Contents

About This Document
What You Need to Know ................................................................. vii
E-docs Website .................................................................................. viii
How to Print the Document .............................................................. viii
Related Information .......................................................................... viii
Contact Us! ...................................................................................... ix
Documentation Conventions ........................................................... ix

1. Introduction
Interoperability, Coexistence, and Transactions ............................ 1-1
Interoperability Among the CORBA, J2EE, and Tuxedo Programming
  Models ......................................................................................... 1-2
  BEA Servers Invoking BEA Servers ............................................ 1-3
  BEA Clients Invoking BEA Servers ............................................ 1-4
  Third-party ORB Interoperability .............................................. 1-6
Overview of the Interoperability Sample Applications .................. 1-6

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 .................... 2-6
  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.............................................................................................. 2-9
Files in the Working Directory........................................................................................................... 2-10
Changing the Protection Attribute on the Files for the EJB-to-CORBA/Java Simpapp Sample Application ................................................................. 2-13
Executing the runme Command ........................................................................................................ 2-13
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
File Generated in the ejb_corbaj Directory ......................................................................................... 2-18
Files Generated in the results Directory ............................................................................................ 2-18
Stopping the EJB-to-CORBA/Java Simpapp Sample Application ........................................................ 2-20

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 ........................................................................... 3-6
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......................................................................................... 3-9
Files in the Working Directory........................................................................................................ 3-10
Changing the Protection Attribute on the Files for the CORBA/C++-to-EJB Simpapp Sample Application ................................................................................................................... 3-13
Executing the runme Command............................................................................................................. 3-13
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
About This Document

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

This document covers 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.

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 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.beasys.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 the Adobe Acrobat Reader, you can get 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 WLE 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@beasys.com if you have questions or comments. Your comments will be reviewed directly by the BEA 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 5.0 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.beasys.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface text</td>
<td>Indicates terms defined in the glossary.</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Indicates that you must press two or more keys simultaneously.</td>
</tr>
</tbody>
</table>

CORBA, J2EE, and Tuxedo Interoperability and Coexistence
<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>italics</em></td>
<td>Indicates emphasis or book titles.</td>
</tr>
</tbody>
</table>
| *monospace text* | Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard. **Examples:**  
#include <iostream.h>  
void main ()  
the pointer psz  
chmod u+w *  
\tux\data\ap  
.doc  
tux.doc  
BITMAP  
float  |
| *boldface monospace text* | Identifies significant words in code. **Example:**  
void commit () |
| *monospace italic text* | Identifies variables in code. **Example:**  
String expr  |
| `UPPERCASE TEXT` | Indicates device names, environment variables, and logical operators. **Examples:**  
LPT1  
SIGNON  
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. **Example:**  
buildobjclient [-v] [-o name ] [-f file-list]...  
[-l file-list]... |
## Documentation Conventions

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

1 Introduction

This chapter describes the interoperability and coexistence capabilities in the WebLogic Enterprise (WLE) system among the CORBA, J2EE, and Tuxedo programming models, and also describes the interoperability sample applications provided with the WLE 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 WLE application.

This chapter does not discuss specific interoperability or coexistence details on the following topics:

- Security
- WebLogic Server and WLE Connectivity

Interoperability, Coexistence, and Transactions

In general, interoperability and coexistence among CORBA, J2EE, and Tuxedo entities in the WLE environment, with respect to transactions, is fully supported. BEA clients can initiate transactions that are propagated to and coordinated among CORBA objects, Tuxedo services, and EJBs in the WLE domain.

Note the following restrictions with regard to transactions and interoperability in the WLE system:

- Transactions can be fully coordinated among Tuxedo services, CORBA objects, and EJBs, with the following exceptions:
1 Introduction

- A CORBA/Java object may start a transaction that is propagated to a Tuxedo service, but a Tuxedo service cannot initiate a transaction that is propagated directly to a CORBA/Java object.
- An EJB in the WLE system may start a transaction that is propagated to a Tuxedo service, but a Tuxedo service cannot initiate a transaction that is propagated directly to an EJB.
- WLE client applications cannot initiate, terminate, or coordinate transactions that span across EJBs and CORBA objects in the WLE domain.
- Third-party client applications cannot initiate, terminate, or coordinate transactions in the WLE domain. Such clients must delegate to the WLE domain all the processes and functions related to coordinating a transaction.

As a WLE application developer, you can use transaction policies with your EJBs, CORBA objects, and Tuxedo services to ensure the appropriate transactional behaviors of WLE server applications.

Interoperability Among the CORBA, J2EE, and Tuxedo Programming Models

The key interoperability features are presented in the following categories:
- BEA servers invoking BEA servers
- BEA clients invoking BEA servers
- Third-party interoperability

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

The following table summarizes the interoperability support among the various BEA server applications. A plus sign (+) means that full interoperability is supported.

<table>
<thead>
<tr>
<th>Server Making Invocation</th>
<th>Tuxedo Service</th>
<th>CORBA C++ Object</th>
<th>CORBA Java Object</th>
<th>EJB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuxedo Service</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>CORBA C++ Object</td>
<td>+(See below)</td>
<td>+</td>
<td>+</td>
<td>+(See below)</td>
</tr>
<tr>
<td>CORBA Java Object</td>
<td>+</td>
<td>+</td>
<td>+(See below)</td>
<td></td>
</tr>
<tr>
<td>EJB</td>
<td>+(See below)</td>
<td>+(See below)</td>
<td>+(See below)</td>
<td>+</td>
</tr>
</tbody>
</table>

Note the following details about the preceding table:

- **CORBA C++ object invoking a Tuxedo service**
  You can create a C++ object with a set of operations that map one-to-one with 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.

- **CORBA C++ object invoking an EJB**
  You can create an intermediary, or wrapper, Java object between the C++ object and the EJB. See Chapter 3, “CORBA/C++-to-EJB Simpapp Sample Application,” for an example application showing this feature.

- **CORBA Java object invoking an EJB**
  In the WLE 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.
1 Introduction

- **EJB invoking a Tuxedo service**
  
  For information, see a BEA Professional Services Organization representative.

- **EJB invoking a CORBA C++ object**
  
  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.

- **EJB invoking a CORBA Java object**
  
  In the WLE environment, an EJB can invoke a CORBA Java object directly. For an example, see Chapter 2, “EJB-to-CORBA/Java Simpapp Sample Application.”

### BEA Clients Invoking BEA Servers

The following table summarizes the interoperability support among BEA clients invoking BEA servers. A checkmark (+) means that full interoperability is supported.

<table>
<thead>
<tr>
<th>Client Making Invocation</th>
<th>Server Being Invoked</th>
<th>Tuxedo Service</th>
<th>CORBA C++ Object</th>
<th>CORBA Java Object</th>
<th>EJB</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLE CORBA</td>
<td></td>
<td>+(See below)</td>
<td>+</td>
<td>+</td>
<td>+(See below)</td>
</tr>
<tr>
<td>WLE RMI</td>
<td></td>
<td>+(See below)</td>
<td>+(See below)</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Tuxedo/WS</td>
<td></td>
<td>+(See below)</td>
<td>+(See below)</td>
<td>+(See below)</td>
<td></td>
</tr>
</tbody>
</table>

Note the following details about the preceding table:

- **WLE CORBA 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. See the Wrapper University sample application for an...
example application that shows this feature. You can extend this example by converting the C++ server object into a WLE CORBA C++ client.

- **WLE CORBA client application invoking an EJB**
  Chapter 3, “CORBA/C++-to-EJB Simpapp Sample Application.” shows an example of a C++ object invoking an EJB via a CORBA Java intermediary object.

- **WLE RMI client application invoking a CORBA C++ object**
  A WLE 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 Java object, which is invoked by the EJB, delegates the invocation to the C++ object.

- **WLE RMI client application invoking a CORBA Java object**
  A WLE 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 WLE 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.
Third-party ORB Interoperability

The WLE C++ ORB supports the IIOP 1.2 protocol, and the WLE 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.

WLE 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 WLE domain. These client products can connect to a WLE 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)

Overview of the Interoperability Sample Applications

The WLE software includes the sample applications described in Table 1-1:

Table 1-1 The Interoperability Sample Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJB-to-CORBA/Java Simpapp</td>
<td>Shows an EJB server acting as a client invoking a request and receiving a response from a CORBA/Java object</td>
</tr>
<tr>
<td>CORBA/C++-to-EJB Simpapp</td>
<td>Shows CORBA/C++ client invoking a request and receiving a response from an EJB server</td>
</tr>
</tbody>
</table>
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 chapter discusses the following topics:

- 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 WLE 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.

The following figure illustrates how the EJB-to-CORBA/Java Simpapp sample application works.
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 WLE 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 WLE 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.

The following code fragment shows the methods on the SimpleBridge object that delegate the SimpBean’s requests to the CORBA/Java Simple object:

```java
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 the following table.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Operation</th>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory</td>
<td>Creates object references to the Simple object</td>
<td>find_simple()</td>
<td>Activation: method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transaction: optional</td>
</tr>
<tr>
<td>Simple</td>
<td>Converts the case of a string</td>
<td>to_upper()</td>
<td>Activation: method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Transaction: optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to_lower()</td>
<td></td>
</tr>
</tbody>
</table>

Listing 2-1 shows the simple.idl file that defines the CORBA interfaces in the EJB-to-CORBA/Java Simpapp sample application.

**Listing 2-1 OMG IDL Code for the EJB-to-CORBA/Java Simpapp Sample Application**

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

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

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 2-1 lists the environment variables required to run the EJB-to-CORBA/Java Simpapp sample application.

**Table 2-1 Required Environment Variables for the EJB-to-CORBA/Java Simpapp Sample Application**

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| TUXDIR               | The directory path where you installed the WLE software. For example:  
                        | **Windows NT**   
                        | TUXDIR=c:\WLEdir  
                        | **UNIX**         
                        | TUXDIR=/usr/local/WLEdir |
| JAVA_HOME            | The directory path where you installed the JDK software. For example:  
                        | **Windows NT**   
                        | JAVA_HOME=c:\JDK1.2.2  
                        | **UNIX**         
                        | JAVA_HOME=/usr/local/JDK1.2.1 |

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

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

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>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.</td>
</tr>
</tbody>
</table>
To verify that the information for the environment variables defined during installation is correct, perform 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

Environment Variable Description

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT</td>
<td>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.</td>
</tr>
<tr>
<td>IPCKEY</td>
<td>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.</td>
</tr>
</tbody>
</table>
Changing the Environment Variables

To change the environment variable settings, perform 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:
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 working 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 working directory created in step 1.

4. Enter the following command, which copies the remaining EJB-to-CORBA/Java Simpapp sample application files to the working directory:

Windows NT:

>nmake -f makefile.nt copy

UNIX:

>make -f makefile.mk copy

Files in the Working Directory

This section lists and describes the files copied into your working directory after you have performed 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

Table 2-3 lists and describes the source files for the EJB portion of this sample application. These are the files that exist after you do the make command. These files are copied into a subdirectory named ejb.

Table 2-3 EJB Simpapp Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ejb-jar.XML</td>
<td>The standard deployment descriptor for the SimpBean class.</td>
</tr>
<tr>
<td>weblogic-ejb-extensions.XML</td>
<td>The XML file specifying the WebLogic EJB extensions to the deployment descriptor DTD.</td>
</tr>
<tr>
<td>SimpClient.Java</td>
<td>The EJB Simpapp client.</td>
</tr>
<tr>
<td>SimpBean.java</td>
<td>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.</td>
</tr>
<tr>
<td>Simp.java</td>
<td>The Remote interface of the SimpBean class.</td>
</tr>
<tr>
<td>SimpHome.Java</td>
<td>The Home interface of the SimpBean class.</td>
</tr>
</tbody>
</table>

Table 2-4 lists and describes the source files for the CORBA/Java portion of this sample application. They are copied into a subdirectory named corbaj.

Table 2-4 CORBA/Java Simpapp Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.idl</td>
<td>The OMG IDL that declares the SimpleFactory and Simple interfaces.</td>
</tr>
<tr>
<td>Simple.xml</td>
<td>The Server Description File for the Simple CORBA object.</td>
</tr>
</tbody>
</table>
Table 2-5 lists and describes the utility files for this sample application.

**Table 2-5 EJB-to-CORBA/Java Utility Files**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleBridge.Java</td>
<td>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.</td>
</tr>
<tr>
<td>SimpleFactoryImpl.Java</td>
<td>The implementation of the SimpleFactory methods.</td>
</tr>
<tr>
<td>SimpleImpl.Java</td>
<td>The implementation of the Simple methods.</td>
</tr>
<tr>
<td>Readme.txt</td>
<td>Contains directions for building and executing the EJB-to-CORBA/Java Simpapp sample application.</td>
</tr>
<tr>
<td>runme.cmd</td>
<td>The Windows NT batch file that contains commands to build and execute the EJB-to-CORBA/Java Simpapp sample application.</td>
</tr>
<tr>
<td>runme.ksh</td>
<td>The UNIX Korn shell script that contains commands to build and execute the EJB-to-CORBA/Java Simpapp sample application.</td>
</tr>
<tr>
<td>makefile.nt</td>
<td>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++ make command. The makefile.nt file is included by the smakefile.nt file.</td>
</tr>
<tr>
<td>smakefile.nt</td>
<td>The makefile for the EJB-to-CORBA/Java Simpapp sample application to be used by Symantec's Visual Café smake program.</td>
</tr>
</tbody>
</table>
Changing the Protection Attribute on the Files for the EJB-to-CORBA/Java Simpapp Sample Application

During the installation of the WLE 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:**

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

**UNIX:**

```bash
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:

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

**Executing the runme Command**

The `runme` command automates the following steps:

1. Setting the system environment variables
2. Loading the `UBBCONFIG` file
3. Compiling the code for the EJB server object
4. Compiling the code for the CORBA/Java server application
5. Starting the server application using the `tmboot` command
6. Starting the client application
7. Stopping 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` 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 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 \texttt{tmboot} command is executed to start the EJB-to-CORBA/Java Simpapp sample application, the following server processes are started:

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMSYSEVT</td>
<td>The BEA Tuxedo system Event Broker.</td>
</tr>
<tr>
<td>TMFFNAME</td>
<td>Starts the following TMFFNAME processes:</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process with the \texttt{-N} option and the \texttt{-M} option is the MASTER NameManager service. The \texttt{-N} option says to start the NameManager Service; the \texttt{-M} option says to start this name manager as a Master. This service maintains a mapping of application-supplied names to object references.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process with the \texttt{-N} option only is a SLAVE NameManager service.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server with the \texttt{-F} option contains the FactoryFinder object.</td>
</tr>
<tr>
<td>JavaServer</td>
<td>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.</td>
</tr>
<tr>
<td>JavaServer</td>
<td>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.</td>
</tr>
<tr>
<td>ISL</td>
<td>The IIOP Listener/Handler.</td>
</tr>
</tbody>
</table>
## Files Generated in the corbaj Directory

The following table lists and describes the files that are generated in the `corbaj` working directory.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.java</td>
<td>Generated by the <code>m3idltojava</code> command for the <code>Simple</code> interface. This interface contains the</td>
</tr>
<tr>
<td></td>
<td>Java version of the IDL interface. It extends the <code>org.omg.CORBA.Object</code> class.</td>
</tr>
<tr>
<td>SimpleHelper.java</td>
<td>Generated by the <code>m3idltojava</code> command for the <code>Simple</code> interface. This class provides auxiliary</td>
</tr>
<tr>
<td></td>
<td>functionality, notably the narrow method.</td>
</tr>
<tr>
<td>SimpleHolder.java</td>
<td>Generated by the <code>m3idltojava</code> command for the <code>Simple</code> interface. This class holds a public</td>
</tr>
<tr>
<td></td>
<td>instance member of type <code>Simple</code>. It provides operations for <code>out</code> and <code>inout</code> arguments, which</td>
</tr>
<tr>
<td></td>
<td>CORBA has, but which do not map easily to Java's semantics.</td>
</tr>
<tr>
<td>_SimpleImplBase.java</td>
<td>Generated by the <code>m3idltojava</code> command for the <code>Simple</code> interface. This abstract class is the</td>
</tr>
<tr>
<td></td>
<td>server skeleton. It implements the <code>Simple.java</code> interface. The server class <code>SimpleImpl</code> extends</td>
</tr>
<tr>
<td></td>
<td><code>_SimpleImplBase</code>.</td>
</tr>
<tr>
<td>_SimpleStub.java</td>
<td>Generated by the <code>m3idltojava</code> command for the <code>Simple</code> interface. This class is the client stub.</td>
</tr>
<tr>
<td></td>
<td>It implements the <code>Simple.java</code> interface.</td>
</tr>
<tr>
<td>SimpleFactory.java</td>
<td>Generated by the <code>m3idltojava</code> command for the <code>SimpleFactory</code> interface.</td>
</tr>
<tr>
<td>SimpleFactoryHelper.java</td>
<td></td>
</tr>
<tr>
<td>SimpleFactoryHolder.java</td>
<td></td>
</tr>
<tr>
<td>_SimpleFactoryImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_SimpleFactoryStub.java</td>
<td></td>
</tr>
<tr>
<td>Simple.ser</td>
<td>The server descriptor file that is generated by the <code>buildjavaserver</code> command.</td>
</tr>
</tbody>
</table>
File Generated in the ejb_corbaj Directory

The following file is generated in the ejb_corbaj directory:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>results directory</td>
<td>Generated by the runme command.</td>
</tr>
<tr>
<td>.adm/.keydb</td>
<td>Generated by the tmloadcf command. Contains the security encryption key database.</td>
</tr>
</tbody>
</table>

Files Generated in the results Directory

The following table lists and describes the files that are generated in the results directory, which is a subdirectory of the ejb_corbaj working directory.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Generated by the runme command. Contains the input that runme gives to the SimpleClient Java application.</td>
</tr>
<tr>
<td>output</td>
<td>Generated by the runme command. Contains the output that is produced when runme executes the SimpleClient Java application.</td>
</tr>
<tr>
<td>expected_output</td>
<td>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.</td>
</tr>
<tr>
<td>log</td>
<td>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.</td>
</tr>
</tbody>
</table>
## Building and Running the EJB-to-CORBA/Java Simpapp Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setenv.cmd</td>
<td>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.</td>
</tr>
<tr>
<td>setenv.ksh</td>
<td>Generated by the UNIX runme.ksh command. Contains the commands to set the environment variables needed to build and execute the Simpapp sample.</td>
</tr>
<tr>
<td>stderr</td>
<td>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.</td>
</tr>
<tr>
<td>stdout</td>
<td>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.</td>
</tr>
<tr>
<td>tmsysevt.dat</td>
<td>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.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>Generated by the tmloadcf command, which is executed by the runme command.</td>
</tr>
<tr>
<td>ubb</td>
<td>The UBBCONFIG file for the EJB-to-CORBA/Java Simpapp sample application.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command. If there are any compile or run-time errors, check this file.</td>
</tr>
</tbody>
</table>
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. Stop the application:
   - **Windows NT:**
     ```shell
     prompt>tmshutdown -y
     ```
   - **UNIX:**
     ```shell
     ksh prompt>tmshutdown -y
     ```

2. Restore the working directory to its original state:
   - **Windows NT:**
     ```shell
     prompt>nmake -f makefile.nt clean
     ```
   - **UNIX:**
     ```shell
     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:
   ```shell
   prompt>results\setenv
   prompt>set JDKDIR=%JAVA_HOME%
   prompt>smake -f smakefile.nt
   ```
This chapter discusses the following topics:

- How the CORBA/C++-to-EJB 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 WLE 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:**

$\text{TUXDIR/samples/interop/cpp_ejb}$

**UNIX:**

$\text{TUXDIR/samples/interop/cpp_ejb}$
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 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.

The following figure illustrates how the CORBA/C++-to-EJB Simpapp sample application works.
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 WLE 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.

The following code fragment shows the methods on the SimpleBridge object that delegate the SimpleImpl’s requests to the EJB server application:

```java
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 the following table.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Operation</th>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory</td>
<td>Creates object references to the Simple object.</td>
<td>find_simple()</td>
<td>Activation: method Transaction: optional</td>
</tr>
<tr>
<td>Simple</td>
<td>Delegates the conversion of the string to the EJB server.</td>
<td>to_upper()</td>
<td>Activation: method Transaction: optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to_lower()</td>
<td></td>
</tr>
</tbody>
</table>

Listing 3-1 shows the simple.idl file that defines the CORBA interfaces in the CORBA/C++-to-EJB Simpapp sample application.

Listing 3-1 OMG IDL Code for the CORBA/C++-to-EJB Simpapp Sample Application

```idl
#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
{

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

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

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

Table 3-1 lists the environment variables required to run the CORBA/C++-to-EJB Simpapp sample application.

Table 3-1 Required Environment Variables for the CORBA/C++-to-EJB Simpapp Sample Application

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| TUXDIR               | The directory path where you installed the WLE software. For example:  
  Windows NT          | TUXDIR=c:\WLEdir |
  UNIX                 | TUXDIR=/usr/local/WLEdir |
| JAVA_HOME            | The directory path where you installed the JDK software. For example:  
  Windows NT          | JAVA_HOME=c:\JDK1.2.2 |
  UNIX                 | JAVA_HOME=/usr/local/JDK1.2.1 |

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 3-2 lists the optional environment variables you can assign prior to running the CORBA/C++-to-EJB Simpapp sample application.

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

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>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.</td>
</tr>
</tbody>
</table>
To verify that the information for the environment variables defined during installation is correct, perform 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

---

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PORT</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>IPCKEY</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Changing the Environment Variables

To change the environment variable settings, perform 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.
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 \texttt{cpp\_ejb} directory to the working directory created in the previous step:

   \textbf{Windows NT:}
   \begin{verbatim}
   > copy \$TUXDIR\samples\interop\cpp\_ejb\*.* < work\_directory>
   \end{verbatim}

   \textbf{UNIX:}
   \begin{verbatim}
   > cp -R \$TUXDIR/samples/interop/cpp\_ejb/* < work\_directory>
   \end{verbatim}

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

   \textbf{Windows NT:}
   \begin{verbatim}
   >nmake -f makefile.nt copy
   \end{verbatim}

   \textbf{UNIX:}
   \begin{verbatim}
   >make -f makefile.mk copy
   \end{verbatim}

\section*{Files in the Working Directory}

This section lists and describes the files copied into your working directory after you have performed 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

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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simplec.cpp</td>
<td>C++ client program for the simple sample application.</td>
</tr>
<tr>
<td>simple.idl</td>
<td>The OMG IDL that declares the SimpleFactory and Simple interfaces.</td>
</tr>
<tr>
<td>simple.xml</td>
<td>The XML source file used to associate activation and transaction policy values with interfaces.</td>
</tr>
<tr>
<td>SimpleFactoryImpl.Java</td>
<td>The Java source code that implements the SimpleFactory methods.</td>
</tr>
<tr>
<td>SimpleImpl.Java</td>
<td>The Java source code that implements the Simple methods.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weblogic-ejb-extensions.XML</td>
<td>The XML file specifying the WebLogic EJB extensions to the deployment descriptor DTD.</td>
</tr>
</tbody>
</table>
Table 3-5 lists and describes the utility files for this sample application.

Table 3-5  CORBA/C++-to-EJB Simpapp Utility Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpBean.java</td>
<td>The Java source code for the SimpBean class. This is an example of a stateless session bean. This bean contains the methods that invoked by the SimpleBridge class.</td>
</tr>
<tr>
<td>Simp.java</td>
<td>The Java source code for the Remote interface of the SimpBean class.</td>
</tr>
<tr>
<td>SimpHome.java</td>
<td>The Java source code for the Home interface of the SimpBean class.</td>
</tr>
<tr>
<td>SimpleBridge.java</td>
<td>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.</td>
</tr>
<tr>
<td>Readme.txt</td>
<td>Contains directions for building and executing the CORBA/C++-to-EJB Simpapp sample application.</td>
</tr>
<tr>
<td>runme.cmd</td>
<td>The Windows NT batch file that contains commands to build and execute the CORBA/C++-to-EJB Simpapp sample application.</td>
</tr>
<tr>
<td>runme.ksh</td>
<td>The UNIX Korn shell script that contains commands to build and execute the CORBA/C++-to-EJB Simpapp sample application.</td>
</tr>
<tr>
<td>makefile.nt</td>
<td>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.</td>
</tr>
</tbody>
</table>
Changing the Protection Attribute on the Files for the CORBA/C++-to-EJB Simpapp Sample Application

During the installation of the WLE 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. Setting the system environment variables
2. Loading the `UBBCONFIG` file
3. Compiling the code for the EJB server object
4. Compiling the code for the CORBA/C++ joint client/server application
5. Compiling the code for the CORBA/Java server application
6. Starting the server application using the `tmboot` command
7. Starting the client application
8. Stopping 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 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 following server processes are started:

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMSYSEVT</td>
<td>The BEA Tuxedo system Event Broker.</td>
</tr>
<tr>
<td>TMFFNAME</td>
<td>Starts the following TMFFNAME processes:</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process with the <code>-N</code> option and the <code>-M</code> option is the MASTER NameManager service.</td>
</tr>
<tr>
<td></td>
<td>- The <code>-N</code> option says to start the NameManager Service; the <code>-M</code> option says to start this name manager as a Master. This service maintains a mapping of application-supplied names to object references.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process with the <code>-n</code> option only is a SLAVE NameManager service.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server with the <code>-F</code> option contains the FactoryFinder object.</td>
</tr>
<tr>
<td>JavaServer</td>
<td>The simpapp server process that implements ejb-jar file for the SimpBean and SimpHome interfaces.</td>
</tr>
<tr>
<td></td>
<td>The JavaServer has one argument, SimpleEjb.jar, which is the EJB Java Archive (JAR) file that was created for the application.</td>
</tr>
<tr>
<td>JavaServer</td>
<td>The simpapp server process that implements the SimpleFactory interface and the Simple interface.</td>
</tr>
<tr>
<td></td>
<td>The JavaServer has one argument, SimpleCorba.jar, which is the CORBA Java Archive (JAR) file that was created for the application.</td>
</tr>
<tr>
<td>ISL</td>
<td>The IIOP Listener/Handler.</td>
</tr>
</tbody>
</table>
## Files Generated in the cpp Directory

The following table lists and describes the files generated in the `cpp` directory.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple_c.cpp</td>
<td>Client stubs for the Simple and SimpleFactory interfaces.</td>
</tr>
<tr>
<td>Simple_c.h</td>
<td>Client stub header for the Simple and SimpleFactory interfaces.</td>
</tr>
<tr>
<td>Simple_client.exe</td>
<td>C++ client executable.</td>
</tr>
<tr>
<td>Simple.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This interface contains the Java version of the IDL interface. It extends the base class <code>org.omg.CORBA.Object</code>.</td>
</tr>
<tr>
<td>SimpleHelper.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class provides auxiliary functionality, notably the <code>narrow</code> method.</td>
</tr>
<tr>
<td>SimpleHolder.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class holds a public instance member of type <code>Simple</code>. It provides operations for <code>out</code> and <code>inout</code> arguments, which CORBA has, but which do not map easily to Java's semantics.</td>
</tr>
<tr>
<td>_SimpleImplBase.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This abstract class is the server skeleton. It implements the <code>Simple.java</code> interface. The server class <code>SimpleImpl</code> extends <code>_SimpleImplBase</code>.</td>
</tr>
<tr>
<td>_SimpleStub.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class is the client stub. It implements the <code>Simple.java</code> interface.</td>
</tr>
</tbody>
</table>
### File Generated in the cpp_ejb Directory

The following files are generated in the `cpp_ejb` directory:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory.java</td>
<td>Generated by the <code>m3idltojava</code> command for the SimpleFactory interface.</td>
</tr>
<tr>
<td>SimpleFactoryHelper.java</td>
<td></td>
</tr>
<tr>
<td>SimpleFactoryHolder.java</td>
<td></td>
</tr>
<tr>
<td>_SimpleFactoryImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_SimpleFactoryStub.java</td>
<td></td>
</tr>
<tr>
<td>Simple.ser</td>
<td>The server descriptor file that is generated by the <code>buildjavaserver</code> command.</td>
</tr>
<tr>
<td>Simple.jar</td>
<td>The Java archive file that is generated by the <code>buildjavaserver</code> command.</td>
</tr>
</tbody>
</table>

### Files Generated in the results Directory

The following table lists and describes the files that are generated in the `results` directory, which is a subdirectory of the `corbaj` working directory.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Generated by the <code>runme</code> command. Contains the input that <code>runme</code> gives to the <code>SimpleClient</code> Java application.</td>
</tr>
<tr>
<td>output</td>
<td>Generated by the <code>runme</code> command. Contains the output that is produced when <code>runme</code> executes the <code>SimpleClient</code> Java application.</td>
</tr>
</tbody>
</table>
### File Description

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expected_output</td>
<td>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.</td>
</tr>
<tr>
<td>log</td>
<td>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.</td>
</tr>
<tr>
<td>setenv.cmd</td>
<td>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.</td>
</tr>
<tr>
<td>setenv.ksh</td>
<td>Generated by the UNIX runme.ksh command. Contains the commands to set the environment variables needed to build and execute the Simpapp sample application.</td>
</tr>
<tr>
<td>stderr</td>
<td>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.</td>
</tr>
<tr>
<td>stdout</td>
<td>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.</td>
</tr>
<tr>
<td>tmsysevt.dat</td>
<td>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.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>Generated by the tmloadcf command, which is executed by the runme command.</td>
</tr>
<tr>
<td>ubb</td>
<td>The UBBCONFIG file for the CORBA/C++-to-EJB Simpapp sample application.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command.</td>
</tr>
</tbody>
</table>
3 CORBA/C++-to-EJB Simpapp Sample Application

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. 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`

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`
Index

B
BEA client interoperability 1-4
BEA server interoperability 1-3

C
client interoperability 1-4
clients
CORBA 1-4
JOLT 1-4
RMI 1-4
Tuxedo 1-4
compiling
client applications
CORBA/C++-to-EJB Simpapp
sample application 3-14
EJB-to-CORBA/Java Simpapp
sample application 2-13
server applications
CORBA/C++-to-EJB Simpapp
sample application 3-14
EJB-to-CORBA/Java Simpapp
sample application 2-13
CORBA C++ objects 1-3
CORBA Java objects 1-3
CORBA/C++-to-EJB Simpapp sample
application 3-1
changing protection on files 3-13
compiling the Java client application 3-13
compiling the Java server application 3-13
loading the UBBCONFIG file 3-13
required environment variables 3-6
runme command 3-13
setting up the work directory 3-9
customer support contact information ix

documentation, where to find it viii

D
directory location of source files
EJB-to-CORBA/Java Simpapp sample application 2-10, 3-10

E
EJBs
third-party 1-6
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 2-13
compiling the Java server application 2-13
files for 2-9
loading the UBBCONFIG file 2-13
required environment variables 2-6
runme command 2-13
setting up the work directory 2-9

CORBA, J2EE, and Tuxedo Interoperability and Coexistence   I-1
source files 2-10, 3-10
environment variables
    changing 2-9
    CORBA/C++-to-EJB Simpapp sample application 3-6
    EJB-to-CORBA/Java Simpapp sample application 2-6
    JAVA_HOME 2-6, 3-6
    TUXDIR 2-6, 3-6
    verifying 2-8

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

H
HOST 2-6

I
interoperability
    BEA clients to BEA servers 1-4
    BEA servers to BEA servers 1-3
    third-party 1-6
    IPCKEY 2-6
    ISL process 2-16

J
JAVA_HOME parameter
    CORBA/C++-to-EJB Simpapp sample application 3-6
    EJB-to-CORBA/Java Simpapp sample application 2-6
    JavaServer process 2-16

M
m3idltojava compiler 2-5

O
ORBs
    third-party 1-6

P
PORT 2-6
printing product documentation viii

R
related information viii
runme command
    description 2-13, 3-13

S
sample applications
    EJB-to-CORBA 2-2
    Simpapp 2-2
    SimpBean 3-6
    software prerequisites 2-5
    support
        technical ix
    Symantec Visual Cafe 2-20

T
third-party interoperability 1-6
    TMFFNAME process 2-16
tmshutdown 2-20
    TMSYSEVT process 2-16
    TUXDIR 2-6
    TUXDIR parameter
        CORBA/C++-to-EJB Simpapp sample application 3-6
        EJB-to-CORBA/Java Simpapp sample application 2-6

I-2    CORBA, J2EE, and Tuxedo Interoperability and Coexistence
Tuxedo service 1-3

U
UBBCONFIG file
   CORBA/C++-to-EJB Simpapp sample
      application 3-13
   EJB-to-CORBA/Java Simpapp sample
      application 2-13

V
Visual C++ compiler 2-5
Visual Cafe 2-20