CORBA/Java Simpapp Sample Application," shows an example of an EJB invoking a CORBA Java object. You can extend this example to include a CORBA C++ object by designing the Java object in that application to serve as an intermediary, or wrapper, object that delegates invocations from the EJB to the C++ object, and vice versa. An alternative means for having an EJB invoke a C++ object is to compile the OMG IDL file for the C++ object using the `m3idltojava` command, which produces a Java client stub file that the EJB can invoke directly.

## Java Enterprise Tuxedo (JET) Software

The WebLogic Enterprise software includes the JET API, which allows T-Engine Java entities -- namely, EJBs and CORBA Java objects -- to make ATMI calls on Tuxedo services that exist in either the same WebLogic Enterprise domain or in a separate WebLogic Enterprise domain. JET is a server-side adaptation of BEA Jolt. JET shares some of its software components with Jolt, including the Repository Editor and the bulk loader. To take full advantage of all the capabilities of JET, you need to install the Jolt software, which is included with the WebLogic Enterprise software.

For more information about JET, see *Using Java Enterprise Tuxedo (JET)*. For more information about installing Jolt, see the *Jolt Installation Guide*, which is included in the WebLogic Enterprise package.

**Note:** JET cannot be used by Java clients to invoke Tuxedo services; this capability is provided by BEA Jolt, which is summarized in "Transactions and Security" on page 1-8.

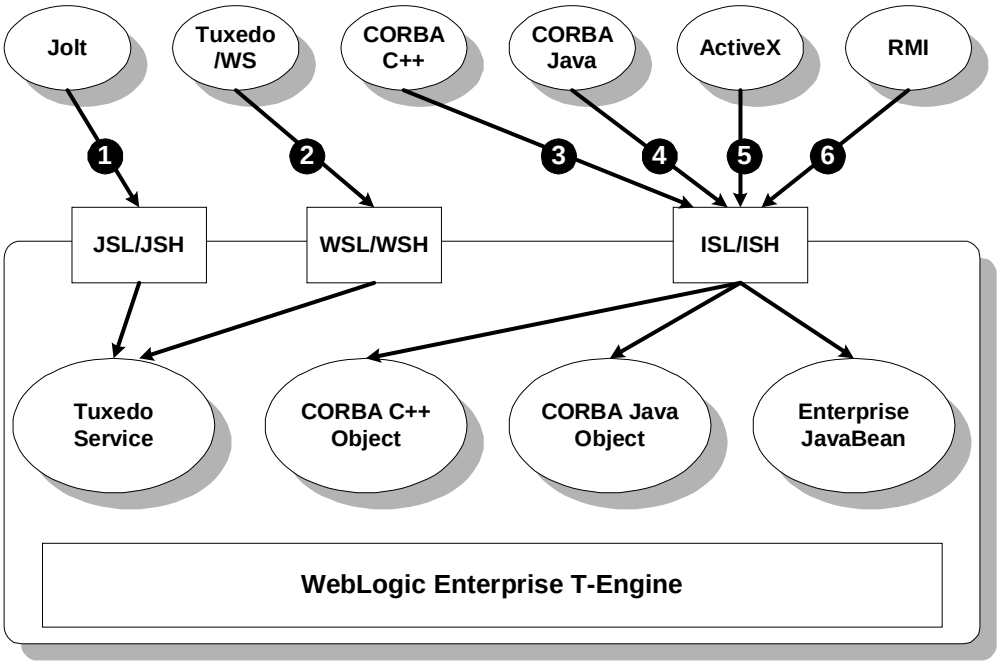
## Transactions and Security

Transaction and security context propagation among BEA server applications running in a WebLogic Enterprise domain is fully supported.

# T-Engine Client and Server Interoperability

Figure 1-2 shows the interoperability support among BEA clients invoking BEA servers.

**Figure 1-2 T-Engine Client and Server Interoperability**



Note the following details in the preceding figure:

1. *Jolt client application invoking a Tuxedo service*

   A Jolt client can invoke a Tuxedo service running in the WebLogic Enterprise domain via a Jolt listener/handler. For more information about Jolt, see the BEA Jolt online documentation.

2. *Tuxedo /WS client application invoking a Tuxedo service*

   A Tuxedo /WS client application can invoke a Tuxedo service running in the WebLogic Enterprise domain via the Workstation listener/handler.

3. *BEA CORBA C++ client application invoking a CORBA object*

   A BEA CORBA C++ client application can invoke both CORBA C++ and Java objects running in a WebLogic Enterprise domain via the IIOP listener/handler. For more information, see *Creating CORBA Client Applications*.

4. *BEA CORBA Java client application invoking a CORBA server-side object*

   A BEA CORBA Java client application can invoke both CORBA C++ and Java objects running in a WebLogic Enterprise domain via the IIOP listener/handler. For more information, see *Creating CORBA Client Applications*.

5. *BEA ActiveX client application invoking a CORBA server-side object*

   A BEA ActiveX client application can invoke both CORBA C++ and Java objects running in a WebLogic Enterprise domain via the IIOP listener/handler. For more information, see *Creating CORBA Client Applications*.

6. *RMI client application invoking an Enterprise JavaBean*

   An RMI client application can invoke an Enterprise JavaBean running in the WebLogic Enterprise domain via the IIOP listener/handler using the WebLogic RMI-on-IIOP protocol. For more information, see *Using RMI in a WebLogic Enterprise Environment*. (Note that an RMI client can also invoke an RMI server running in a WebLogic Enterprise domain.)

The following additional invocation paths are also supported in the WebLogic Enterprise environment via proxy objects or servers:

- *BEA CORBA C++ client application invoking a Tuxedo service*

  You can create a C++ client with a set of operations that map one-to-one with calls to Tuxedo services using an intermediary C++ server-side object. See the

Wrapper University sample application for an example application that shows this feature, available in the *Guide to the University Sample Applications*.

- *WebLogic Enterprise RMI client application invoking a CORBA C++ object*

  A WebLogic Enterprise RMI client application can invoke a CORBA C++ object by using an EJB and a CORBA Java object in the server process as intermediaries. For an example, you can extend the sample application described in Chapter 2, "EJB-to-CORBA/Java Simpapp Sample Application," as follows:

  - The RMI client application invokes the EJB to initiate the request.
  - The CORBA Java object, which is invoked by the EJB, delegates the invocation to the C++ object.

- *WebLogic Enterprise RMI client application invoking a CORBA Java object*

  A WebLogic Enterprise RMI client application can invoke a CORBA Java object by using an EJB as an intermediary. For an example, you can extend the sample application described in Chapter 2, "EJB-to-CORBA/Java Simpapp Sample Application," to have the RMI client application initiate the request instead of the EJB.

- *Tuxedo/WS client application invoking a CORBA C++ object*

  Interoperability is provided via a Tuxedo service wrapper. You create a Tuxedo service wrapper as a CORBA C++ object that runs in the WebLogic Enterprise domain and that makes invocations on the legacy CORBA C++ object.

- *Tuxedo/WS client application invoking a CORBA Java object*

  Interoperability is provided via a Tuxedo service wrapper.

- *Tuxedo/WS client application invoking an EJB*

  Interoperability is provided via a Tuxedo service wrapper on a CORBA Java object in the server process, which then delegates the invocation to the EJB.

## Transactions and Security

Transaction and security context propagation between BEA client and server applications is fully supported, with the following restrictions:

- BEA client applications can demarcate a transaction -- that is, they can explicitly begin, suspend, resume, and commit a transaction -- but they cannot participate in a transaction.

For example, a client can begin a transaction and make multiple invocations on services and objects within the domain, and those services and objects can in turn make invocations on yet other services and objects. However, the client application cannot, within the scope of that transaction, perform operations locally and have them included in that transaction. That is, if the client application starts a transaction, invokes an object within the domain, then writes data to a database local to the client, the local database operation cannot not be included in the transaction.

- When a client application authenticates itself to the domain, and invokes various services and objects in the domain -- which in turn may invoke other services and objects in the domain -- the client's security context is passed along with each operation. However, if in the course of satisfying a client request, a service in one domain makes an invocation on a service in a second domain, the client's security context cannot be passed to the second domain. The service in the second domain does not have knowledge of the original client.

## A Note About BEA Jolt

BEA Jolt provides a means for allowing Java clients to make ATMI calls on Tuxedo services that exists in a Tuxedo or WebLogic Enterprise domain. Jolt also provides a means for allowing J-Engine servers to invoke T-Engine Tuxedo services . This latter capability is performed via Jolt connection pools, which is shown in “J-Engine and T-Engine Interoperability” on page 1-10. BEA provides Jolt with the WebLogic Enterprise software.

For more information, see the following documents:

- For more information about Jolt, see the BEA Jolt version 1.2 online documentation.
- For more information about setting up Jolt connection pools to connect J-Engine servers to T-Engine Tuxedo services, see *Using WebLogic Enterprise Connectivity*.

## RMI Clients and the WebLogic RMI-on-IIOP Protocol

RMI clients of EJBs running in a T-Engine domain must use the WebLogic RMI-on-IIOP protocol. This is a proprietary protocol and is different from the RMI-over-IIOP protocol, which is used by clients of the BEA WebLogic Server™ system.

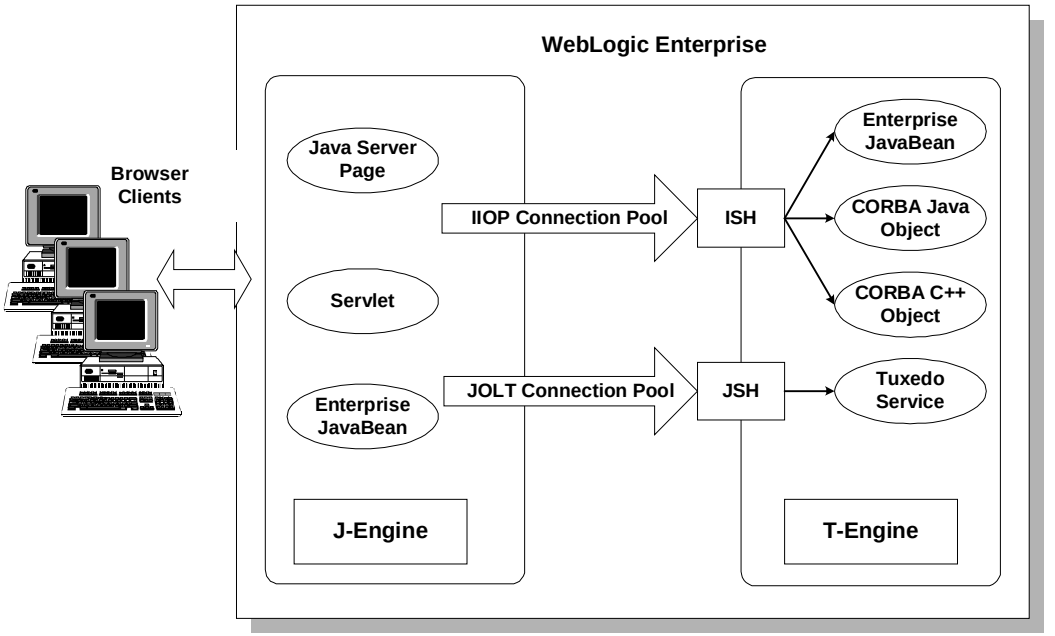
For more information about the WebLogic RMI-on-IIOP protocol, see *Using RMI in a WebLogic Enterprise Environment*.

# J-Engine and T-Engine Interoperability

The WebLogic Enterprise domain can comprise both T-Engine and J-Engine components. As of WebLogic Enterprise 5.1, this connectivity is available in only one direction -- from the J-Engine to the T-Engine -- via IIOP and Jolt connection pools.

Figure 1-3 shows how these connection pools allow components hosted by the J-Engine can invoke objects and services hosted by a T-Engine running in the same WebLogic Enterprise domain.

**Figure 1-3 J-Engine and T-Engine Interoperability**



Note the following about these connection pools:

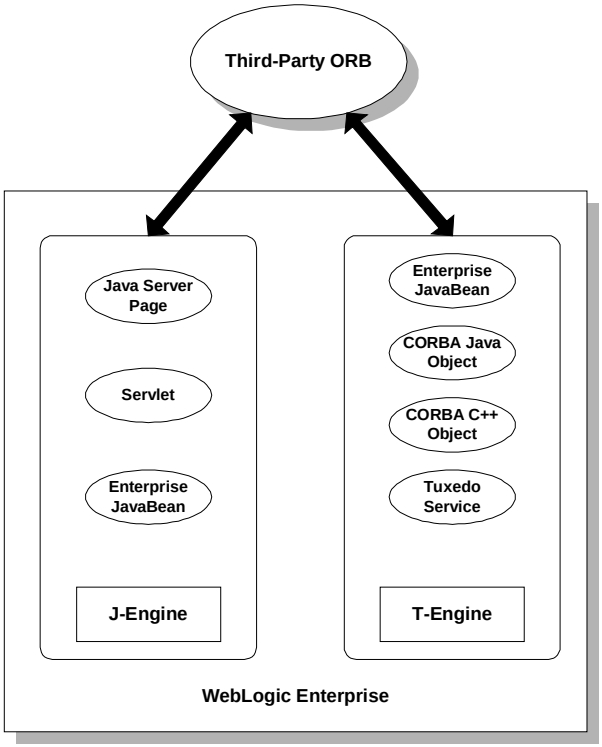
- IIOP connection pools allow J-Engine applications to invoke T-Engine EJBs and CORBA objects in a WebLogic Enterprise domain. For information about setting up and using IIOP connection pools, see *Using WebLogic Connectivity*.
- Jolt connection pools allow J-Engine applications to invoke Tuxedo services in a WebLogic Enterprise domain. For information about setting up and using Jolt connection pools, see *Configuring Jolt for WebLogic*.

For details about the versions of J-Engine and T-Engine server components that can interoperate, see the *Release Notes.*

# Third-party ORB Interoperability

CORBA applications based on third-party ORBs can interoperate with CORBA, Tuxedo, and J2EE server applications running in a WebLogic Enterprise domain provided that there is a correct match-up between IIOP protocols. Figure 1-4 provides a high-level view of third-party ORB interoperability with the WebLogic Enterprise domain.

**Figure 1-4 Third-party ORB Interoperability**




Note the following regarding third-party ORB interoperability with the WebLogic Enterprise domain:

- The WebLogic Enterprise C++ ORB supports the IIOP 1.2 protocol, and the WebLogic Enterprise Java ORB supports the IIOP 1.0 protocol. Both ORBs interoperate with client products from other vendors that support the IIOP 1.2, or earlier, protocol.
- WebLogic Enterprise provides transactional and security support for the following third-party client products. However, BEA does not provide environmental objects for these clients, so these products cannot directly access transactional and security capabilities inside the WebLogic Enterprise domain. These client products can connect to a WebLogic Enterprise server application using a stringified object reference.

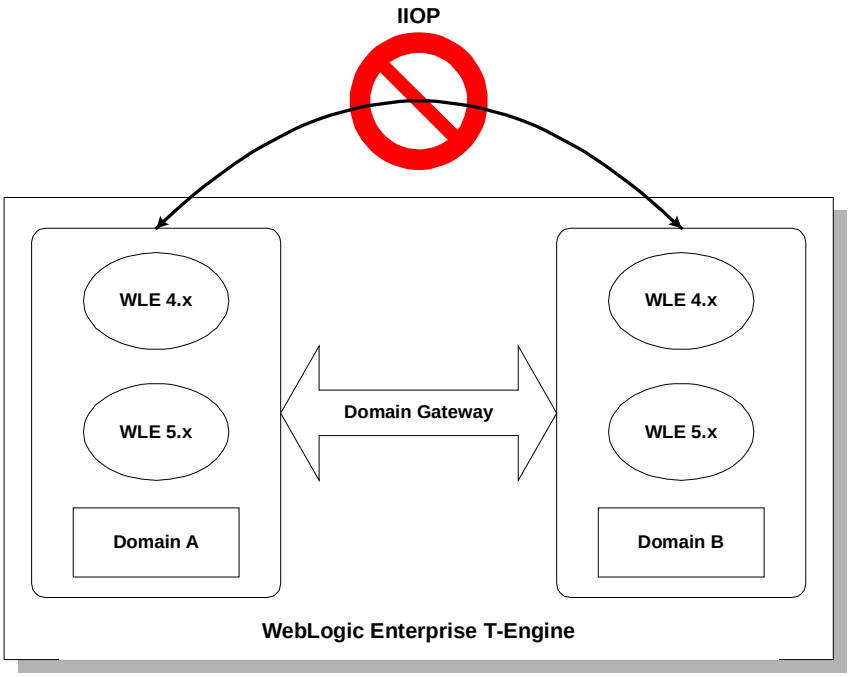
  - ActiveX
  - Netscape Communicator
  - Visibroker C++ version 3.3 (not clients using the Visibroker Java ORB)
  - Orbix 2.3c02 (with patch 26 or greater)

- CORBA applications that use a third-party ORB cannot initiate or coordinate a transaction propagated to the WebLogic Enterprise domain. These applications can invoke transactional objects running in the WebLogic Enterprise domain, and the WebLogic Enterprise transaction coordinator can manage those transactions; however, all the transactional management is fully delegated to the WebLogic Enterprise domain.
- If the CORBA application using the third-party ORB supports the Secure Sockets Layer (SSL), that application can use SSL mutual authentication as an alternative authentication mechanism.
- WebLogic Enterprise can call out to applications using third-party ORBs, using whatever callback mechanism is supported by the third-party ORB.
- WebLogic Enterprise client ORBs can interoperate with third-party ORBs (including SSL support).
- The WebLogic Enterprise J-Engine supports RMI over IIOP; therefore, J-Engine server applications can interoperate with third-party ORBs and other J2EE application servers that support the RMI over IIOP protocol. For information about restrictions or limitations on this interoperability, see the *Release Notes.*

# T-Engine Interdomain Interoperability

A server application running in one WebLogic Enterprise domain can interoperate with a server application running in another WebLogic Enterprise domain via the domain gateway (and *not* IIOP), as shown in Figure 1-5.

**Figure 1-5 T-Engine Interdomain Interoperability**




Domain gateways provide the following interoperability features:

- Domains can be heterogeneous with respect to the WebLogic Enterprise version. That is, a given WebLogic Enterprise can run both version 4.*x* and version 5.*x* WebLogic Enterprise applications; and those applications can invoke operations on either WebLogic Enterprise version 4.*x* or version 5.*x* applications running in a separate domain.
- Domain gateways fully support transaction propagation across domains. For example, a transactional object in one WebLogic Enterprise domain can include an object running in another domain in that transaction.
- Security context propagation is fully supported across domains for CORBA and EJB applications. (Interdomain security context propagation that span Tuxedo services running in WebLogic Enterprise domains is not supported.) However, note the following restriction:

When a client application authenticates itself to the domain, and invokes various services and objects in the domain -- which in turn may invoke other services and objects in the domain -- the client's security context is passed along with each operation. However, if in the course of satisfying a client request, a service in one domain makes an invocation on a service in a second domain, the client's security context cannot be passed to the second domain. The service in the second domain does not have knowledge of the original client.

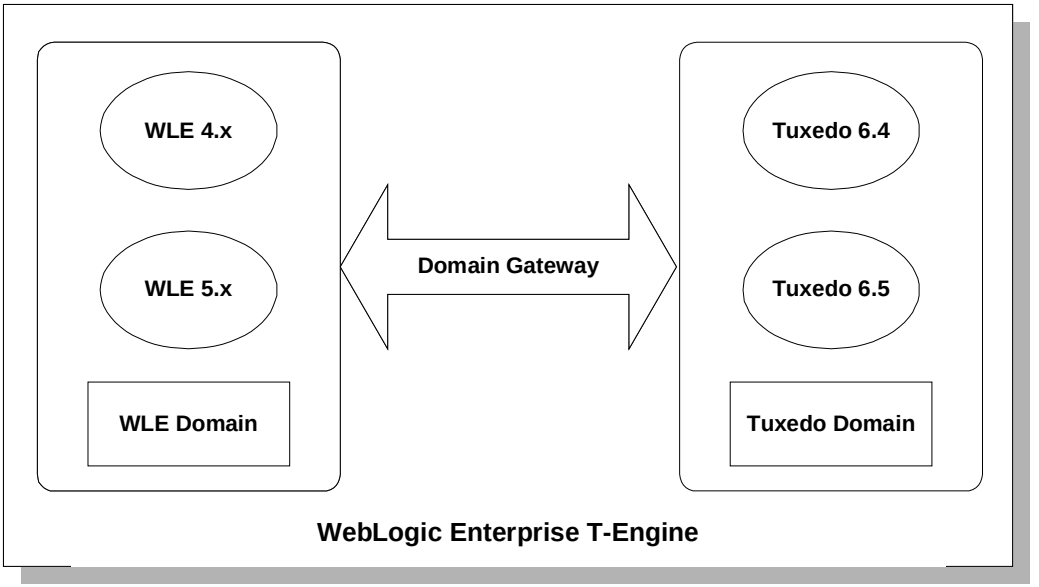
- You can secure all domain gateway communications with Link-Level Encryption (LLE).
- You can advertise factory objects and EJB home interfaces across domains.

For more information about interdomain WebLogic Enterprise interoperability, see *Administration* in the WebLogic Enterprise online documentation.

# WebLogic Enterprise and Tuxedo Domains Interoperability

WebLogic Enterprise and Tuxedo domains can interoperate via domain gateways, as shown in Figure 1-6.

**Figure 1-6 WebLogic Enterprise and Tuxedo Domains Interoperability**




Note the following about WebLogic Enterprise and Tuxedo interdomain interoperability:

- Transactions and security contexts can be fully propagated between WebLogic Enterprise and Tuxedo domains.
- Domain mixing is still not supported; that is, you cannot combine a WebLogic Enterprise domain and a Tuxedo domain into a single domain.

For more information about WebLogic Enterprise and Tuxedo domains interoperability, see *Administration* in the WebLogic Enterprise online documentation.

# Overview of the Interoperability Sample Applications

The WebLogic Enterprise software includes the sample applications as described in Table 1-1.

**Table 1-1 The Interoperability Sample Applications**

| Application | Description |
|---|---|
| EJB-to-CORBA/Java Simpapp | Shows an EJB server acting as a client invoking a request and receiving a response from a CORBA/Java object. |
| CORBA/C++-to-EJB Simpapp | Shows CORBA/C++ client invoking a request and receiving a response from an EJB server. |
| CORBA/Java-to-Tuxedo | Shows a CORBA/Java object that invokes a Tuxedo service using the Java Enterprise Tuxedo (JET) API. |
| EJB-to-Tuxedo | Shows an Enterprise JavaBean application that invokes a Tuxedo service using the Java Enterprise Tuxedo (JET) API. |

Use the interoperability sample applications in conjunction with the following documents:

- *Getting Started*
- *Guide to the University Sample Applications*
- *Guide to the Java Sample Applications*

# 2 EJB-to-CORBA/Java Simpapp Sample Application

The topic includes the following sections:

- How the EJB-to-CORBA/Java Simpapp Sample Application Works
- Software Prerequisites
- Building and Running the EJB-to-CORBA/Java Simpapp Sample Application
- Stopping the EJB-to-CORBA/Java Simpapp Sample Application

**Note:** Each sample application directory tree provided with the WebLogic Enterprise software includes a `Readme.txt` file that explains how to build and run the sample. Refer to this file in the following directory for troubleshooting information or other last-minute information about using the EJB-to-CORBA/Java sample application:

**Window NT**

```
$TUXDIR\samples\interop\ejb_corbaj
```

**UNIX**

```
$TUXDIR/samples/interop/ejb_corbaj
```

# How the EJB-to-CORBA/Java Simpapp Sample Application Works

The EJB-to-CORBA/Java Simpapp sample application has an EJB client, an EJB server deploying the `SimpBean` EJB and an EJB-to-CORBA bridge object, and a CORBA server deploying a CORBA object.

The `SimpBean` EJB has the following two remote methods:

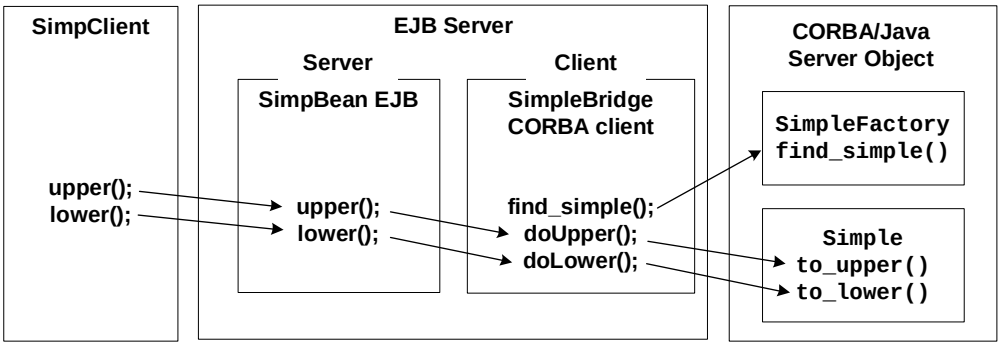
- The `upper` method delegates invocations to the `to_upper` method on the CORBA Simple object.
- The `lower` method method delegates invocations to the `to_lower` method on the CORBA Simple object.

The CORBA Simple object has the following two methods:

- The `to_upper` method accepts a string from the bridge object and converts the string to uppercase letters.
- The `to_lower` method accepts a string from the bridge object and converts the string to lowercase letters.

Figure 2-1 illustrates how the EJB-to-CORBA/Java Simpapp sample application works.

**Figure 2-1 EJB-to-CORBA/Java Simpapp Sample Application**



# Software Prerequisites

To run the `m3idltojava` compiler that is used by the EJB-to-CORBA/Java Simpapp sample application, you need to install Visual C++ version 6.0 with Service Pack 3 or later for Visual Studio. The `m3idltojava` compiler is installed by the WebLogic Enterprise software in the `bin` directory under `TUXDIR`.

# Implementing the Bridge Object to Invoke a CORBA/Java Object

The `SimpleBridge` Java object implements bridge design pattern. This object serves as a bridge between the `SimpBean` EJB and the CORBA/Java `Simple` object, and it is created by the `SimpBean` EJB.

The `SimpleBridge` Java object performs the following functions:

- Uses the Bootstrap object to obtain a reference to the WebLogic Enterprise FactoryFinder, from which the `SimpleBridge` object can obtain a reference to the `SimpleFactory` object.
- Invokes the `SimpleFactory` object to obtain a reference to the `Simple` object.
- Invokes the appropriate methods on the `Simple` object to satisfy the `SimpBean`'s requests.

Listing 2-1 shows the methods on the `SimpleBridge` object that delegate the `SimpBean` requests to the CORBA/Java `Simple` object.

**Listing 2-1 SimpleBridge Object Code**

```
public class SimpleBridge
{
  private Simple simple = null;

  public SimpleBridge ()
  {
    simple = getSimple();
  }
```

```
  public String doUpper(String mixedStr)
  {
    // Convert the string to upper case.
    org.omg.CORBA.StringHolder upperStr =
      new org.omg.CORBA.StringHolder(mixedStr);
    simple.to_upper(upperStr);

    System.out.println("in SimpleBridge.doUpper()");
    return upperStr.value;
  }


  public String doLower(String mixedStr)
  {
    // Convert the string to lower case.
    String lowerStr = simple.to_lower(mixedStr);

    System.out.println("in SimpleBridge.doLower()");
    return lowerStr;
  }


  public Simple getSimple()
  {
    try {
      // Obtain the bootstrap object,
     // the TOBJADDR property contains host and port to connect to.
      Tobj_Bootstrap bootstrap = TP.bootstrap();

      // 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.
      SimpleFactory simple_factory_ref =
        SimpleFactoryHelper.narrow(simple_fact_oref);

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

```
        // everything succeeded.
        return simple;
      }
      // catch the exceptions
      return null;
    }
  }
```

# The OMG IDL Code for the EJB-to-CORBA/Java Simpapp Interfaces

The sample application described in this chapter implements the CORBA interfaces listed in Table 2-1.

**Table 2-1 Sample Application IDL Interfaces**

| Interface | Description | Operation | Policies |
|---|---|---|---|
| `SimpleFactory` | Creates object references to the `Simple` object | `find_simple()` | Activation: `method`<br>Transaction: `optional` |
| `Simple` | Converts the case of a string | `to_upper()`<br>`to_lower()` | Activation: `method`<br>Transaction: `optional` |

Listing 2-2 shows the `simple.idl` file that defines the CORBA interfaces in the EJB-to-CORBA/Java Simpapp sample application.

**Listing 2-2 OMG IDL Code for the EJB-to-CORBA/Java Simpapp Sample Application**

```
#pragma prefix "beasys.com"

interface Simple
{
    //Convert a string to lower case (return a new string)
    string to_lower(in     string val);

    //Convert a string to upper case (in place)
    void to_upper(inout string val);
```

```
};

interface SimpleFactory
{
    Simple find_simple();
};
```

# Building and Running the EJB-to-CORBA/Java Simpapp Sample Application

To build and run the EJB-to-CORBA/Java Simpapp sample application, complete the following steps:

1. Verify the environment variables.
2. Copy the files for the EJB-to-CORBA/Java Simpapp sample application into a work directory.
3. Change the protection attribute on the files for the EJB-to-CORBA/Java Simpapp sample application.
4. Execute the `runme` command.

The following sections describe these steps, and also explain the following:

- How to run the EJB-to-CORBA/Java Simpapp sample application
- Processes and files created by the EJB-to-CORBA/Java Simpapp sample application

# Verifying the Settings of the Environment Variables

Before building and running the EJB-to-CORBA/Java Simpapp sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure they reflect correct information.

Table 2-2 lists the environment variables required to run the EJB-to-CORBA/Java Simpapp sample application.

**Table 2-2 Required Environment Variables for the EJB-to-CORBA/Java Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| TUXDIR | The directory path where you installed the WebLogic Enterprise software. For example:<br>**Windows NT**<br>`TUXDIR=c:\WLEdir`<br>**UNIX**<br>`TUXDIR=/usr/local/WLEdir` |
| JAVA_HOME | The directory path where you installed the JDK software. For example:<br>**Windows NT**<br>`JAVA_HOME=c:\JDK1.2.2`<br>**UNIX**<br>`JAVA_HOME=/usr/local/JDK1.2.2` |

You may optionally set the following system environment variables to change their default value prior to running the EJB-to-CORBA/Java Simpapp sample application `runme` command. See the *Administration Guide* for more information about selecting appropriate values for these environment variables.

Table 2-3 lists the optional environment variables required to run the EJB-to-CORBA/Java Simpapp sample application.

**Table 2-3 Optional Environment Variables for the EJB-to-CORBA/Java Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| HOST | The host name portion of the TCP/IP network address used by the ISL process to accept connections from CORBA. The default value is the name of the local machine. |
| PORT | The TCP port number at which the ISL process listens for incoming requests; it must be a number between 0 and 65535. The default value is 2468. |
| IPCKEY | The address of shared memory; the address must be a number greater than 32769 unique to this application on this system. The default value is 55432. |

## Verifying the Environment Variables

To verify that the information for the environment variables defined during installation is correct, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. Check the settings for TUXDIR and JAVA_HOME.

**UNIX**

1. Enter the `ksh` command to use the Korn shell.
2. Enter the `printenv` command to display the values of `TUXDIR` and `JAVA_HOME`, as in the following example:

```
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
```

## Changing the Environment Variables

To change the environment variable settings, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. On the Environment page in the System Properties window, click the environment variable you want to change or enter the name of the environment variable in the Variable field.
6. Enter the correct information for the environment variable in the Value field.
7. Click OK to save the changes.

**UNIX**

1. Enter the `ksh` command to use the Korn shell.
2. Enter the `export` command to set the correct values for the `TUXDIR` and `JAVA_HOME` environment variables, as in the following example:

```
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath
```

# Copying the Files for the Java Simpapp Sample Application into a Work Directory

You need to copy the files for the EJB-to-CORBA/Java Simpapp sample application into a work directory on your local machine. The files for the EJB-to-CORBA/Java Simpapp sample application are located in the following directories under TUXDIR:

**Windows NT**

```
$TUXDIR\samples\interop\ejb_corbaj
```

**UNIX**

```
$TUXDIR/samples/interop/ejb_corbaj
```

The following steps describe how to execute a makefile to copy all the example files into a work directory.

1. Create the work directory on your machine.
2. Copy the entire `ejb_corbaj` directory to the work directory created in the previous step:

   **Windows NT**

```
> copy $TUXDIR\samples\interop\ejb_corbaj\*.* <work_directory>
```

   **UNIX**

```
> cp -R $TUXDIR/samples/interop/ejb_corbaj/* <work_directory>
```

3. Change to the work directory created in step 1.
4. Enter the following command, which copies the remaining EJB-to-CORBA/Java Simpapp sample application files to the work directory:

   **Windows NT**

```
>nmake -f makefile.nt copy
```

   **UNIX**

```
>make -f makefile.mk copy
```

# Files in the Work Directory

This section lists and describes the files copied into your work directory after you have completed the steps described in the previous section.

The EJB-to-CORBA/Java Simpapp sample application files exist in the following sets:

- EJB Simpapp files
- CORBA/Java Simpapp files
- EJB-to-CORBA/Java utility files

## EJB Simpapp Files

Table 2-4 lists and describes the source files for the EJB portion of this sample application. These are the files that exist after you execute the `make` command. These files are copied into a subdirectory named `ejb`.

**Table 2-4 EJB Simpapp Files**

| File | Description |
|---|---|
| `ejb-jar.xml` | The standard deployment descriptor for the `SimpBean` class. |
| `weblogic-ejb-extensions.xml` | The XML file specifying the WebLogic EJB extensions to the deployment descriptor DTD. |
| `SimpClient.java` | The EJB Simpapp client. |
| `SimpBean.java` | The `SimpBean` class. This is an example of a stateless session bean. This bean contains the methods that invoke the `SimpleBridge` class to delegate the invocations on the `Simple` CORBA/Java object. |
| `Simp.java` | The Remote interface of the `SimpBean` class. |
| `SimpHome.java` | The Home interface of the `SimpBean` class. |

## CORBA/Java Simpapp files

Table 2-5 lists and describes the source files for the CORBA/Java portion of this sample application. They are copied into a subdirectory named `corbaj`.

**Table 2-5 CORBA/Java Simpapp Files**

| File | Description |
|---|---|
| `Simple.idl` | The OMG IDL that declares the `SimpleFactory` and `Simple` interfaces. |
| `Simple.xml` | The Server Description File for the `Simple` CORBA object. |
| `SimpleBridge.Java` | The EJB-to-CORBA/Java Simpapp `SimpleBridge` class. This class is used by the `SimpBean` class to communicate with the CORBA/Java `Simple` object. This is the class that effects the interoperability between the EJB and the CORBA/Java object. |
| `ServerImpl.Java` | The implementation of the `Server.initialize` and `Server.release` methods. |
| `SimpleFactoryImpl.Java` | The implementation of the `SimpleFactory` methods. |
| `SimpleImpl.Java` | The implementation of the `Simple` methods. |

## Utility Files

Table 2-6 lists and describes the utility files for this sample application.

**Table 2-6 EJB-to-CORBA/Java Utility Files**

| File | Description |
|---|---|
| `Readme.txt` | Contains directions for building and executing the EJB-to-CORBA/Java Simpapp sample application. |
| `runme.cmd` | The Windows NT batch file that contains commands to build and execute the EJB-to-CORBA/Java Simpapp sample application. |

**Table 2-6 EJB-to-CORBA/Java Utility Files (Continued)**

| File | Description |
|---|---|
| `runme.ksh` | The UNIX Korn shell script that contains commands to build and execute the EJB-to-CORBA/Java Simpapp sample application. |
| `makefile.nt` | The common makefile for the EJB-to-CORBA/Java Simpapp sample application on the Windows NT platform. This makefile can be used directly by the Visual C++ `nmake` command. The `makefile.nt` file is included by the `smakefile.nt` file. |
| `smakefile.nt` | The makefile for the EJB-to-CORBA/Java Simpapp sample application to be used by Symantec's Visual Café `smake` program. |
| `makefile.mk` | The makefile for the EJB-to-CORBA/Java Simpapp sample application on the UNIX platform. |

# Changing the Protection Attribute on the Files for the EJB-to-CORBA/Java Simpapp Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit or build the files in the EJB-to-CORBA/Java Simpapp sample application, you need to change the protection attribute of the files you copied into your work directory (including the respective `ejb` and `corbaj` subdirectories), as follows:

**Windows NT**

```
prompt>attrib /S -r drive:\workdirectory\*.*
```

**UNIX**

```
prompt>/bin/ksh

ksh prompt>chmod +w /workdirectory/*.*
```

On the UNIX operating system platform, you also need to change the permission of `runme.ksh` to give execute permission to the file, as follows:

```
ksh prompt>chmod +x runme.ksh
```

# Executing the runme Command

The `runme` command automates the following steps:

1. Sets the system environment variables
2. Loads the `UBBCONFIG` file
3. Compiles the code for the EJB server object
4. Compiles the code for the CORBA/Java server application
5. Starts the server application using the `tmboot` command
6. Starts the client application
7. Stops the server application using the `tmshutdown` command

To build and run the EJB-to-CORBA Simpapp sample application, enter the `runme` command, as follows:

**Windows NT**

```
prompt>cd workdirectory
prompt>runme
```

**UNIX**

```
ksh prompt>cd workdirectory
ksh prompt>./runme.ksh
```

The EJB-to-CORBA/Java Simpapp sample application runs and prints the following messages:

```
Testing simpapp
    cleaned up
    prepared
    built
    loaded ubb
    booted
    ran
    shutdown
```

```
  saved results
PASSED
```

All of the sample application output is placed in the `results` directory, which is located in the `ejb_corbaj` work directory. You can check in the `results` directory for the following files:

- The `log` file, for any compile, server boot, or server shutdown errors
- The `ULOG` file for server application errors and exceptions
- The `output` file for EJB client application output and exceptions

# Running the Sample Application

After you have executed the `runme` command, you can run the EJB-to-CORBA/Java Simpapp sample application manually if you like.

To manually run the EJB-to-CORBA/Java Simpapp sample application:

1. Verify that your environment variables are correct by entering the following command:

   **Windows NT**

   ```
   prompt>results\setenv
   ```

   **UNIX**

   ```
   prompt>. results/setenv.ksh
   ```

2. Run the sample, as follows:

   **Windows NT**

```
prompt>tmboot -y
prompt>java -classpath %CLIENTCLASSPATH% ejb.SimpClient corbaloc:%TOBJADDR%
```

   **UNIX**

```
prompt>tmboot -y
prompt>java -classpath ${CLIENTCLASSPATH} ejb.SimpClient corbaloc:${TOBJADDR}
```

3. The EJB-to-CORBA/Java Simpapp sample application prompts you to enter a string. After you enter the string, the application returns the string in uppercase and lowercase characters, respectively:

```
String?
Hello World
HELLO WORLD
hello world
```

All of the sample application output is placed in the `results` directory. You can check in that directory for the following files:

- The `.log` file, for any compile, server boot, or server shutdown errors
- The `ULOG` file for server application errors and exceptions
- The `output` file for EJB client application output and exceptions

# Processes and Files Generated by the EJB-to-CORBA/Java Simpapp Sample Application

This section lists and describes the processes started and the files generated by the EJB-to-CORBA/Java Simpapp sample application.

## Processes Started

When the `tmboot` command is executed to start the EJB-to-CORBA/Java Simpapp sample application, the server processes in Table 2-7 are started:

**Table 2-7 EJB-to-CORBA/Java Simpapp Server Processes**

| Process | Description |
|---|---|
| `TMSYSEVT` | The BEA Tuxedo system Event Broker. |

**Table 2-7 EJB-to-CORBA/Java Simpapp Server Processes (Continued)**

| Process | Description |
|---|---|
| TMFFNAME | Starts the following TMFFNAME processes:<br>■ The TMFFNAME server process with the -N option and the -M option is the MASTER NameManager service. The -N option says to start the NameManager Service; the -M option says to start this name manager as a Master. This service maintains a mapping of application-supplied names to object references.<br>■ The TMFFNAME server process with the -N option only is a SLAVE NameManager service.<br>■ The TMFFNAME server with the -F option contains the FactoryFinder object. |
| JavaServer | The JavaServer process that deploys the SimpBean EJB and hosts the implementation of the SimpBridge CORBA object. The JavaServer takes one argument, SimpleEjb.jar, which is the module for the SimpBean EJB. |
| JavaServer | The JavaServer process which deploys the Simple CORBA object (the deployment of this process also includes the SimpleFactory factory for the Simple object). The JavaServer takes one argument, SimpleCorba.jar, which is the module for the Simple CORBA object. |
| ISL | The IIOP Listener/Handler. |

## Files Generated in the corbaj Directory

Table 2-8 lists and describes the files that are generated in the corbaj work directory.

**Table 2-8 Files Generated in the corbaj Directory**

| File | Description |
|---|---|
| Simple.java | Generated by the m3idltojava command for the Simple interface. This interface contains the Java version of the IDL interface. It extends the org.omg.CORBA.Object class. |

**Table 2-8 Files Generated in the corbaj Directory (Continued)**

| File | Description |
|---|---|
| `SimpleHelper.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class provides auxiliary functionality, notably the narrow method. |
| `SimpleHolder.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class holds a public instance member of type `Simple`. It provides operations for `out` and `inout` arguments, which CORBA has, but which do not map easily to Java's semantics. |
| `_SimpleImplBase.java` | Generated by the `m3idltojava` command for the `Simple` interface. This abstract class is the server skeleton. It implements the `Simple.java` interface. The server class `SimpleImpl` extends `_SimpleImplBase`. |
| `_SimpleStub.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class is the client stub. It implements the `Simple.java` interface. |
| `SimpleFactory.java`<br>`SimpleFactoryHelper.java`<br>`SimpleFactoryHolder.java`<br>`_SimpleFactoryImplBase.java`<br>`_SimpleFactoryStub.java` | Generated by the `m3idltojava` command for the `SimpleFactory` interface. |
| `Simple.ser` | The server descriptor file that is generated by the `buildjavaserver` command. |
| `Simple.jar` | The Java ARchive (JAR) file that is generated by the `buildjavaserver` command. |

## Files Generated in the ejb_corbaj Directory

Table 2-9 lists and describes the files generated in the `ejb_corbaj` directory.

**Table 2-9 Files Generated in the ejb_corbaj Directory**

| File | Description |
|---|---|
| `results directory` | Generated by the `runme` comand. |
| `.adm/.keydb` | Generated by the `tmloadcf` command. Contains the security encryption key database. |

## Files Generated in the results Directory

Table 2-10 lists and describes the files that are generated in the `results` directory, which is a subdirectory of the `ejb_corbaj` work directory.

**Table 2-10 Files Generated in the results Directory**

| File | Description |
|---|---|
| `input` | Generated by the `runme` command. Contains the input that `runme` gives to the `SimpleClient` Java application. |
| `output` | Generated by the `runme` command. Contains the output that is produced when `runme` executes the `SimpleClient` Java application. |
| `expected_output` | Generated by the `runme` command. Contains the output that is expected when the `SimpleClient` Java application is executed by the `runme` command. The data in the output file is compared with the data in the `expected_output` file to determine whether the test passed or failed. |
| `log` | Generated by the `runme` command. Contains the output generated by the `runme` command. If the `runme` command fails, check this file and the `ULOG` file for errors. |

**Table 2-10 Files Generated in the results Directory (Continued)**

| File | Description |
|---|---|
| setenv.cmd | Generated by the Windows NT runme.cmd command. Contains the commands to set the environment variables needed to build and execute the EJB-to-CORBA/Java Simpapp sample application. |
| setenv.ksh | Generated by the UNIX runme.ksh command. Contains the commands to set the environment variables needed to build and execute the Simpapp sample. |
| stderr | Generated by the tmboot command, which is executed by the runme command. If the -noredirect server option is specified in the UBBCONFIG file, the System.err.println method sends the output to the stderr file instead of to the ULOG user log file. |
| stdout | Generated by the tmboot command, which is executed by the runme command. If the -noredirect server option is specified in the UBBCONFIG file, the System.out.println method sends the output to the stdout file instead of to the ULOG user log file. |
| tmsysevt.dat | Generated by the tmboot command, which is executed by the runme command. It contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. |
| tuxconfig | Generated by the tmloadcf command, which is executed by the runme command. |
| ubb | The UBBCONFIG file for the EJB-to-CORBA/Java Simpapp sample application. |
| ULOG.<*date*> | A log file that contains messages generated by the tmboot command. If there are any compile or run-time errors, check this file. |

# Stopping the EJB-to-CORBA/Java Simpapp Sample Application

Before using another sample application, use the following procedure to stop the EJB-to-CORBA/Java Simpapp sample application and to remove unnecessary files from the work directory.

1. To stop the application:

   **Windows NT**

   ```
   prompt>tmshutdown -y
   ```

   **UNIX**

   ```
   ksh prompt>tmshutdown -y
   ```

2. To restore the work directory to its original state:

   **Windows NT**

   ```
   prompt>nmake -f makefile.nt clean
   ```

   **UNIX**

   ```
   prompt>. ./results/setenv.ksh
   prompt>make -f makefile.nt clean
   ```

3. If Symantec's Visual Café is installed on your system, you can use the `smakefile.nt` file rather than the `makefile.nt` file, which is intended for use with the Visual C++ `nmake` program. For example, execute the following commands:

   ```
   prompt>results\setenv
   prompt>set JDKDIR=%JAVA_HOME%
   prompt>smake -f smakefile.nt
   ```

# 3 CORBA/C++-to-EJB Simpapp Sample Application

This topic includes the following sections:

- How the CORBA/C++-to-EJB Simpapp Sample Application Works
- Software Prerequisites
- The OMG IDL Code for the CORBA/C++-to-EJB Simpapp Interfaces
- Building and Running the CORBA/C++-to-EJB Simpapp Sample Application
- Stopping the CORBA/C++-to-EJB Simpapp Sample Application

**Note:** Each sample application directory tree provided with the WebLogic Enterprise software includes a `Readme.txt` file that explains how to build and run the sample. Refer to this file in the following directory for troubleshooting information or other last-minute information about using the CORBA/C++-to-EJB Simpapp sample application.

**Windows NT**

```
$TUXDIR\samples\interop\cpp_ejb
```

**UNIX**

```
$TUXDIR/samples/interop/cpp_ejb
```

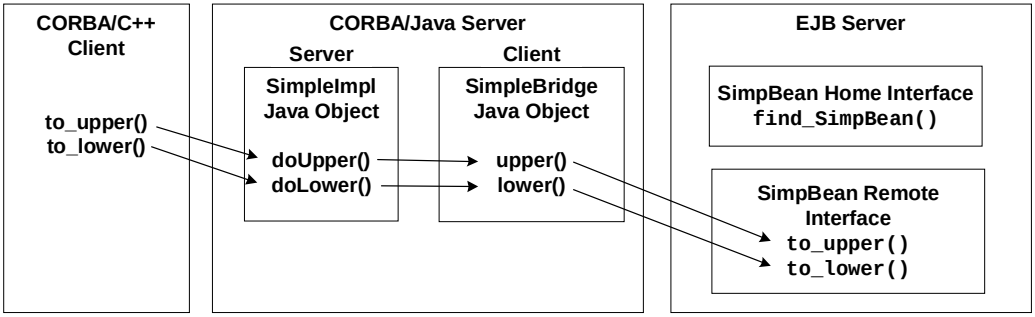
# How the CORBA/C++-to-EJB Simpapp Sample Application Works

The CORBA/C++-to-EJB Simpapp sample application features the following:

- A CORBA/C++ client application.
- A CORBA/Java server application acting as a liaison between the C++ client application and an EJB server. Contains the `SimpleImpl` object, and the `SimpleBridge` Java object.
- An EJB server that provides the following two operations:
  - One operation accepts a string from the client and converts the string to uppercase letters.
  - Another operation that accepts a string from the client and converts the string to lowercase letters.

Figure 3-1 illustrates how the CORBA/C++-to-EJB Simpapp sample application works.

**Figure 3-1 CORBA/C++-to-EJB Simpapp Sample Application**



# Software Prerequisites

To run the `m3idltojava` compiler that is used by the CORBA/C++-to-EJB Simpapp sample application, you need to install Visual C++ version 6.0 with Service Pack 3 or later for Visual Studio. The `m3idltojava` compiler is installed by the WebLogic Enterprise software in the `bin` directory under `TUXDIR`.

# Implementing the Bridge Object to Invoke an EJB

The `SimpleBridge` Java object serves as the intermediary between the CORBA/Java server and the EJB server application. The `SimpleBridge` Java object is created by the `SimpleImpl` Java object. The `SimpleBridge` Java object performs the following functions:

- Obtains the initial context for the EJB server application.
- Performs a lookup on the EJB Home interface.
- Invokes the appropriate methods on the `SimpBean` class to satisfy the client application requests.

Listing 3-1 shows the methods on the `SimpleBridge` object that delegate the `SimpleImpl` object's requests to the EJB server application:

**Listing 3-1 SimpleBridge Object Implementation Code**

```
public class SimpBridge {


  public String doUpper(String mixedStr)
  {
    String upperStr = "";
   javax.naming.Context ctx = null;
    SimpHome home = null;

    try {
      // create connection
      ctx = getContext();

      // look up home object
```

```
      home = (SimpHome) ctx.lookup("ejb.SimpHome");

      // create the object and use it
      Simp simp = home.create();
      upperStr = simp.upper(mixedStr);
    } // catch exceptions
    }

    return upperStr;
  }


  public String doLower(String mixedStr)
  {
    String lowerStr = "";
    javax.naming.Context ctx = null;
    SimpHome home = null;

    try {
      // create connection
      ctx = getContext();

      // look up home object
      home = (SimpHome) ctx.lookup("ejb.SimpHome");

      // create the object and use it
      Simp simp = home.create();
      lowerStr = simp.lower(mixedStr);
    } // catch exceptions
    }

    return lowerStr;
  }


  public static Context getContext()
  {
    Context context = null;

    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.beasys.jndi.WLEInitialContextFactory");
    env.put(Context.SECURITY_AUTHENTICATION, "none");

    try {
      context = new InitialContext(env);
    } catch (NamingException ee) {
      System.out.println("getContext failed: " + ee);
      ee.printStackTrace();
```

```
    }

    return context;
  }
}
```

# The OMG IDL Code for the CORBA/C++-to-EJB Simpapp Interfaces

The C++ and Java objects in the sample application described in this chapter implement the CORBA interfaces listed in Table 3-1.

**Table 3-1 Sample Application IDL Interfaces**

| Interface | Description | Operation | Policies |
|---|---|---|---|
| `SimpleFactory` | Creates object references to the `Simple` object. | `find_simple()` | Activation: `method`<br>Transaction: `optional` |
| `Simple` | Delegates the conversion of the string to the EJB server. | `to_upper()`<br>`to_lower()` | Activation: `method`<br>Transaction: `optional` |

Listing 3-2 shows the `simple.idl` file that defines the CORBA interfaces in the CORBA/C++-to-EJB Simpapp sample application.

**Listing 3-2 OMG IDL Code for the CORBA/C++-to-EJB Simpapp Sample Application**

```
#pragma prefix "beasys.com"

interface Simple
{
    //Convert a string to lower case (return a new string)
    string to_lower(in     string val);

    //Convert a string to upper case (in place)
```

```
    void to_upper(inout string val);
};

interface SimpleFactory
{
    Simple find_simple();
};
```

# Building and Running the CORBA/C++-to-EJB Simpapp Sample Application

To build and run the CORBA/C++-to-EJB Simpapp sample application, complete the following steps:

1. Verify the environment variables.
2. Copy the files for the CORBA/C++-to-EJB Simpapp sample application into a work directory.
3. Change the protection attribute on the files for the CORBA/C++-to-EJB Simpapp sample application.
4. Execute the `runme` command.

The following sections describe these steps, and also explain the following:

- How to run the CORBA/C++-to-EJB Simpapp sample application
- Processes and files generated by the CORBA/C++-to-EJB Simpapp sample application

# Verifying the Settings of the Environment Variables

Before building and running the CORBA/C++-to-EJB Simpapp sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure they reflect correct information.

Table 3-2 lists the environment variables required to run the CORBA/C++-to-EJB Simpapp sample application.

**Table 3-2 Required Environment Variables for the CORBA/C++-to-EJB Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| `TUXDIR` | The directory path where you installed the WebLogic Enterprise software. For example:<br>**Windows NT**<br>`TUXDIR=c:\WLEdir`<br>**UNIX**<br>`TUXDIR=/usr/local/WLEdir` |
| `JAVA_HOME` | The directory path where you installed the JDK software. For example:<br>**Windows NT**<br>`JAVA_HOME=c:\JDK1.2.2`<br>**UNIX**<br>`JAVA_HOME=/usr/local/JDK1.2.2` |

You may optionally set the following system environment variables to change their default value prior to running the CORBA/C++-to-EJB Simpapp sample `runme` command. See the *Administration Guide* for more information about selecting appropriate values for these environment variables.

Table 3-3 lists the optional environment variables you can assign prior to running the CORBA/C++-to-EJB Simpapp sample application.

**Table 3-3 Optional Environment Variables for the CORBA/C++-to-EJB Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| HOST | The host name portion of the TCP/IP network address used by the ISL process to accept connections from CORBA. The default value is the name of the local machine. |
| PORT | The TCP port number at which the ISL process listens for incoming requests; it must be a number between 0 and 65535. The default value is 2468. |
| IPCKEY | The address of shared memory; it must be a number greater than 32769 unique to this application on this system. The default value is 55432. |

## Verifying the Environment Variables

To verify that the information for the environment variables defined during installation is correct, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. Check the settings for TUXDIR and JAVA_HOME.

**UNIX**

1. Enter the ksh command to use the Korn shell.

2. Enter the `printenv` command to display the values of `TUXDIR` and `JAVA_HOME`, as in the following example:

```
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
```

## Changing the Environment Variables

To change the environment variable settings, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. On the Environment page in the System Properties window, click the environment variable you want to change or enter the name of the environment variable in the Variable field.
6. Enter the correct information for the environment variable in the Value field.
7. Click OK to save the changes.

**UNIX**

1. Enter the `ksh` command to use the Korn shell.
2. Enter the `export` command to set the correct values for the `TUXDIR` and `JAVA_HOME` environment variables, as in the following example:

```
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath
```

# Copying the Files for the CORBA/C++-to-EJB Simpapp Sample Application into a Work Directory

You need to copy the files for the CORBA/C++-to-EJB Simpapp sample application into a work directory on your local machine. The files for the CORBA/C++-to-EJB Simpapp sample application are located in the following directories.

**Windows NT**

```
$TUXDIR\samples\interop\cpp_ejb
```

**UNIX**

```
$TUXDIR/samples/interop/cpp_ejb
```

The following steps describe how to execute a makefile to copy all the example files into a work directory.

1. Create the work directory on your machine.
2. Copy the entire `cpp_ejb` directory to the work directory created in the previous step:

   **Windows NT**

```
> copy $TUXDIR\samples\interop\cpp_ejb\*.* <work_directory>
```

   **UNIX**

```
> cp -R $TUXDIR/samples/interop/cpp_ejb/* <work_directory>
```

3. Change to the work directory created in step 1.
4. Enter the following command, which copies the remaining EJB-to-CORBA/Java Simpapp sample application files to the work directory:

   **Windows NT**

```
>nmake -f makefile.nt copy
```

   **UNIX**

```
>make -f makefile.mk copy
```

# Files in the Work Directory

This section lists and describes the files copied into your work directory after you have completed the steps described in the previous section.

The CORBA/C++-to-EJB Simpapp sample application files exist in the following sets:

- CORBA C++ and Java source files
- EJB source files
- CORBA/C++-to-EJB Simpapp utility files

## CORBA/C++ Client Files

Table 3-4 lists and describes the files needed to create the CORBA/C++ client. Also included are the files needed to create the CORBA/Java server that acts as a bridge for the CORBA/C++-to-EJB Simpapp sample application. These files are located in the `cpp` subdirectory.

**Table 3-4 CORBA C++ and Java Files for the CORBA/C++-to-EJB Simpapp Sample Application**

| File | Description |
|---|---|
| `simplec.cpp` | C++ client program for the `simple` sample application. |
| `simple.idl` | The OMG IDL that declares the `SimpleFactory` and `Simple` interfaces. |
| `simple.xml` | The XML source file used to associate activation and transaction policy values with interfaces. |
| `ServerImpl.Java` | The Java source code that implements the `Server.initialize` and `Server.release` methods. |
| `SimpleFactoryImpl.Java` | The Java source code that implements the `SimpleFactory` methods. |
| `SimpleImpl.Java` | The Java source code that implements the `Simple` methods. |

## EJB Server Files

Table 3-5 lists and describes the files needed to create the EJB server for the CORBA/C++-to-EJB Simpapp sample application. These files are located in the `ejb` subdirectory.

**Table 3-5 EJB Source Files for the CORBA/C++-to-EJB Simpapp Sample Application**

| File | Description |
|---|---|
| `weblogic-ejb-extensions.XML` | The XML file specifying the WebLogic EJB extensions to the deployment descriptor DTD. |
| `SimpBean.java` | The Java source code for the `SimpBean` class. This is an example of a stateless session bean. This bean contains the methods invoked by the `SimpleBridge` class. |
| `Simp.java` | The Java source code for the Remote interface of the `SimpBean` class. |
| `SimpHome.java` | The Java source code for the Home interface of the `SimpBean` class. |
| `SimpleBridge.java` | The Java source code for the `SimpleBridge` class. This class is used by the `SimpleImpl` class to communicate with the EJB server. This is the class that effects the interoperability between the CORBA/C++ object and the EJB server. |

## Utility Files

Table 3-6 lists and describes the utility files for this sample application.

**Table 3-6 CORBA/C++-to-EJB Simpapp Utility Files**

| File | Description |
|---|---|
| `Readme.txt` | Contains directions for building and executing the CORBA/C++-to-EJB Simpapp sample application. |

**Table 3-6 CORBA/C++-to-EJB Simpapp Utility Files (Continued)**

| File | Description |
|---|---|
| `runme.cmd` | The Windows NT batch file that contains commands to build and execute the CORBA/C++-to-EJB Simpapp sample application. |
| `runme.ksh` | The UNIX Korn shell script that contains commands to build and execute the CORBA/C++-to-EJB Simpapp sample application. |
| `makefile.nt` | The common makefile for the CORBA/C++-to-EJB Simpapp sample application on the Windows NT platform. This makefile can be used directly by the Visual C++ `nmake` command. The `makefile.nt` file is included by the `smakefile.nt` file. |
| `smakefile.nt` | The makefile for the CORBA/C++-to-EJB Simpapp sample application to be used by Symantec's Visual Café `smake` program. |
| `makefile.mk` | The makefile for the CORBA/C++-to-EJB Simpappp sample application on the UNIX platform. |

# Changing the Protection Attribute on the Files for the CORBA/C++-to-EJB Simpapp Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit or build the files in the CORBA/C++-to-EJB Simpapp sample application, you need to change the protection attribute of the files you copied into your work directory (including the respective `ejb` and `corbaj` subdirectories), as follows:

**Windows NT**

```
prompt>attrib /S -r drive:\workdirectory\*.*
```

**UNIX**

```
prompt>/bin/ksh
ksh prompt>chmod +w /workdirectory/*.*
```

On the UNIX operating system platform, you also need to change the permission of `runme.ksh` to give execute permission to the file, as follows:

```
ksh prompt>chmod +x runme.ksh
```

# Executing the runme Command

The `runme` command automates the following steps:

1. Sets the system environment variables
2. Loads the `UBBCONFIG` file
3. Compiles the code for the EJB server object
4. Compiles the code for the CORBA/C++ joint client/server application
5. Compiles the code for the CORBA/Java server application
6. Starts the server application using the `tmboot` command
7. Starts the client application
8. Stops the server application using the `tmshutdown` command

To build and run the CORBA/Java Simpapp sample application, enter the `runme` command, as follows:

**Windows NT**

```
prompt>cd workdirectory

prompt>runme
```

**UNIX**

```
ksh prompt>cd workdirectory

ksh prompt>./runme.ksh
```

The CORBA/C++-to-EJB Simpapp sample application runs and prints the following messages:

```
Testing simpapp
    cleaned up
    prepared
```

```
    built
    loaded ubb
    booted
    ran
    shutdown
    saved results
  PASSED
```

All of the sample application output is placed in the `results` directory. You can check in that directory for the following files:

- The `.log` file, for any compile, server boot, or server shutdown errors
- The `ULOG` file for server application errors and exceptions
- The `output` file for EJB client application output and exceptions

# Running the Sample Application

After you have executed the `runme` command, you can run the CORBA/C++-to-EJB Simpapp sample application manually, if you like.

To manually run the CORBA/C++-to-EJB Simpapp sample application:

1. Verify that your environment variables are correct by entering the following command:

   **Windows NT**

   ```
   prompt>results\setenv
   ```

   **UNIX**

   ```
   prompt>. results/setenv.ksh
   ```

2. Run the sample:

   **Windows NT**

   ```
   prompt>tmboot -y
   prompt>java -DTOBJADDR=%TOBJADDR% SimpleClient
   ```

   **UNIX**

   ```
   prompt>tmboot -y
   prompt>java -DTOBJADDR=$TOBJADDR SimpleClient
   ```

3. To run the CORBA/C++ joint client/server application, enter a string. After you enter the string, the application returns the string in uppercase and lowercase characters, respectively:

```
String?
Hello World
HELLO WORLD
hello world
```

All of the sample application output is placed in the `results` directory. You can check in that directory for the following files:

- The `.log` file, for any compile, server boot, or server shutdown errors
- The `ULOG` file for server application errors and exceptions
- The `output` file for EJB client application output and exceptions

# Processes and Files Generated by the CORBA/C++-to-EJB Simpapp Sample Application

This section lists and describes the processes started and the files generated by the CORBA/C++-to-EJB Simpapp sample application.

## Processes Started

When the `tmboot` command is executed to start the CORBA/C++-to-EJB Simpapp sample application, the server processes in Table 3-7 are started:

**Table 3-7 CORBA/C++-to-EJB Simpapp Server Processes**

| Process | Description |
|---|---|
| `TMSYSEVT` | The BEA Tuxedo system Event Broker. |

**Table 3-7 CORBA/C++-to-EJB Simpapp Server Processes (Continued)**

| Process | Description |
|---|---|
| TMFFNAME | Starts the following TMFFNAME processes:<br>■ The TMFFNAME server process with the -N option and the -M option is the MASTER NameManager service. The -N option says to start the NameManager Service; the -M option says to start this name manager as a Master. This service maintains a mapping of application-supplied names to object references.<br>■ The TMFFNAME server process with the -N option only is a SLAVE NameManager service.<br>■ The TMFFNAME server with the -F option contains the FactoryFinder object. |
| JavaServer | The Simpapp server process that implements EJB JAR file for the SimpBean and SimpHome interfaces. The JavaServer has one argument, SimpleEjb.jar, which is the EJB Java ARchive (JAR) file that was created for the application. |
| JavaServer | The Simpapp server process that implements the SimpleFactory interface and the Simple interface. The JavaServer has one argument, SimpleCorba.jar, which is the CORBA Java ARchive (JAR) file that was created for the application. |
| ISL | The IIOP Listener/Handler. |

## Files Generated in the cpp Directory

Table 3-8 lists and describes the files generated in the cpp directory.

**Table 3-8 Files Generated in the cpp Directory**

| File | Description |
|---|---|
| Simple_c.cpp | Client stubs for the Simple and SimpleFactory interfaces. |
| Simple_c.h | Client stub header for the Simple and SimpleFactory interfaces. |

**Table 3-8 Files Generated in the cpp Directory (Continued)**

| File | Description |
|---|---|
| `Simple_client.exe` | C++ client executable. |
| `Simple.java` | Generated by the `m3idltojava` command for the `Simple` interface. This interface contains the Java version of the IDL interface. It extends the base class `org.omg.CORBA.Object`. |
| `SimpleHelper.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class provides auxiliary functionality, notably the `narrow` method. |
| `SimpleHolder.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class holds a public instance member of type `Simple`. It provides operations for `out` and `inout` arguments, which CORBA has, but which do not map easily to Java's semantics. |
| `_SimpleImplBase.java` | Generated by the `m3idltojava` command for the `Simple` interface. This abstract class is the server skeleton. It implements the `Simple.java` interface. The server class `SimpleImpl` extends `_SimpleImplBase`. |
| `_SimpleStub.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class is the client stub. It implements the `Simple.java` interface. |
| `SimpleFactory.java`<br>`SimpleFactoryHelper.java`<br>`SimpleFactoryHolder.java`<br>`_SimpleFactoryImplBase.java`<br>`_SimpleFactoryStub.java` | Generated by the `m3idltojava` command for the `SimpleFactory` interface. |
| `Simple.ser` | The server descriptor file that is generated by the `buildjavaserver` command. |
| `Simple.jar` | The Java ARchive (JAR) file that is generated by the `buildjavaserver` command. |

## File Generated in the cpp_ejb Directory

Table 3-9lists and described the files generated in the `cpp_ejb` directory.

**Table 3-9 Files Generated in the cpp_ejb Directory**

| File | Description |
|---|---|
| `results` directory | Generated by the `runme` command. |
| `.adm/.keydb` | Generated by the `tmloadcf` command. Contains the security encryption key database. |

## Files Generated in the results Directory

Table 3-10 lists and describes the files that are generated in the `results` directory, which is a subdirectory of the `corbaj` work directory.

**Table 3-10 Files Generated in the results Directory**

| File | Description |
|---|---|
| `input` | Generated by the `runme` command. Contains the input that `runme` gives to the `SimpleClient` Java application. |
| `output` | Generated by the `runme` command. Contains the output that is produced when `runme` executes the `SimpleClient` Java application. |
| `expected_output` | Generated by the `runme` command. Contains the output that is expected when the `SimpleClient` Java application is executed by the `runme` command. The data in the `output` file is compared with the data in the `expected_output` file to determine whether the test passed or failed. |
| `log` | Generated by the `runme` command. Contains the output generated by the `runme` command. If the `runme` command fails, check this file, and the `ULOG` file, for errors. |

**Table 3-10 Files Generated in the results Directory (Continued)**

| File | Description |
|---|---|
| setenv.cmd | Generated by the Windows NT runme.cmd command. Contains the commands to set the environment variables needed to build and execute the CORBA/C++-to-EJB Simpapp sample application. |
| setenv.ksh | Generated by the UNIX runme.ksh command. Contains the commands to set the environment variables needed to build and execute the Simpapp sample application. |
| stderr | Generated by the tmboot command, which is executed by the runme command. If the -noredirect server option is specified in the UBBCONFIG file, the System.err.println method sends the output to stderr instead of to the ULOG user log file. |
| stdout | Generated by the tmboot command, which is executed by the runme command. If the -noredirect server option is specified in the UBBCONFIG file, the System.out.println method sends the output to the stdout file instead of to the ULOG user log file. |
| tmsysevt.dat | Generated by the tmboot command, which is executed by the runme command. It contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. |
| tuxconfig | Generated by the tmloadcf command, which is executed by the runme command. |
| ubb | The UBBCONFIG file for the CORBA/C++-to-EJB Simpapp sample application. |
| ULOG.<*date*> | A log file that contains messages generated by the tmboot command. |

# Stopping the CORBA/C++-to-EJB Simpapp Sample Application

Before using another sample application, use the following procedure to stop the CORBA/C++-to-EJB Simpapp sample application and to remove unnecessary files from the work directory:

1. To stop the application:

   **Windows NT**

   ```
   prompt>tmshutdown -y
   ```

   **UNIX**

   ```
   ksh prompt>tmshutdown -y
   ```

2. To restore the work directory to its original state:

   **Windows NT**

   ```
   prompt>nmake -f makefile.nt clean
   ```

   **UNIX**

   ```
   prompt>. ./results/setenv.ksh
   prompt>make -f makefile.nt clean
   ```

3. If Symantec's Visual Café is installed on your system, you can use the `smakefile.nt` file rather than the `makefile.nt` file, which is intended for use with the Visual C++ `nmake` program. For example, execute the following commands:

   ```
   prompt>results\setenv
   prompt>set JDKDIR=%JAVA_HOME%
   prompt>smake -f smakefile.nt
   ```

# 4 CORBA/Java-to-Tuxedo Simpapp Sample Application

This topic includes the following sections:

- How the CORBA/Java-to-Tuxedo Simpapp Sample Application Works
- Building and Running the CORBA/Java-to-Tuxedo Simpapp Sample Application
- Stopping the CORBA/Java-to-Tuxedo Simpapp Sample Application

**Note:** Each sample application directory tree provided with the WebLogic Enterprise software includes a `Readme.txt` file that explains how to build and run the sample. Refer to this file in the following directory for troubleshooting information or other last-minute information about using the CORBA/Java-to-Tuxedo sample application:

**Window NT**

```
$TUXDIR\samples\interop\corbaj_tux
```

**UNIX**

```
$TUXDIR/samples/interop/corbaj_tux
```

# How the CORBA/Java-to-Tuxedo Simpapp Sample Application Works

This topic includes the following sections:

- Key Application Components
- Application Flow
- OMG IDL Code for the CORBA/Java-to-Tuxedo Simpapp Interfaces
- Software Prerequisites
- Example Code

The CORBA/Java-to-Tuxedo Simpapp sample application demonstrates the use of Java Enterprise Tuxedo (JET) technology to invoke a Tuxedo service from a CORBA/Java server running in the WebLogic Enterprise domain. For more information about JET, see *Using Java Enterprise Tuxedo*.

## Key Application Components

The CORBA/Java-to-Tuxedo Simpapp sample application consists of the following components:

- A CORBA/Java client (`SimpleClient`) prompts the user for a string and then invokes methods on the CORBA/Java server, `to_upper` and `to_lower`, to convert the string to all uppercase and all lowercase text, respectively. This client then displays the results of the conversion to the user.
- A CORBA/Java server object (`SimpleImpl`) acts as the bridge between the WebLogic Enterprise and Tuxedo environments. This object provides the following methods to handle the conversion:
  - The `to_upper` method calls another method, `joltNativeCall`, which instantiates a `JoltService` object and uses the `call` method to invoke the TOUPPER service in the Tuxedo environment. The `JoltService` class is a component of the JET Class Library.

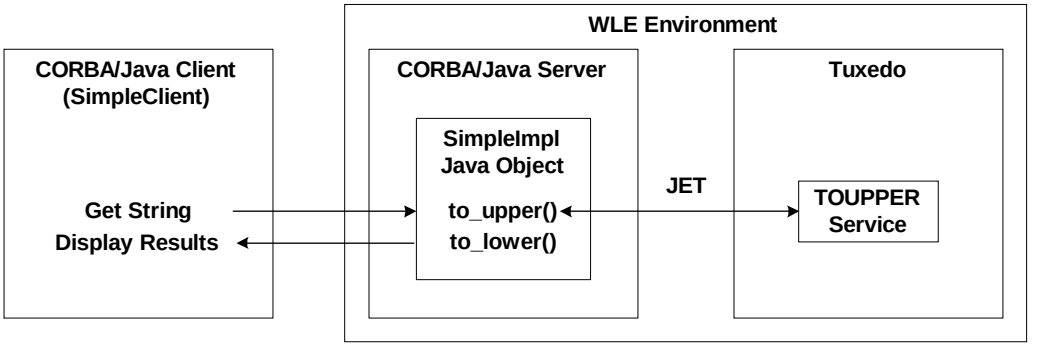
  - The `to_lower` method uses the Java `toLowerCase` method to perform the lowercase conversion.

- The TOUPPER service in the Tuxedo environment, which converts a text string to all uppercase characters.

# Application Flow

Figure 4-1 illustrates how the CORBA/Java-to-Tuxedo Simpapp sample application works.

**Figure 4-1 Overview of CORBA/Java-to-Tuxedo Simpapp Sample Application**




# OMG IDL Code for the CORBA/Java-to-Tuxedo Simpapp Interfaces

Table 4-1 describes the CORBA interfaces that are implemented in the CORBA/Java-to-Tuxedo Simpapp application.

**Table 4-1 CORBA Interfaces Implemented in CORBA/Java-to-Tuxedo Simpapp Application**

| Interface | Description | Operation | Policies |
|---|---|---|---|
| `SimpleFactory` | Creates object references to the Simple object | `find_simple()` | Activation: `method`<br>Transaction: `optional` |
| `Simple` | Converts the case of a string | `to_upper()`<br>`to_lower()` | Activation: `method`<br>Transaction: `optional` |

Listing 4-1 shows the `simple.idl` file that defines the CORBA interfaces in the CORBA/Java-to-Tuxedo Simpapp sample application.

**Listing 4-1 OMG IDL Code for the CORBA/Java-to-Tuxedo Simpapp Sample Application**

```
#pragma prefix "beasys.com"
interface Simple
{
    // convert a string to lower case (return a new string)
    string to_lower(in    string val);

    // convert a string to upper case (in place)
    void   to_upper(inout string val);
};

interface SimpleFactory
{
    Simple find_simple();

   // To make simpapp scalable have the SimpleFactory use some means
   // to identify (specify in criteria) the user in the Simple object
    // reference it creates. eg. Name (string), SS# (unsigned long),
     // tel_no (string).
};
```

# Software Prerequisites

To run the `m3idltojava` compiler that is used by the CORBA/Java-to-Tuxedo Simpapp sample application, you need to install Visual C++ version 6.0 with Service Pack 3 or later for Visual Studio. The `m3idltojava` compiler is installed by the WebLogic Enterprise software in the `bin` directory under `TUXDIR`.

# Example Code

Listing 4-2 shows the `joltNativeCall` method from the SimpleImpl CORBA/Java server in the CORBA/Java-to-Tuxedo Simpapp sample application. This method uses JET technology to invoke the TOUPPER service in the Tuxedo environment. It accepts the following parameters:

- `svcName` is the name of the Tuxedo service to invoke (TOUPPER).
- `data` is the string that the user entered in the SimpleClient CORBA/Java client.

**Listing 4-2 The joltNativeCall() Method in the SimpleImpl CORBA/Java Server**

```
 String joltNativeCall (String svcName, org.omg.CORBA.StringHolder
data)
  {
    JoltService svc;

    try {
      svc = new JoltService (svcName);
      svc.addString("STRING", data.value);
      svc.call (null);
    } catch (ServiceException ee) {
      System.out.println("JoltService got " + ee);
      return new String("");
    }

    return svc.getStringDef("STRING", "no_response");
  }
}
```

The `joltNativeCall()` method performs the following operations:

- Instantiating a new `JoltService` object to represent the Tuxedo TOUPPER service. The `JoltService` class is part of the JET Class Library. For more information about the JET Class Library, see *Using Java Enterprise Tuxedo*.
- Calling the `addString` method on the `JoltService` object to set up the input parameters to the TOUPPER service: the conversion type (`"STRING"`) and the string to convert (`data`).
- Calling the `call` method to invoke the TOUPPER service.
- Returning the results to the calling method, `to_upper`, in the SimpleImpl CORBA/Java server object.

# Building and Running the CORBA/Java-to-Tuxedo Simpapp Sample Application

To build and run the CORBA/Java-to-Tuxedo Simpapp sample application, complete the following steps:

- Step 1: Verify the Settings of Environment Variables
- Step 2: Copy the Files into a Work Directory
- Step 3: Change the Protection Attribute on the Files
- Step 4: Run the CORBA/Java-to-Tuxedo Simpapp Sample Application

# Step 1: Verify the Settings of Environment Variables

Before building and running the CORBA/Java-to-Tuxedo Simpapp sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure that they reflect correct information.

## Required Environment Variables

Table 4-2 describes the environment variables that are required to run the CORBA/Java-to-Tuxedo Simpapp sample application.

**Table 4-2 Required Environment Variables for the CORBA/Java-to-Tuxedo Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| `TUXDIR` | The directory path where you installed the WebLogic Enterprise software. For example:<br>**Windows NT**<br>`TUXDIR=c:\WLEdir`<br>**UNIX**<br>`TUXDIR=/usr/local/WLEdir` |
| `JAVA_HOME` | The directory path where you installed the JDK software. For example:<br>**Windows NT**<br>`JAVA_HOME=c:\JDK1.2.2`<br>**UNIX**<br>`JAVA_HOME=/usr/local/JDK1.2.2` |

## Optional Environment Variables

You may optionally set the following system environment variables to change their default value before running the CORBA/Java-to-Tuxedo Simpapp sample application `runme` command. See the *Administration Guide* for more information about selecting appropriate values for these environment variables.

Table 4-3 describes the optional environment variables that you can set before running the CORBA/Java-to-Tuxedo Simpapp sample application.

**Table 4-3 Optional Environment Variables for the CORBA/Java-to-Tuxedo Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| HOST | The host name portion of the TCP/IP network address used by the ISL process to accept connections from CORBA. The default value is the name of the local machine. |
| PORT | The TCP port number at which the ISL process listens for incoming requests. It must be a number between 0 and 65535. The default value is 2468. |
| IPCKEY | The address of shared memory. The address must be a number greater than 32769 that is unique to this application on this system. The default value is 55432. |

## Verifying the Environment Variables

To verify that the information for the environment variables defined during installation is correct, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. Check the settings for TUXDIR and JAVA_HOME.

**UNIX**

1. Execute the `ksh` command to use the Korn shell.
2. Execute the `printenv` command to display the values of `TUXDIR` and `JAVA_HOME`, as shown in the following example:

```
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
```

## Changing the Environment Variables

To change the environment variable settings, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. On the Environment page in the System Properties window, click the environment variable you want to change, or enter the name of the environment variable in the Variable field.
6. Enter the correct information for the environment variable in the Value field.
7. Click OK to save the changes.

**UNIX**

1. Execute the `ksh` command to use the Korn shell.
2. Execute the `export` command to set the correct values for the `TUXDIR` and `JAVA_HOME` environment variables, as in the following example:

```
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath
```

# Step 2: Copy the Files into a Work Directory

You need to copy the files for the CORBA/Java-to-Tuxedo Simpapp sample application into a work directory on your local machine. The files for the CORBA/Java-to-Tuxedo Simpapp sample application are located in the following directories under `TUXDIR`:

**Windows NT**

```
$TUXDIR\samples\interop\corbaj_tux
```

**UNIX**

```
$TUXDIR/samples/interop/corbaj_tux
```

## Copying the Files

The following steps describe how to execute a makefile to copy all of the example files into a work directory.

1. Create the work directory on your machine.
2. Copy the entire `corbaj_tux` directory to the working directory created in the previous step:

   **Windows NT**

   ```
   > copy $TUXDIR\samples\interop\corbaj_tux\*.* <work_directory>
   ```

   **UNIX**

   ```
   > cp -R $TUXDIR/samples/interop/corbaj_tux/* <work_directory>
   ```

3. Change to the working directory created in step 1.
4. For UNIX, start a `ksh`.
5. Change the permissions on all the files to give them read-access.
6. Verify that the following command is in your search path:

   **Windows NT**

   ```
   nmake
   ```

   **UNIX**

   ```
   make
   ```

7. Execute the following command, which copies the remaining CORBA/Java-to-Tuxedo Simpapp sample application files to the working directory:

   **Windows NT**

   ```
   >nmake -f makefile.nt copy
   ```

   **UNIX**

   ```
   >make -f makefile.mk copy
   ```

## Files Copied to the Working Directory

This section describes the directories and files that were copied into your working directory when you executed the makefile.

### Utility Files

Table 4-4 describes the utility files for this sample application. These files reside in the root of the working directory.

**Table 4-4 Utility Files in the Root of the Working Directory**

| File | Description |
|---|---|
| `Readme.txt` | Contains directions for building and executing the CORBA/Java-to-Tuxedo Simpapp sample application. |
| `runme.cmd` | The Windows NT batch file that contains commands to build and execute the CORBA/Java-to-Tuxedo Simpapp sample application. |
| `runme.ksh` | The UNIX Korn shell script that contains commands to build and execute the CORBA/Java-to-Tuxedo Simpapp sample application. |
| `makefile.nt` | The common makefile for the CORBA/Java-to-Tuxedo Simpapp sample application on the Windows NT platform. This makefile can be used directly by the Visual C++ `nmake` command. |
| `makefile.mk` | The makefile for the CORBA/Java-to-Tuxedo Simpapp sample application on the UNIX platform. |

## CORBA/Java Client and Server Files

Table 4-5 describes the source files for the CORBA/Java client and server portions of this sample application. These files reside in the `corbaj` subdirectory of the working directory.

**Table 4-5 CORBA/Java Client and Server Files in the corbaj Subdirectory**

| File | Description |
|---|---|
| `SimpleClient.java` | CORBA/Java client application. |
| `Simple.idl` | The OMG IDL that declares the `SimpleFactory` and `Simple` interfaces. |
| `Simple.xml` | The Server Description File for the `Simple` CORBA/Java server object. |
| `SimpleFactoryImpl.Java` | The implementation of the `SimpleFactory` methods. |
| `SimpleImpl.Java` | The implementation of the `Simple` methods. Illustrates the interoperability between the CORBA/Java server and the Tuxedo server by providing the bridge between them. |
| `ServerImpl.Java` | The implementation of the `Server.initialize` and `Server.release` methods. |

## Tuxedo Files

Table 4-6 describes the source files for the Tuxedo portion of this sample application. These files reside in the `tux` subdirectory of the working directory.

**Table 4-6 Tuxedo Files in the tux Subdirectory**

| File | Description |
|---|---|
| `jrepository` | Jolt Repository definition file. |
| `simpserv.c` | Simpapp server source code. |

# Step 3: Change the Protection Attribute on the Files

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit or build the files in the CORBA/Java-to-Tuxedo Simpapp sample application, you need to change the protection attribute of the files you copied into your work directory (including the respective `ejb` and `corbaj` subdirectories), as follows:

**Windows NT**

```
prompt>attrib /S -r drive:\workdirectory\*.*
```

**UNIX**

```
prompt>/bin/ksh

ksh prompt>chmod +w /workdirectory/*.*
```

On UNIX, you also need to change the permission of `runme.ksh` to give execute permission to the file, as follows:

```
ksh prompt>chmod +x runme.ksh
```

# Step 4: Run the CORBA/Java-to-Tuxedo Simpapp Sample Application

Once you have copied the files and changed their protection attributes, you can build and run the CORBA/Java-to-Tuxedo Simpapp sample application by executing the `runme` command, which starts server processes, generates files in various subdirectories of the working directory, and starts the sample application.

## Executing the runme Command

The `runme` command automates the following steps:

1. Set the system environment variables.
2. Load the `UBBCONFIG` file.
3. Compile the code for the CORBA/Java Server application.

4. Compile the code for the Jet Server application.
5. Start the server application using the `tmboot` command.
6. Start the client application.
7. Stop the server application using the `tmshutdown` command.

To build and run the EJB-to-CORBA Simpapp sample application, execute the `runme` command, as follows:

**Windows NT**

```
prompt>cd workdirectory

prompt>runme
```

**UNIX**

```
ksh prompt>cd workdirectory

ksh prompt>./runme.ksh
```

The CORBA/Java-to-Tuxedo Simpapp sample application runs and prints the following messages:

```
Testing simpapp
    cleaned up
    prepared
    built
    loaded ubb
    booted
    ran
    shutdown
    saved results
  PASSED
```

All of the sample application output is placed in the `results` directory, which is located in the `corbaj_tux` working directory. You can check in the `results` directory for the following files:

- The `log` file, for any compile, server boot, or server shutdown errors.
- The `ULOG` file for server application errors and exceptions.
- The `output` file for CORBA/Java client application output and exceptions.

## Running the Sample Application Manually

After you have executed the `runme` command one time, you can subsequently run the CORBA/Java-to-Tuxedo Simpapp sample application manually.

To run the CORBA/Java-to-Tuxedo Simpapp sample application manually:

1. Verify that your environment variables are correct by entering the following command:

   **Windows NT**

   ```
   prompt>results\setenv
   ```

   **UNIX**

   ```
   prompt>. results/setenv.ksh
   ```

2. Run the sample, as follows:

   **Windows NT**

   ```
   prompt>tmboot -y
   prompt>java -DTOBJADDR=%TOBJADDR% -classpath %CLIENTCLASSPATH%
   SimpleClient
   ```

   **UNIX**

   ```
   prompt>tmboot -y
   prompt>java -DTOBJADDR=${TOBJADDR} -classpath
   ${CLIENTCLASSPATH} SimpleClient
   ```

3. The CORBA/Java-to-Tuxedo Simpapp sample application prompts you to enter a string. After you enter the string, the application returns the string in uppercase and lowercase characters, respectively:

   ```
   String?
   Hello World
   HELLO WORLD
   hello world
   ```

All of the sample application output is placed in the `results` directory. You can check in that directory for the following files:

- The `.log` file, for any compile, server boot, or server shutdown errors.
- The `ULOG` file for server application errors and exceptions.
- The `output` file for CORBA/Java client application output and exceptions.

## Server Processes Started by the Sample Application

Table 4-7 describes the server processes that are started when the `tmboot` command is executed to start the CORBA/Java-to-Tuxedo Simpapp sample application.

**Table 4-7 Server Processes Started When tmboot Is Executed**

| Process | Description |
|---|---|
| TMSYSEVT | BEA Tuxedo system Event Broker. |
| TMFFNAME | Starts the following TMFFNAME processes:<br>■ The TMFFNAME server process with the `-N` option and the `-M` option is the MASTER `NameManager` service. The `-N` option says to start the NameManager Service. The `-M` option says to start this name manager as a Master. This service maintains a mapping of application-supplied names to object references.<br>■ The TMFFNAME server process with the `-N` option only is a SLAVE `NameManager` service.<br>■ The TMFFNAME server with the `-F` option contains the `FactoryFinder` object. |
| JavaServer | JavaServer process that deploys the `Simple` CORBA/Java object (the deployment of this process also includes the `SimpleFactory` factory for the `Simple` object). The JavaServer takes one argument, `Simple.jar`, which is the module for the `Simple` CORBA/Java server object. |
| JREPSVR | Jolt Repository Server, which manages the Jolt Repository. The Jolt Repository contains service definitions for BEA Tuxedo services. |
| ISL | IIOP Listener/Handler. |

## Files Generated by the Sample Application

This section describes the files that are generated in various subdirectories of the working directory.

## File Generated in the .adm Subdirectory

Table 4-8 describes the file that is generated in the `.adm` subdirectory.

**Table 4-8 File in the .adm Subdirectory**

| File | Description |
|---|---|
| `.keydb` | Generated by the `tmloadcf` command. Contains the security encryption key database. |

## Files Generated in the corbaj Subdirectory

Table 4-9 describes the files that are generated in the `corbaj` subdirectory.

**Table 4-9 Files Generated in the corbaj Subdirectory**

| File | Description |
|---|---|
| `Simple.java` | Generated by the `m3idltojava` command for the `Simple` interface. This interface contains the Java version of the IDL interface. It extends the `org.omg.CORBA.Object` class. |
| `SimpleHelper.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class provides auxiliary functionality, notably the `narrow` method. |
| `SimpleHolder.java` | Generated by the `m3idltojava` command for the `Simple` interface. This class holds a public instance member of type `Simple`. It provides operations for `out` and `inout` arguments, which CORBA has, but which do not map easily to Java's semantics. |
| `_SimpleImplBase.java` | Generated by the `m3idltojava` command for the `Simple` interface. This abstract class is the server skeleton. It implements the `Simple.java` interface. The server class `SimpleImpl` extends `_SimpleImplBase`. |

**Table 4-9 Files Generated in the corbaj Subdirectory (Continued)**

| File | Description |
|---|---|
| _SimpleStub.java | Generated by the m3idltojava command for the Simple interface. This class is the client stub. It implements the Simple.java interface. |
| SimpleFactory.java<br>SimpleFactoryHelper.java<br>SimpleFactoryHolder.java<br>_SimpleFactoryImplBase.java<br>_SimpleFactoryStub.java | Generated by the m3idltojava command for the SimpleFactory interface. |
| Simple.ser | The server descriptor file that is generated by the buildjavaserver command. |
| Simple.jar | The CORBA/Java server archive file that is generated by the buildjavaserver command. |
| SimpleClient.jar | The CORBA/Java client archive file that is generated by the make or nmake command. |

## Files Generated in the tux Subdirectory

Table 4-10 describes the files that are generated in the tux subdirectory.

**Table 4-10 File Generated in the tux Subdirectory**

| | |
|---|---|
| simpserv.exe | Simpapp server object file that is generated by the buildserver command. |
| simpserv.obj | Simpapp server object file. |

## Files Generated in the results Subdirectory

Table 4-11 describes the files that are generated in the results subdirectory.

**Table 4-11 Files Generated in the results Subdirectory**

| File | Description |
|---|---|
| `input` | Generated by the `runme` command. Contains the input that `runme` gives to the `SimpleClient` CORBA/Java application. |
| `output` | Generated by the `runme` command. Contains the output that is produced when `runme` executes the `SimpleClient` CORBA/Java application. |
| `expected_output` | Generated by the `runme` command. Contains the output that is expected when the `SimpleClient` CORBA/Java application is executed by the `runme` command. The data in the output file is compared with the data in the `expected_output` file to determine whether the test passed or failed. |
| `log` | Generated by the `runme` command. Contains the output generated by the `runme` command. If the `runme` command fails, check this file and the `ULOG` file for errors. |
| `setenv.cmd` | Generated by the Windows NT `runme.cmd` command. Contains the commands to set the environment variables needed to build and execute the CORBA/Java-to-Tuxedo Simpapp sample application. |
| `setenv.ksh` | Generated by the UNIX `runme.ksh` command. Contains the commands to set the environment variables needed to build and execute the `Simpapp` sample. |
| `stderr` | Generated by the `tmboot` command, which is executed by the `runme` command. If the `-noredirect` server option is specified in the `UBBCONFIG` file, the `System.err.println` method sends the output to the `stderr` file instead of to the `ULOG` user log file. |
| `stdout` | Generated by the `tmboot` command, which is executed by the `runme` command. If the `-noredirect` server option is specified in the `UBBCONFIG` file, the `System.out.println` method sends the output to the `stdout` file instead of to the `ULOG` user log file |
| `tmsysevt.dat` | Generated by the `tmboot` command, which is executed by the `runme` command. It contains filtering and notification rules used by the `TMSYSEVT` (system event reporting) process. |

**Table 4-11 Files Generated in the results Subdirectory (Continued)**

| File | Description |
|---|---|
| tuxconfig | Generated by the tmloadcf command, which is executed by the runme command. |
| ubb | The UBBCONFIG file for the CORBA/Java-to-Tuxedo Simpapp sample application. |
| ULOG.<*date*> | A log file that contains messages generated by the tmboot command. If there are any compile or run-time errors, check this file. |

# Stopping the CORBA/Java-to-Tuxedo Simpapp Sample Application

Before using another sample application, use the following procedure to stop the CORBA/Java-to-Tuxedo Simpapp sample application and to remove unnecessary files from the work directory.

1. Stop the application:

   **Windows NT**

   ```
   prompt>tmshutdown -y
   ```

   **UNIX**

   ```
   ksh prompt>tmshutdown -y
   ```

2. Restore the working directory to its original state:

   **Windows NT**

   ```
   prompt>nmake -f makefile.nt clean
   ```

   **UNIX**

   ```
   prompt>. ./results/setenv.ksh
   prompt>make -f makefile.nt clean
   ```

# 5 EJB-to-Tuxedo Simpapp Sample Application

This topic includes the following sections:

- How the EJB-to-Tuxedo Simpapp Sample Application Works
- Building and Running the EJB-to-Tuxedo Simpapp Sample Application
- Stopping the EJB-to-Tuxedo Simpapp Sample Application

**Note:** Each sample application directory tree provided with the WebLogic Enterprise software includes a `Readme.txt` file that explains how to build and run the sample. Refer to this file in the following directory for troubleshooting information or other last-minute information about using the EJB-to-Tuxedo Simpapp sample application.

**Windows NT**

```
$TUXDIR\samples\interop\ejb_tux
```

**UNIX**

```
$TUXDIR/samples/interop/ejb_tux
```

# How the EJB-to-Tuxedo Simpapp Sample Application Works

This topic includes the following sections:

- Key Application Components
- Application Flow
- Software Prerequisites
- Example Code

The EJB-to-Tuxedo Simpapp sample application demonstrates the use of Java Enterprise Tuxedo (JET) technology to invoke a Tuxedo service from an EJB server running in the WebLogic Enterprise EJB container. For more information about JET, See *Using Java Enterprise Tuxedo*.

## Key Application Components

The EJB-to-Tuxedo Simpapp sample application consists of the following main components:

- An EJB client (`SimpClient`) prompts the user for a string and then invokes methods on the EJB, `upper` and `lower`, to convert the string to all uppercase and all lowercase text, respectively. This client then displays the results of the conversion to the user.
- An Enterprise JavaBean (`SimpBean`) acts as the bridge between the WebLogic Enterprise and Tuxedo environments. This object provides the following methods to handle the conversion:
  - The `upper` method calls another method, `joltNativeCall`, which instantiates a `JoltService` object and uses the `call` method to invoke the TOUPPER service in the Tuxedo environment. The `JoltService` class is a component of the JET Class Library.

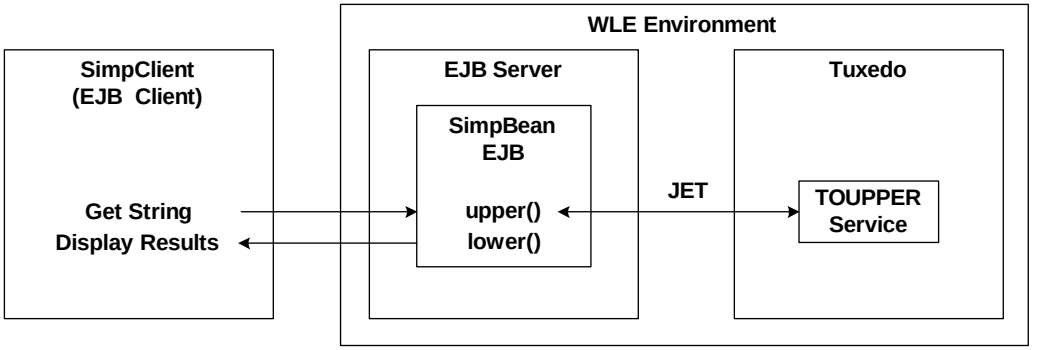
  - The `lower` method uses the Java `toLowerCase` method to perform the lowercase conversion.

- The TOUPPER service in the Tuxedo environment, which converts a text string to all uppercase characters.

# Application Flow

Figure 5-1 illustrates how the EJB-to-Tuxedo Simpapp sample application works.

**Figure 5-1 Overview of the EJB-to-Tuxedo Simpapp Sample Application**




# Software Prerequisites

In order to invoke a Tuxedo service using JET, you need to install Visual C++ version 6.0 with Service Pack 3 or later for Visual Studio.

# Example Code

Listing 5-1 shows the `joltNativeCall` method from the SimpBean EJB in the EJB-to-Tuxedo Simpapp sample application. This method uses JET technology to invoke the TOUPPER service in the Tuxedo environment. It accepts the following parameters:

- svcName is the name of the Tuxedo service to invoke (TOUPPER).
- data is the string that the user entered in the SimpClient client.

**Listing 5-1 The joltNativeCall() Method in the SimpBean EJB**

```
String joltNativeCall (String svcName, String data)
{
  JoltService svc;

  try {
    svc = new JoltService (svcName);
    svc.addString("STRING", data);
    svc.call (null);
  } catch (ServiceException e) {
    System.out.println("JoltService got "+e);
    return new String("");
  }

  return svc.getStringDef("STRING", "no_response");
}
```

The joltNativeCall method performs the following operations:

- Instantiating a new JoltService object to represent the Tuxedo TOUPPER service. The JoltService class is part of the JET Class Library. For more information about the JET Class Library, see *Using Java Enterprise Tuxedo*.
- Calling the addString method on the JoltService object to set up the input parameters to the TOUPPER service: the conversion type (“STRING”) and the string to convert (data).
- Calling the call method to invoke the TOUPPER service.
- Returning the results to the calling method, upper(), in the SimpBean EJB.

# Building and Running the EJB-to-Tuxedo Simpapp Sample Application

This section includes the following steps to build the EJB-to-Tuxedo Simpapp sample application:

- Step 1: Verify the Settings of Environment Variables
- Step 2: Copy the Files into a Work Directory
- Step 3: Change the Protection Attribute on the Files for the EJB-to-Tuxedo Simpapp Sample Application
- Step 4: Run the EJB-to-Tuxedo Simpapp Sample Application

## Step 1: Verify the Settings of Environment Variables

Before building and running the EJB-to-Tuxedo Simpapp sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure that they reflect correct information.

### Required Environment Variables

Table 5-1 describes the environment variables that are required to run the EJB-to-Tuxedo Simpapp sample application.

**Table 5-1 Required Environment Variables for the EJB-to-Tuxedo Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| TUXDIR | The directory path where you installed the WebLogic Enterprise software. For example:<br>**Windows NT**<br>TUXDIR=c:\WLEdir<br>**UNIX**<br>TUXDIR=/usr/local/WLEdir |
| JAVA_HOME | The directory path where you installed the JDK software. For example:<br>**Windows NT**<br>JAVA_HOME=c:\JDK1.2.2<br>**UNIX**<br>JAVA_HOME=/usr/local/JDK1.2.2 |

## Optional Environment Variables

You may optionally set the following system environment variables to change their default value before running the EJB-to-Tuxedo Simpapp sample `runme` command. See the *Administration Guide* for more information about selecting appropriate values for these environment variables.

Table 5-2 describes the optional environment variables that you can set before running the EJB-to-Tuxedo Simpapp sample application.

**Table 5-2 Optional Environment Variables for the EJB-to-Tuxedo Simpapp Sample Application**

| Environment Variable | Description |
|---|---|
| HOST | The host name portion of the TCP/IP network address used by the ISL process to accept connections from CORBA. The default value is the name of the local machine. |

**Table 5-2 Optional Environment Variables for the EJB-to-Tuxedo Simpapp Sample Application (Continued)**

| Environment Variable | Description |
|---|---|
| PORT | The TCP port number at which the ISL process listens for incoming requests. It must be a number between 0 and 65535. The default value is 2468. |
| IPCKEY | The address of shared memory. It must be a number greater than 32769 that is unique to this application on this system. The default value is 55432. |

## Verifying the Environment Variables

To verify that the information for the environment variables defined during installation is correct, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. Check the settings for `TUXDIR` and `JAVA_HOME`.

**UNIX**

1. Execute the `ksh` command to use the Korn shell.
2. Execute the `printenv` command to display the values of `TUXDIR` and `JAVA_HOME`, as shown in the following example:

```
ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
```

## Changing the Environment Variables

To change the environment variable settings, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.

   The Control Panel appears.
3. Click the System icon.

   The System Properties window appears.
4. Click the Environment tab.

   The Environment page appears.
5. On the Environment page in the System Properties window, click the environment variable you want to change, or enter the name of the environment variable in the Variable field.
6. Enter the correct information for the environment variable in the Value field.
7. Click OK to save the changes.

**UNIX**

1. Execute the `ksh` command to use the Korn shell.
2. Execute the `export` command to set the correct values for the `TUXDIR` and `JAVA_HOME` environment variables, as in the following example:

```
ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath
```

# Step 2: Copy the Files into a Work Directory

You need to copy the files for the EJB-to-Tuxedo Simpapp sample application into a work directory on your local machine. The files for the EJB-to-Tuxedo Simpapp sample application are located in the following directories under `TUXDIR`:

**Windows NT**

```
$TUXDIR\samples\interop\ejb_tux
```

**UNIX**

```
$TUXDIR/samples/interop/ejb_tux
```

## Copying the Files

The following steps describe how to execute a makefile to copy all of the example files into a work directory.

1. Create the work directory on your machine.
2. Copy the entire `ejb_tux` directory to the working directory created in the previous step:

   **Windows NT**

   ```
   > copy $TUXDIR\samples\interop\ejb_tux\*.* <work_directory>
   ```

   **UNIX**

   ```
   > cp -R $TUXDIR/samples/interop/ejb_tux/* <work_directory>
   ```

3. Change to the working directory created in step 1.
4. Execute the following command, which copies the remaining EJB-to-Tuxedo Simpapp sample application files to the working directory:

   **Windows NT**

   ```
   >nmake -f makefile.nt copy
   ```

   **UNIX**

   ```
   >make -f makefile.mk copy
   ```

## Files Copied to the Working Directory

This section describes the files that were copied into your working directory when you executed the makefile.

## Utility Files

Table 5-3 describes the utility files for this sample application. These files reside in the root of the working directory.

**Table 5-3 Utility Files in the Root of the Working Directory**

| File | Description |
|---|---|
| `Readme.txt` | Contains directions for building and executing the EJB-to-Tuxedo Simpapp sample application. |
| `runme.cmd` | Windows NT batch file that contains commands to build and execute the EJB-to-Tuxedo Simpapp sample application. |
| `runme.ksh` | UNIX Korn shell script that contains commands to build and execute the EJB-to-Tuxedo Simpapp sample application. |
| `makefile.nt` | Common makefile for the EJB-to-Tuxedo Simpapp sample application on the Windows NT platform. This makefile can be used directly by the Visual C++ `nmake` command. |
| `makefile.mk` | Makefile for the EJB-to-Tuxedo Simpapp sample application on the UNIX platform. |

## EJB Files

Table 5-4 describes the files that are needed to create the EJB client and EJB server for this sample application. These files reside in the `ejb` subdirectory of the working directory.

**Table 5-4 EJB Files in the ejb Subdirectory**

| File | Description |
|---|---|
| `SimpClient.java` | The EJB client application that calls methods on the `SimpBean` object. |

**Table 5-4 EJB Files in the ejb Subdirectory (Continued)**

| File | Description |
|---|---|
| SimpBean.java | The Java source code for the SimpBean class. This is an example of a stateless session bean. This bean contains the methods that are invoked by the SimpClient class. |
| Simp.java | The Java source code for the Remote interface of the SimpBean class. |
| SimpHome.Java | The Java source code for the Home interface of the SimpBean class. |
| weblogic-ejb-extensions.XML | The XML file specifying the WebLogic EJB extensions to the deployment descriptor DTD. |
| ejb-jar.xml | The standard deployment descriptor for the SimpBean class. |

## Tuxedo Files

Table 5-5 describes the source files for the Tuxedo portion of this sample application. These files reside in the tux subdirectory of the working directory.

**Table 5-5 Tuxedo Files in the tux Subdirectory**

| File | Description |
|---|---|
| jrepository | Jolt Repository definition file. |
| simpserv.c | Simpapp server source code. |

# Step 3: Change the Protection Attribute on the Files for the EJB-to-Tuxedo Simpapp Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit or build the files in the EJB-to-Tuxedo Simpapp sample application, you need to change the protection attribute of the files you copied into your work directory (including the respective `ejb` and `corbaj` subdirectories), as follows:

**Windows NT**

```
prompt>attrib /S -r drive:\workdirectory\*.*
```

**UNIX**

```
prompt>/bin/ksh
ksh prompt>chmod +w /workdirectory/*.*
```

On UNIX, you also need to change the permission of `runme.ksh` to give execute permission to the file, as follows:

```
ksh prompt>chmod +x runme.ksh
```

# Step 4: Run the EJB-to-Tuxedo Simpapp Sample Application

Once you have copied the files and changed their protection attributes, you can build and run the EJB-to-Tuxedo Simpapp sample application by executing the `runme` command, which starts server processes, generates files in various subdirectories of the working directory, and starts the sample application.

## Executing the runme Command

The `runme` command automates the following steps:

1. Set the system environment variables.
2. Load the `UBBCONFIG` file.

3. Compile the code for the EJB server object.
4. Compile the code for the Jet Server application.
5. Start the server application using the `tmboot` command.
6. Start the client application.
7. Stop the server application using the `tmshutdown` command.

To build and run the CORBA/Java Simpapp sample application, execute the `runme` command, as follows:

**Windows NT**

```
prompt>cd workdirectory

prompt>runme
```

**UNIX**

```
ksh prompt>cd workdirectory

ksh prompt>./runme.ksh
```

The EJB-to-Tuxedo Simpapp sample application runs and prints the following messages:

```
Testing simpapp
    cleaned up
    prepared
    built
    loaded ubb
    booted
    ran
    shutdown
    saved results
  PASSED
```

All of the sample application output is placed in the `results` directory. You can check in that directory for the following files:

- The `.log` file, for any compile, server boot, or server shutdown errors.
- The `ULOG` file for server application errors and exceptions.
- The `output` file for EJB client application output and exceptions.

## Running the Sample Application Manually

After you have executed the `runme` command one time, you can subsequently run the EJB-to-Tuxedo Simpapp sample application manually.

To run the EJB-to-Tuxedo Simpapp sample application manually:

1. Verify that your environment variables are correct by entering the following command:

   **Windows NT**

   ```
   prompt>results\setenv
   ```

   **UNIX**

   ```
   prompt>. results/setenv.ksh
   ```

2. Run the sample:

   **Windows NT**

   ```
   prompt>tmboot -y
   prompt>java -classpath %CLIENTCLASSPATH% ejb.SimpClient
   corbaloc:%TOBJADDR%
   ```

   **UNIX**

   ```
   prompt>tmboot -y
   prompt>java -classpath ${CLIENTCLASSPATH} ejb.SimpClient
   corbaloc:${TOBJADDR}
   ```

3. To run the CORBA/C++ joint client/server application, enter a string. After you enter the string, the application returns the string in uppercase and lowercase characters, respectively:

   ```
   String?
   Hello World
   HELLO WORLD
   hello world
   ```

All of the sample application output is placed in the `results` directory. You can check in that directory for the following files:

- The `.log` file, for any compile, server boot, or server shutdown errors.
- The `ULOG` file for server application errors and exceptions.
- The `output` file for EJB client application output and exceptions.

## Server Processes Started by the Sample Application

Table 5-6 describes the server processes that are started when the `tmboot` command is executed to start the EJB-to-Tuxedo Simpapp sample application.

**Table 5-6 Server Processes Started When tmboot Is Executed**

| Process | Description |
|---|---|
| TMSYSEVT | The BEA Tuxedo system Event Broker. |
| TMFFNAME | Starts the following TMFFNAME processes:<br>■ The TMFFNAME server process with the `-N` option and the `-M` option is the MASTER `NameManager` service. The `-N` option says to start the NameManager Service. The `-M` option says to start this name manager as a Master. This service maintains a mapping of application-supplied names to object references.<br>■ The TMFFNAME server process with the `-N` option only is a SLAVE `NameManager` service.<br>■ The TMFFNAME server with the `-F` option contains the `FactoryFinder` object. |
| JavaServer | The Simpapp server process that implements EJB-JAR file for the `SimpBean` and `SimpHome` interfaces. The JavaServer has one argument, `SimpleEjb.jar`, which is the EJB Java ARchive (JAR) file that was created for the application. |
| JREPSVR | Jolt Repository Server, which manages the Jolt Repository. The Jolt Repository contains service definitions for BEA Tuxedo services. |
| ISL | The IIOP Listener/Handler. |

## Files Generated by the Sample Application

This section describes the files that are generated in various subdirectories of the working directory.

## File Generated in the .adm Subdirectory

Table 5-7 describes the file that is generated in the `.adm` subdirectory.

**Table 5-7 File in the .adm Subdirectory**

| File | Description |
|---|---|
| `.keydb` | Generated by the `tmloadcf` command. Contains the security encryption key database. |

## File Generated in the ejb Subdirectory

Table 5-10 describes the file that is generated in the `ejb` subdirectory.

**Table 5-8 File Generated in the ejb Subdirectory**

| File | Description |
|---|---|
| `ejb.jar` | EJB client archive file that is generated by the `make` or `nmake` command. |

## Files Generated in the tux Subdirectory

Table 5-9 describes the files that are generated in the `tux` subdirectory.

**Table 5-9 File Generated in the tux Subdirectory**

| | |
|---|---|
| `simpserv.exe` | Simpapp server object file that is generated by the `buildserver` command. |
| `simpserv.obj` | Simpapp server object file. |

## Files Generated in the results Subdirectory

Table 5-10 describes the files that are generated in the `results` subdirectory.

**Table 5-10 Files Generated in the results Subdirectory**

| File | Description |
|---|---|
| `input` | Generated by the `runme` command. Contains the input that `runme` gives to the `SimpClient` Java application. |
| `output` | Generated by the `runme` command. Contains the output that is produced when `runme` executes the `SimpClient` Java application. |
| `expected_output` | Generated by the `runme` command. Contains the output that is expected when the `SimpClient` Java application is executed by the `runme` command. The data in the `output` file is compared with the data in the `expected_output` file to determine whether the test passed or failed. |
| `log` | Generated by the `runme` command. Contains the output generated by the `runme` command. If the `runme` command fails, check this file, and the `ULOG` file, for errors. |
| `setenv.cmd` | Generated by the Windows NT `runme.cmd` command. Contains the commands to set the environment variables needed to build and execute the EJB-to-Tuxedo Simpapp sample application. |
| `setenv.ksh` | Generated by the UNIX `runme.ksh` command. Contains the commands to set the environment variables needed to build and execute the `Simpapp` sample application. |
| `stderr` | Generated by the `tmboot` command, which is executed by the `runme` command. If the `-noredirect` server option is specified in the `UBBCONFIG` file, the `System.err.println` method sends the output to `stderr` instead of to the `ULOG` user log file. |
| `stdout` | Generated by the `tmboot` command, which is executed by the `runme` command. If the `-noredirect` server option is specified in the `UBBCONFIG` file, the `System.out.println` method sends the output to the `stdout` file instead of to the `ULOG` user log file. |

**Table 5-10 Files Generated in the results Subdirectory (Continued)**

| File | Description |
|---|---|
| tmsysevt.dat | Generated by the tmboot command, which is executed by the runme command. It contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. |
| tuxconfig | Generated by the tmloadcf command, which is executed by the runme command. |
| ubb | The UBBCONFIG file for the EJB-to-Tuxedo Simpapp sample application. |
| ULOG.<*date*> | A log file that contains messages generated by the tmboot command. |

# Stopping the EJB-to-Tuxedo Simpapp Sample Application

Before using another sample application, use the following procedure to stop the EJB-to-Tuxedo Simpapp sample application and to remove unnecessary files from the work directory:

1. Stop the application:

   **Windows NT**

   ```
   prompt>tmshutdown -y
   ```

   **UNIX**

   ```
   ksh prompt>tmshutdown -y
   ```

2. Restore the working directory to its original state:

   **Windows NT**

   ```
   prompt>nmake -f makefile.nt clean
   ```

**UNIX**

```
prompt>. ./results/setenv.ksh
prompt>make -f makefile.nt clean
```

# Index

## C


compiling
  client applications
    CORBA/C++-to-EJB Simpapp sample application 3-14
    EJB-to-CORBA/Java Simpapp sample application 2-14
  server applications
    CORBA/C++-to-EJB Simpapp sample application 3-14
    CORBA/Java-to-Tuxedo Simpapp sample application 4-13, 4-14
    EJB-to-CORBA/Java Simpapp sample application 2-14
    EJB-to-Tuxedo Simpapp sample application 5-13
copied files
  CORBA/Java-to-Tuxedo Simpapp sample application 4-11
  EJB-to-Tuxedo Simpapp sample application 5-9
CORBA/C++-to-EJB Simpapp sample application 3-1
  changing protection on files 3-13
  compiling the Java client application 3-14
  compiling the Java server application 3-14
  loading the UBBCONFIG file 3-14
  required environment variables 3-7
  runme command 3-14
  setting up the work directory 3-10
CORBA/Java-to-Tuxedo Simpapp sample application
  application flow 4-3
  changing protection on files 4-13
  compiling the CORBA/Java server application 4-13
  compiling the JET server application 4-14
  components of 4-2
  copied files 4-11
  environment variables 4-7
  example code 4-5
  generated files 4-16
  how it works 4-2
  loading the UBBCONFIG file 4-13
  OMG IDL code 4-3
  runme command 4-13
  running the application manually 4-15
  server processes started 4-16
  setting up the work directory 4-10
  software prerequisites 4-5
  source files 4-10
  starting the client application 4-14, 4-15
  starting the server application 4-14
  stopping 4-20
customer support contact information ix

## D

directory location of source files
- CORBA/Java-to-Tuxedo Simpapp sample application 4-10
- EJB-to-CORBA/Java Simpapp sample application 2-10, 3-10
- EJB-to-Tuxedo Simpapp sample application 5-9

documentation, where to find it viii

## E

EJBs
- third-party 1-11

EJB-to-CORBA/Java sample application 2-2
EJB-to-CORBA/Java Simpapp sample application
- changing protection on files 2-13
- compiling the Java client application 2-14
- compiling the Java server application 2-14
- files for 2-10
- loading the UBBCONFIG file 2-14
- required environment variables 2-7
- runme command 2-14
- setting up the work directory 2-10
- source files 2-10, 3-10

EJB-to-Tuxedo Simpapp sample application
- application flow 5-3
- changing protection on files 5-12
- compiling the EJB server object 5-13
- compiling the JET server application 5-13
- components of 5-2
- copied files 5-9
- environment variables 5-5
- example code 5-3
- generated files 5-15
- how it works 5-2
- loading the UBBCONFIG file 5-12
- runme command 5-12
- running the application manually 5-14
- server processes started 5-15
- setting up the work directory 5-8
- software prerequisites 5-3
- source files 5-9
- starting the client application 5-13, 5-14
- starting the server application 5-13
- stopping 5-18

environment variables
- changing 2-9
- CORBA/C++-to-EJB Simpapp sample application 3-7
- CORBA/Java-to-Tuxedo Simpapp sample application 4-7
- EJB-to-CORBA/Java Simpapp sample application 2-7
- EJB-to-Tuxedo Simpapp sample application 5-5
- HOST 4-8, 5-6
- IPCKEY 4-8, 5-7
- JAVA_HOME 2-7, 3-7, 4-7, 5-6
- PORT 4-8, 5-7
- TUXDIR 2-7, 3-7, 4-7, 5-6
- verifying 2-8

example code
- CORBA/Java-to-Tuxedo Simpapp sample application 4-5
- EJB-to-Tuxedo Simpapp sample application 5-3

## F

file protections
- CORBA/C++-to-EJB Simpapp sample application 3-13
- CORBA/Java-to-Tuxedo Simpapp sample application 4-13
- EJB-to-CORBA/Java Simpapp sample application 2-13
- EJB-to-Tuxedo Simpapp sample

application 5-12

## G

generated files
CORBA/Java-to-Tuxedo Simpapp sample application 4-16
EJB-to-Tuxedo Simpapp sample application 5-15

## H

HOST 2-7
HOST environment variable 4-8, 5-6

## I

interoperability
third-party 1-11
IPCKEY 2-7
IPCKEY environment variable 4-8, 5-7
ISL process 2-16, 4-16, 5-15

## J

JAVA_HOME environment variable 4-7, 5-6
JAVA_HOME parameter
CORBA/C++-to-EJB Simpapp sample application 3-7
EJB-to-CORBA/Java Simpapp sample application 2-7
JavaServer process 2-16, 4-16, 5-15
JREPSVR process 4-16, 5-15

## M

m3idltojava compiler 2-5, 4-5

## O

ORBs
third-party 1-11

## P

PORT 2-7
PORT environment variable 4-8, 5-7
printing product documentation viii

## R

related information viii
runme command
CORBA/Java-to-Tuxedo Simpapp sample application 4-13
description 2-14, 3-14
EJB-to-Tuxedo Simpapp sample application 5-12
running the application manually
CORBA/Java-to-Tuxedo Simpapp sample application 4-15
EJB-to-Tuxedo Simpapp sample application 5-14

## S

sample applications
EJB-to-CORBA 2-2
setting up the work directory
CORBA/Java-to-Tuxedo Simpapp sample application 4-10
EJB-to-Tuxedo Simpapp sample application 5-8
Simpapp 2-2
SimpBean 3-7
software prerequisites 2-5
CORBA/Java-to-Tuxedo Simpapp sample application 4-5
EJB-to-Tuxedo Simpapp sample application 5-3
stopping
CORBA/Java-to-Tuxedo Simpapp sample application 4-20
EJB-to-Tuxedo Simpapp sample application 5-18

support
technical ix
Symantec Visual Cafe 2-21

## T

the 2-14, 3-14
third-party interoperability 1-11
TMFFNAME process 2-16, 4-16, 5-15
tmshutdown 2-21, 4-20
TMSYSEVT process 2-16, 4-16, 5-15
TUXDIR 2-7
TUXDIR environment variable 4-7, 5-6
TUXDIR parameter
CORBA/C++-to-EJB Simpapp sample application 3-7
EJB-to-CORBA/Java Simpapp sample application 2-7

## U

UBBCONFIG file
CORBA/C++-to-EJB Simpapp sample application 3-14
EJB-to-CORBA/Java Simpapp sample application 2-14

## V

Visual C++ compiler 2-5, 4-5, 5-3
Visual Cafe 2-21