Copyright

Copyright © 2000 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, BEA Builder, BEA Jolt, BEA Manager, BEA MessageQ, BEA Tuxedo, BEA TOP END, BEA WebLogic, and ObjectBroker are registered trademarks of BEA Systems, Inc. BEA eLink, BEA eSolutions, BEA TAP, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Express, BEA WebLogic Personalization Server, BEA WebLogic Server, Java Enterprise Tuxedo, and WebLogic Enterprise Connectivity are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

Guide to the University Sample Applications

<table>
<thead>
<tr>
<th>Document Edition</th>
<th>Date</th>
<th>Software Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>May 2000</td>
<td>BEA WebLogic Enterprise 5.1</td>
</tr>
</tbody>
</table>
Contents

About This Document
What You Need to Know ................................................................. x
e-docs Web Site ................................................................................. x
How to Print the Document .............................................................. x
Related Information ........................................................................... xi
Contact Us! ....................................................................................... xi
Documentation Conventions ......................................................... xii

1. Introduction
An Overview of the University Sample Applications ....................... 1-1
Naming Conventions Used in the University Sample Applications ....... 1-3

2. Setting Up Your Environment
System Prerequisites ........................................................................ 2-1
Editing the setenv and UBBCONFIG Files ........................................ 2-2
  Naming Conventions for the setenv and UBBCONFIG Files .......... 2-3
Setting setenv Parameters .............................................................. 2-4
Setting the UBBCONFIG Parameters .............................................. 2-6
Running the setenv Command ......................................................... 2-9

3. The Basic Sample Application
How the Basic Sample Application Works ...................................... 3-2
The OMG IDL for the Basic Sample Application ............................ 3-3
Generating the Client Stubs and the Skeletons ................................. 3-5
Writing the Client Application ......................................................... 3-6
  Writing the Server Application .................................................... 3-6
Configuring the Basic Sample Application ..................................... 3-7
Building the Basic Sample Application............................................................. 3-8
Copying the Files for the Basic Sample Application into a Work Directory .. 3-8
Changing the Protection on the File for the Basic Sample Application... 3-11
Setting the Environment Variables................................................................. 3-11
Initializing the University Database ........................................................... 3-11
Loading the UBBCONFIG File.................................................................. 3-12
Compiling the Basic Sample Application ....................................................... 3-12
Running the Basic Sample Application........................................................... 3-13
Starting the Server Application ................................................................. 3-13
Starting the CORBA C++ Client Application.......................................... 3-14
Starting the CORBA Java Client Application.......................................... 3-14
Starting the ActiveX Client Application .................................................. 3-15
Using the Client Applications in the Basic Sample Application ..................... 3-17
The CORBA C++ Client Application ...................................................... 3-17
The CORBA Java Client Application ...................................................... 3-18
The ActiveX Client Application............................................................... 3-19

4. The Security Sample Application
   How the Security Sample Application Works................................................ 4-1
   The Development Process for the Security Sample Application....................... 4-3
      OMG IDL ............................................................................................ 4-3
      The Client Application ....................................................................... 4-3
      The Server Application ...................................................................... 4-4
      The UBBCONFIG File ...................................................................... 4-4
      The ICF File ...................................................................................... 4-4
   Building the Security Sample Application .................................................... 4-4
   Copying the Files for the Security Sample Application into a Work Directory 4-5
    Changing the Protection on the Files for the Security Sample Application.... 4-7
    Setting the Environment Variables.......................................................... 4-8
    Initializing the University Database ........................................................ 4-8
    Loading the UBBCONFIG File................................................................ 4-8
   Compiling the Security Sample Application................................................. 4-9
   Running the Security Sample Application.................................................... 4-10
5. The Transactions Sample Application

How the Transactions Sample Application Works ........................................... 5-1
The Development Process for the Transactions Sample Application .......... 5-3
  OMG IDL................................................................................................... 5-4
  The Client Application ............................................................................ 5-4
  The University Server Application ............................................................ 5-4
  The UBBCONFIG File .............................................................................. 5-5
  The ICF File ............................................................................................... 5-5
Building the Transactions Sample Application................................................. 5-6
  Copying the Files for the Transactions Sample Application into a Work
  Directory ..................................................................................................... 5-6
  Changing the Protection on the Files for the Transactions Sample Application ........................................ 5-9
  Setting the Environment Variables ............................................................ 5-10
  Initializing the University Database ........................................................... 5-10
  Loading the UBBCONFIG File .................................................................. 5-10
  Creating a Transaction Log ...................................................................... 5-11
Compiling the Transactions Sample Application.............................................. 5-12
Running the Transactions Sample Application .............................................. 5-13
  Starting the Server Application ............................................................... 5-13
  Starting the CORBA C++ Client Application ............................................. 5-14
  Starting the CORBA Java Client Application ............................................. 5-14
  Starting the ActiveX Client Application ................................................... 5-16
Using the Client Applications in the Transactions Sample Application .......... 5-17
  The CORBA C++ Client Application .......................................................... 5-17
  The CORBA Java Client Application .......................................................... 5-18
6. The Wrapper Sample Application

How the Wrapper Sample Application Works .............................................. 6-1
The Development Process for the Wrapper Sample Application .................. 6-3
    OMG IDL ................................................................. 6-3
    The Client Application ................................................. 6-4
    The Server Application ............................................... 6-4
    The UBBCONFIG File .................................................. 6-5
    The ICF File .................................................................. 6-5
Building the Wrapper Sample Application ................................................. 6-6
    Copying the Files for the Wrapper Sample Application into a Work Directory...
    6-6
    Changing the Protection on the Files for the Wrapper Sample Application ... 6-9
    Setting the Environment Variables ............................................. 6-10
    Initializing the University Database .......................................... 6-10
    Loading the UBBCONFIG File ........................................... 6-10
    Creating a Transaction Log .................................................. 6-11
Compiling the Wrapper Sample Application .............................................. 6-12
Running the Wrapper Sample Application .................................................. 6-12
    Starting the Server Application ........................................ 6-13
    Starting the CORBA C++ Client Application ......................... 6-14
    Starting the CORBA Java Client Application ......................... 6-14
    Starting the ActiveX Client Application .............................. 6-16
Using the Client Applications in the Wrapper Sample Application ............ 6-17
    The CORBA C++ Client Application ................................ 6-17
    The CORBA Java Client Application .................................. 6-18
    The ActiveX Client Application ....................................... 6-18

7. The Production Sample Application

How the Production Sample Application Works ......................................... 7-2
    Replicating Server Applications ......................................... 7-3
    Replicating Server Groups .............................................. 7-5
    Using a Stateless Object Model ........................................... 7-7
    Using Factory-based Routing ............................................. 7-7
A. Setting Up the Database

Database Support................................................................. A-1
Setup Instructions for the Oracle Database ......................... A-2
Setup for Local Database Instance ...................................... A-2
Setup for Remote Database Instance ................................. A-3

Index
About This Document

This document describes the University sample applications that are provided with the BEA WebLogic Enterprise™ software.

This document covers the following topics:

- Chapter 1, “Introduction,” provides an overview of the sample applications.
- Chapter 2, “Setting Up Your Environment,” describes the system requirements and provides information about setting up the system environment variables and parameters in the UBBCONFIG file.
- Chapter 3, “The Basic Sample Application,” describes the Basic sample application.
- Chapter 5, “The Transactions Sample Application,” describes the Transactions sample application.
- Chapter 6, “The Wrapper Sample Application,” describes the Wrapper sample application.
- Chapter 7, “The Production Sample Application,” describes the Production sample application.
- Appendix A, “Setting Up the Database,” describes how to set up the database that is used with the University sample applications.
What You Need to Know

This document is intended for application designers and programmers who would find a set of progressive examples useful in understanding the WebLogic Enterprise software.

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 WebLogic Enterprise 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 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.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface text</strong></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>
<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 filenames 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 |
| **monospace boldface 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 |
## Documentation Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ }</code></td>
<td>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</td>
</tr>
</tbody>
</table>
| `[ ]`      | 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]... |
| `|`         | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |
| `...`      | Indicates one of the following in a command line:  
- That an argument can be repeated several times in a command line  
- That the statement omits additional optional arguments  
- That you can enter additional parameters, values, or other information  
The ellipsis itself should never be typed.  
*Example:*  
buildobjclient [-v] [-o name ] [-f file-list]... 
[-l file-list]... |
| `.`        | 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 describes the University sample applications provided with the WebLogic Enterprise software. The sample applications provide client and server programmers with the basic concepts of developing distributed client/server applications using the WebLogic Enterprise software and introduces many of the more advanced features of the WebLogic Enterprise product.

This topic includes the following sections:

- An Overview of the University Sample Applications
- Naming Conventions Used in the University Sample Applications

An Overview of the University Sample Applications

The WebLogic Enterprise software includes a sample application suite based on client and server applications implemented at a university. Each University sample application demonstrates a new set of WebLogic Enterprise features while building on the experience obtained from the previous examples. The University sample applications are intentionally simplified to demonstrate only the steps and processes associated with using a particular feature of the WebLogic Enterprise product.
Table 1-1 describes the University sample applications.

### Table 1-1 The University Sample Applications

<table>
<thead>
<tr>
<th>University Sample Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Describes how to create WebLogic Enterprise client and server applications, configure the WebLogic Enterprise application, and build and run the client and server applications included in the Basic sample application. CORBA C++, CORBA Java, and ActiveX client applications are provided as well as a C++ server application.</td>
</tr>
<tr>
<td>Security</td>
<td>Adds application-level security to the client applications in the Basic sample application and to the configuration of the WebLogic Enterprise application.</td>
</tr>
<tr>
<td>Transactions</td>
<td>Adds transactional objects to the client and server applications in the Basic sample application. The Transactions sample application demonstrates how to use the Implementation Configuration File (ICF) to define transaction policies for CORBA objects.</td>
</tr>
<tr>
<td>Wrapper</td>
<td>Demonstrates how to wrap an existing BEA Tuxedo application as a CORBA object.</td>
</tr>
<tr>
<td>Production</td>
<td>Demonstrates replicating server applications, creating stateless objects, and implementing factory-based routing in server applications.</td>
</tr>
</tbody>
</table>

Use the University sample applications in conjunction with the following manuals:

- *Getting Started*
- *Creating Client Applications*
- *Creating C++ Server Applications*
Naming Conventions Used in the University Sample Applications

The naming conventions listed and described in Table 1-2 are used in the code of the University sample applications.

Table 1-2  Naming Conventions Used in the University Sample Applications

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crs</td>
<td>The abbreviation for course.</td>
</tr>
<tr>
<td>syn</td>
<td>The abbreviation for synopsis.</td>
</tr>
<tr>
<td>det</td>
<td>The abbreviation for details.</td>
</tr>
<tr>
<td>lst</td>
<td>The abbreviation for list.</td>
</tr>
<tr>
<td>enum</td>
<td>The abbreviation for enumerator.</td>
</tr>
<tr>
<td>stu</td>
<td>The abbreviation for student.</td>
</tr>
<tr>
<td>num</td>
<td>The abbreviation for number.</td>
</tr>
<tr>
<td>cur</td>
<td>The abbreviation for current.</td>
</tr>
<tr>
<td>_oref</td>
<td>A CORBA::Object reference.</td>
</tr>
<tr>
<td>_ref</td>
<td>A typed object reference.</td>
</tr>
<tr>
<td>p_</td>
<td>The abbreviation for ptr.</td>
</tr>
<tr>
<td>v_</td>
<td>The abbreviation for var.</td>
</tr>
<tr>
<td>s_</td>
<td>The abbreviation for file static data.</td>
</tr>
<tr>
<td>m_</td>
<td>The abbreviation for class member data.</td>
</tr>
</tbody>
</table>

Method names and variable names: Use all lowercase letters for the name and underscores to separate words within the method name (for example, m_v_crs_syn_list is member data that is a var holding a course synopsis list).
1 Introduction

Table 1-2 Naming Conventions Used in the University Sample Applications

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type names</td>
<td>Start with an uppercase letter and use an uppercase letter to separate words with a type name. Type names do not use abbreviations. An example of a type name is UniversityB::CourseSynopsisEnumerator_var.</td>
</tr>
</tbody>
</table>
CHAPTER

2 Setting Up Your Environment

This topic describes how to configure your WebLogic Enterprise (WLE) application so that you can run the University sample applications.

This topic includes the following sections:

- System Prerequisites
- Editing the setenv and UBBCONFIG Files

A copy of the `Readme.txt` file with troubleshooting information and the latest information about setting up the sample applications is included in each sample application directory.

For information about setting up a database so that it can be used with the University sample applications, see Appendix A, “Setting Up the Database.”

System Prerequisites

For information about the operating system platforms supported by the product, see the *BEA WebLogic Enterprise Installation Guide*.

To run the client applications in the University sample applications, you need the following development tools:

- Visual C++ version 5.0 with Service Pack 3
Setting Up Your Environment

- If you want to run the ActiveX client application, Visual Basic version 5.0 with Service Pack 3
- If you want to run the CORBA Java client application, Java Development Kit (JDK) version 1.1.6 or greater or JDK Version 1.2
  - The UNIX platforms supported by version 4.2 of the WebLogic Enterprise software use the JDK Version 1.1.6 or greater
  - The Windows NT platforms supported by version 4.2 of the WebLogic Enterprise software use the JDK version 1.2

Editing the setenv and UBBCONFIG Files

You need to set several parameters in the setenv and UBBCONFIG files in order for the University sample applications to work properly, as follows:

- The setenv file sets the system environment variables needed to build and run the sample applications. Each sample application directory contains a unique setenv file. The name of the setenv file designates which sample application the file is to be used with. For example, setenvb is for the Basic sample application. Each sample application directory contains a setenv file for the Windows NT and UNIX operating systems. For a list of the specific filenames for the setenv file, see Table 2-1.

- The UBBCONFIG file is the configuration file for the sample application. The UBBCONFIG file defines parameters for how the client and server applications in the sample application should work. Each sample application directory contains a unique UBBCONFIG file. The name of the UBBCONFIG file designates which sample application the file is to be used with. For example, ubb_b is for the Basic sample application. Each sample application directory contains a UBBCONFIG file for the Windows NT and UNIX operating systems. For a list of the specific filenames for the UBBCONFIG file, see Table 2-1.

The information in the setenv and UBBCONFIG files must match. The following sections explain how to edit the setenv and UBBCONFIG files.
# Naming Conventions for the setenv and UBBCONFIG Files

Table 2-1 describes the naming conventions for the setenv and UBBCONFIG files.

<table>
<thead>
<tr>
<th>University Sample Application</th>
<th>Naming Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>setenvb.cmd— the setenv file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>setenvb.sh— the setenv file for UNIX</td>
</tr>
<tr>
<td></td>
<td>ubb_b.nt— the UBBCONFIG file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>ubb_b.mk— the UBBCONFIG file for UNIX</td>
</tr>
<tr>
<td>Security</td>
<td>setenvs.cmd— the setenv file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>setenvs.sh— the setenv file for UNIX</td>
</tr>
<tr>
<td></td>
<td>ubb_s.nt— the UBBCONFIG file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>ubb_s.mk— the UBBCONFIG file for UNIX</td>
</tr>
<tr>
<td>Transactions</td>
<td>setenvt.cmd— the setenv file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>setenvt.sh— the setenv file for UNIX</td>
</tr>
<tr>
<td></td>
<td>ubb_t.nt— the UBBCONFIG file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>ubb_t.mk— the UBBCONFIG file for UNIX</td>
</tr>
<tr>
<td>Wrapper</td>
<td>setenvw.cmd— the setenv file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>setenvw.sh— the setenv file for UNIX</td>
</tr>
<tr>
<td></td>
<td>ubb_w.nt— the UBBCONFIG file for Windows NT</td>
</tr>
<tr>
<td></td>
<td>ubb_w.mk— the UBBCONFIG file for UNIX</td>
</tr>
</tbody>
</table>
Setting Up Your Environment

Table 2-1 Naming Conventions for setenv and UBBCONFIG Files (Continued)

<table>
<thead>
<tr>
<th>University Sample Application</th>
<th>Naming Convention</th>
</tr>
</thead>
</table>
| Production                    | setenvp.cmd—the setenv file for Windows NT  
                                       setenvp.sh—the setenv file for UNIX  
                                       ubb_p.nt—the UBBCONFIG file for Windows NT  
                                       ubb_p.mk—the UBBCONFIG file for UNIX |

Setting setenv Parameters

Table 2-2 lists the parameters you need to modify in the setenv file.

Table 2-2 Parameters in the setenv File

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| APPDIR    | The directory path where you copied the sample application files. For example:  
             Windows NT  
             APPDIR=c:\work\university\basic  
             UNIX  
             APPDIR=/usr/work/university/basic |
| TUXCONFIG | The directory path and name of the configuration file. For example:  
             Windows NT  
             TUXCONFIG=c:\work\university\basic\tuxconfig  
             UNIX  
             TUXCONFIG=/usr/work/university/basic/tuxconfig |
| TUXDIR    | The directory path where you installed the WebLogic Enterprise software. For example:  
             Windows NT  
             TUXDIR=c:\WLEdir  
             UNIX  
             TUXDIR=/usr/local/WLEdir |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORACLE_HOME</td>
<td>The directory path where you installed the Oracle software. For example:</td>
</tr>
<tr>
<td></td>
<td>Windows NT</td>
</tr>
<tr>
<td></td>
<td>ORADIR=c:\Orant</td>
</tr>
<tr>
<td></td>
<td>UNIX</td>
</tr>
<tr>
<td></td>
<td>ORACLE_HOME=/usr/local/oracle</td>
</tr>
<tr>
<td>NETSCAPE</td>
<td>The directory path where you installed the Netscape Enterprise Server software. For example:</td>
</tr>
<tr>
<td></td>
<td>Windows NT</td>
</tr>
<tr>
<td></td>
<td>NETSCAPE=c:\Netscape\SuiteSpot</td>
</tr>
<tr>
<td></td>
<td>UNIX</td>
</tr>
<tr>
<td></td>
<td>NETSCAPE=/usr/local/netscape/suitespot</td>
</tr>
<tr>
<td></td>
<td>You need to specify this parameter only if you plan to use the CORBA Java client application.</td>
</tr>
<tr>
<td>JARTYPE</td>
<td>Specifies the version of the JDK being used.</td>
</tr>
<tr>
<td></td>
<td>Windows NT</td>
</tr>
<tr>
<td></td>
<td>JARTYPE = JDK or jdk</td>
</tr>
<tr>
<td></td>
<td>UNIX</td>
</tr>
<tr>
<td></td>
<td>JARTYPE = JDK or jdk</td>
</tr>
<tr>
<td></td>
<td>If you do not specify the JARTYPE parameter, it is assumed you are using the JDK version 1.1. You need to specify this parameter only if you plan to use the CORBA Java sample application.</td>
</tr>
<tr>
<td>JDKDIR</td>
<td>The directory path where you installed the JDK software. For example:</td>
</tr>
<tr>
<td></td>
<td>Windows NT</td>
</tr>
<tr>
<td></td>
<td>JDKDIR=c:\JDK1.2</td>
</tr>
<tr>
<td></td>
<td>UNIX</td>
</tr>
<tr>
<td></td>
<td>JDKDIR=/usr/local/jdk1.1.6</td>
</tr>
<tr>
<td></td>
<td>You need to specify this parameter only if you plan to use the CORBA Java sample application.</td>
</tr>
<tr>
<td>TOBJADDR</td>
<td>If you are using a CORBA C++ client application that does not reside on the same machine as the server application, enter the host and port of the machine where the server application runs. It must be specified exactly (including case) as it appears in the UBBCONFIG file for the machine. For example: //BEANIE:2500</td>
</tr>
</tbody>
</table>
### Setting the UBBCONFIG Parameters

Table 2-3 lists the parameters you need to modify in the UBBCONFIG file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>If you are using a remote instance of the Oracle database, the format is as follows: USERID=username/password@aliasname. This is the same information you defined when you set up a remote instance of the Oracle database. If you are using a local instance of the Oracle database, the format is as follows: USERID=username/password.</td>
</tr>
<tr>
<td>ORACLE_SID</td>
<td>The instance ID of the Oracle database. On Windows NT, you do not need to specify the ORACLE_SID, the parameter automatically defaults to ORCL.</td>
</tr>
<tr>
<td>CCMPL</td>
<td>The directory location of the C compiler. This parameter is set to a typical installation directory. Verify that your installation matches this directory location and change the location if necessary. This parameter applies only to the UNIX operating system.</td>
</tr>
<tr>
<td>CPPCMPL</td>
<td>The directory location of the C++ compiler. This parameter is set to a typical installation directory. Verify that your installation matches this directory location and change the location if necessary. This parameter applies only to the UNIX operating system.</td>
</tr>
<tr>
<td>CPPINC</td>
<td>The directory location of the C++ include directory. This parameter is set to a typical installation directory. Verify that your installation matches this directory location and change the location if necessary. This parameter applies only to the UNIX operating system.</td>
</tr>
<tr>
<td>SHLIB_PATH, LD_LIBRARY_PATH, or LIBPATH</td>
<td>The directory location of the shared library. This parameter is set to a typical installation directory. Verify that your installation matches this directory location and change the location if necessary. This parameter applies only to the UNIX operating system.</td>
</tr>
<tr>
<td>PROC</td>
<td>The directory location of the Oracle Programmer C/C++ SQL Precompiler. You only need to specify this parameter if you are using the Windows NT operating system.</td>
</tr>
<tr>
<td>PRODIR</td>
<td>The directory location of the Oracle Programmer C/C++ SQL Precompiler. You only need to specify this parameter if you are using the Windows NT operating system.</td>
</tr>
</tbody>
</table>
### Table 2-3 Parameters in the UBBCONFIG File

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| MY_SERVER_MACHINE | Delete this parameter and replace it with the name of the server machine.  
On Windows NT, you can obtain the server machine name by entering the following command at the MS-DOS prompt:  
  set COMPUTERNAME  
On UNIX, you can obtain the server machine name by entering the following command at the shell prompt:  
  prompt>uname -n  
You must enter the server machine name exactly (including case) as it appears in the output of the command.  
Specify the server machine name as it appears. For example, BEANIE.  
Full names must be included in quotation marks. For example:  
  "beanie.bea.com". |
| APPDIR | The full directory path where you copied the sample application files. The directory path needs to be included in quotation marks. For example:  
  **Windows NT**  
  APPDIR="c:\work\university\basic"  
  **UNIX**  
  APPDIR="/usr/work/university/basic"  
This parameter needs to match the APPDIR parameter in the *setenv* file. |
| TUXCONFIG | The full directory path of the configuration file. This is the subdirectory of the sample application. The directory path needs to be included in quotation marks. For example:  
  **Windows NT**  
  TUXCONFIG="c:\work\university\basic\tuxconfig"  
  **UNIX**  
  TUXCONFIG="/usr/work/university/basic/tuxconfig"  
This parameter needs to match the TUXCONFIG parameter in the *setenv* file. |
Setting Up Your Environment

Table 2-3 Parameters in the UBBCONFIG File (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUXDIR</td>
<td>The full directory path where you installed the WebLogic Enterprise software. The directory path needs to be included in quotation marks. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>TUXDIR=&quot;c:\M3dir&quot;</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>TUXDIR=&quot;/usr/local/M3dir&quot;</td>
</tr>
<tr>
<td></td>
<td>This parameter needs to match the TUXDIR parameter in the setenv file.</td>
</tr>
<tr>
<td>CLOPT for the ISL process</td>
<td>Enter the host name and port number of the machine on which the server application is installed. For example:</td>
</tr>
<tr>
<td></td>
<td>ISL</td>
</tr>
<tr>
<td></td>
<td>SRVGRP = SYS_GRP</td>
</tr>
<tr>
<td></td>
<td>SRVID =</td>
</tr>
<tr>
<td></td>
<td>CLOPT = &quot;-A --n //BEANIE:2500&quot;</td>
</tr>
<tr>
<td>OPENINFO</td>
<td>If you are using the Transactions, Wrapper, or Production sample applications, you need to specify this parameter for the Oracle database.</td>
</tr>
<tr>
<td></td>
<td>If you are using a remote instance of the Oracle database, the OPENINFO parameter is specified as follows:</td>
</tr>
<tr>
<td></td>
<td>OPENINFO = &quot;Oracle_XA:Oracle_XA+SqlNet=aliasname+Acc=P/account/pswrd+SesTM=100+LogDir=.+MaxCur=5&quot;</td>
</tr>
<tr>
<td></td>
<td>For example, on Windows NT:</td>
</tr>
<tr>
<td></td>
<td>OPENINFO = &quot;Oracle_XA:Oracle_XA+SqlNet=ORCL+Acc=P/scott/tiger+SesTM=100+LogDir=.+MaxCur=5&quot;</td>
</tr>
<tr>
<td></td>
<td>If you are using a local instance of the Oracle database, the OPENINFO parameter is specified as follows:</td>
</tr>
<tr>
<td></td>
<td>OPENINFO = &quot;Oracle_XA:Oracle_XA+Acc=P/account/pswrd+SesTM=100+LogDir=.+MaxCur=5&quot;</td>
</tr>
<tr>
<td></td>
<td>For example, on Windows NT:</td>
</tr>
<tr>
<td></td>
<td>OPENINFO = &quot;Oracle_XA:Oracle_XA+Acc=P/scott/tiger+SesTM=100+LogDir=.+MaxCur=5&quot;</td>
</tr>
</tbody>
</table>
Editing the setenv and UBBCONFIG Files

Running the setenv Command

Before you can use the University sample applications, you need to run the `setenv` script to ensure your system environment variables reflect all the changes made in the process of setting up the Oracle database and your configuration. Instructions for running the `setenv` command are included in the descriptions of building the individual sample applications.

**Note:** The makefiles for the University sample applications assume Microsoft Visual C++ is installed in the following location on Windows NT:

```
c:\Progra~1\Devstu~1\VC
```

If your copy of Microsoft Visual C++ is not installed in that directory, run the following command procedure to set the appropriate system environment variables.

```
c:\Progra~1\Devstu~1\VC\Bin\VCVARS32.bat
```

The Oracle Pro*C/C++ compiler uses short names so you need to use `~` in the directory path to ensure the system variable is set correctly.
2 Setting Up Your Environment
CHAPTER

3 The Basic Sample Application

This topic includes the following sections:

- How the Basic Sample Application Works
- The OMG IDL for the Basic Sample Application Generating the Client Stubs and the Skeletons
- Generating the Client Stubs and the Skeletons
- Writing the Client Application
- Configuring the Basic Sample Application
- Building the Basic Sample Application
- Compiling the Basic Sample Application
- Running the Basic Sample Application
- Using the Client Applications in the Basic Sample Application

Refer to README.txt in the basic directory for troubleshooting information and the latest information about using the Basic sample application.

For an explanation of concepts associated with WebLogic Enterprise applications and a description of the development process for WebLogic Enterprise applications, see Getting Started.
The Basic sample application allows users to browse for available courses and get details on selected courses. Figure 3-1 illustrates how the Basic sample application works.

Figure 3-1  The Basic Sample Application

The Basic sample application demonstrates the following features:

- Creating WebLogic Enterprise client and server applications
- Defining the configuration for a WebLogic Enterprise application
- Building client and server applications using the commands and tools provided by the WebLogic Enterprise software
The OMG IDL for the Basic Sample Application

The first step in creating client and server applications is to specify all of the CORBA interfaces and their methods using OMG IDL. The Basic sample application implements the following CORBA interfaces:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistrarFactory</td>
<td>Creates object references to the Registrar object.</td>
<td>find_registrar()</td>
</tr>
<tr>
<td>Registrar</td>
<td>Obtains course information from the database.</td>
<td>get_courses_synopsis()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_courses_details()</td>
</tr>
<tr>
<td>CourseSynopsisEnumerator</td>
<td>Gets synopses of courses that match the search criteria from the course database and reads them into memory; returns the first subset of the synopses to the Registrar object, which in turns returns them to the client application; and provides a means for a client application to retrieve the remainder of the synopses.</td>
<td>get_next_n()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>destroy()</td>
</tr>
</tbody>
</table>

Listing 3-1 shows the univb.idl file that defines the CORBA interfaces in the Basic sample application. A copy of this file is included in the directory for the Basic sample application.

**Listing 3-1  OMG IDL for the Basic Sample Application**

```idl
module UniversityB
{
    typedef unsigned long CourseNumber;
    typedef sequence<CourseNumber> CourseNumberList;
}  
```
struct CourseSynopsis
{
    CourseNumber   course_number;
    string         title;
};

typedef sequence<CourseSynopsis> CourseSynopsisList;

interface CourseSynopsisEnumerator
{
    CourseSynopsisList get_next_n(
        in unsigned long number_to_get,
        out unsigned long number_remaining
    );
    void destroy();
};
tytypedef unsigned short Days;
const Days MONDAY    =  1;
const Days TUESDAY   =  2;
const Days WEDNESDAY =  4;
const Days THURSDAY  =  8;
const Days FRIDAY    = 16;

struct ClassSchedule
{
    Days           class_days; // bitmask of days
    unsigned short start_hour; // whole hours in military time
    unsigned short duration;   // minutes
};

struct CourseDetails
{
    CourseNumber   course_number;
    double         cost;
    unsigned short number_of_credits;
    ClassSchedule  class_schedule;
    unsigned short number_of_seats;
    string         title;
    string         professor;
    string         description;
};

typedef sequence<CourseDetails> CourseDetailsList;

interface Registrar
{
    CourseSynopsisList
    get_courses_synopsis(
        in string                    search_criteria,
        out unsigned long number_of_results,
        out unsigned long number_remaining
    );
    void destroy();
};
Generating the Client Stubs and the Skeletons

in unsigned long number_to_get, // 0 = all
out unsigned long number_remaining,
out CourseSynopsisEnumerator rest
);
}

Generating the Client Stubs and the Skeletons

Note: The CORBA client applications in the University sample applications use static invocation. For an example of using the dynamic invocation interface, see Creating Client Applications. When creating CORBA Java client applications, see your Java ORB's documentation for information about compiling the OMG IDL to get client stubs. ActiveX client applications do not use client stubs.

The interface specification defined in OMG IDL is used by the IDL compiler to generate client stubs for the client application and skeletons for the server application. The client stubs are used by the client application for all operation invocations. You use the skeleton, along with the code you write, to create the server application that implements the CORBA objects. For information about generating and using client stubs and skeletons, see Getting Started.

During the development process, you would use the \texttt{idl} command to compile the OMG IDL file and produce client stubs and skeletons. This task has been automated in the \texttt{makefile} for the Basic sample application. For a description of the \texttt{idl} command, see Commands, System Processes, and MIB Reference.
Writing the Client Application

The WebLogic Enterprise software supports three types of client applications:

- CORBA C++
- CORBA Java
- ActiveX

**Note:** The ActiveX client application is written in Visual Basic.

During the development process, you would write client application code that does the following:

- Initializes the ORB
- Uses the Bootstrap environmental object to establish communication with the WebLogic Enterprise domain
- Resolves initial references to the FactoryFinder environmental object
- Uses a factory to get an object reference for the Registrar object
- Invokes the `get_courses_synopsis()` and `get_courses_details()` methods on the Registrar object

C++, Java, and Visual Basic versions of the client application code in the Basic sample application are provided. For information about writing client applications, see *Getting Started* and *Creating Client Applications*.

Writing the Server Application

During the development process, you would write the following:

- The Server object that initializes the University server application and registers a factory for the Registrar object with the WebLogic Enterprise domain.

- The method implementations for the operations on the Registrar, RegistrarFactory, and CourseSynopsisEnumerator objects.
Configuring the Basic Sample Application

C++ code for the Server object and the method implementations in the University server application are provided.

During the development process, you use the `genicf` command to create an Implementation Configuration File (ICF). You then edit the ICF file to define activation and transaction policies for the Registrar, RegistrarFactory, and CourseSynopsisEnumerator objects. For the Basic sample application, the Registrar, RegistrarFactory, and CourseSynopsisEnumerator objects have an activation policy of `process` and a transaction policy of `ignore`. An ICF file for the Basic sample application is provided.

For information about writing server applications, see *Creating C++ Server Applications*.

## Configuring the Basic Sample Application

A key part of any WebLogic Enterprise application is the `UBBCONFIG` file. Although creating a `UBBCONFIG` file is the task of the administrator, it is important for the client and server programmers to understand that the file exists and how the file is used. When system administrators create a configuration file, they are describing the WebLogic Enterprise application using a set of parameters that the WebLogic Enterprise software interprets to create a runnable application.

There are two forms of the configuration file:

- The `UBBCONFIG` file, an ASCII version of the file, created and modified with any editor. Chapter 2, “Setting Up Your Environment,” describes setting the required parameters in the `UBBCONFIG` file used by all University sample applications.

- The `TUXCONFIG` file, a binary version of the `UBBCONFIG` file created using the `tmloadcf` command. When the `tmloadcf` command is executed, the environment variable `TUXCONFIG` must be set to the name and directory location of the `TUXCONFIG` file.

For information about the `UBBCONFIG` file and the `tmloadcf` command, see *Administration Guide* and *Commands, System Processes, and MIB Reference*. 

Guide to the University Sample Applications 3-7
Building the Basic Sample Application

To build the Basic sample application, complete the following steps:

1. Copy the files for the Basic sample application into a work directory.
2. Change the protection on the files for the Basic sample application.
3. Set the environment variables.
4. Initialize the University database.
5. Load the UBBCONFIG file.
6. Build the client and server sample applications.

The following sections describe these steps.

**Note:** Before you can build or run the Basic sample application, you need to complete the steps in Chapter 2, “Setting Up Your Environment.”

Copying the Files for the Basic Sample Application into a Work Directory

The files for the Basic sample application are located in the following directories:

**Windows NT**

```
\drive:WLEdir\samples\corba\university\basic
```

**UNIX**

```
/usr/WLEdir/samples/corba/university/basic
```

In addition, you need to copy the `utils` directory into your work directory. The `utils` directory contains files that set up logging, tracing, and access to the University database.

Table 3-1 lists and describes the files you will use to create the Basic sample application.
Table 3-1  Files Included in the Basic Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>univb.idl</td>
<td>The OMG IDL that declares the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univbs.cpp</td>
<td>The C++ source code for the University server application in the Basic sample application.</td>
</tr>
<tr>
<td>univb_i.h univb_i.cpp</td>
<td>The C++ source code for method implementations of the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univbc.cpp</td>
<td>The C++ source code for the CORBA C++ client application in the Basic sample application.</td>
</tr>
<tr>
<td>frmBrowser.frm frmBrowser.frx</td>
<td>The Visual Basic source code for the ActiveX client application in the Basic sample application.</td>
</tr>
<tr>
<td>modPublicDeclarations.bas</td>
<td>A Visual Basic file that contains the declarations for variables used in the sample applications.</td>
</tr>
<tr>
<td>frmTracing.frm frmTracing.frx</td>
<td>The files that provide tracing capabilities to the ActiveX client application.</td>
</tr>
<tr>
<td>University.vbp University.vbw</td>
<td>The Visual Basic project file for the ActiveX client application in the Basic sample application.</td>
</tr>
<tr>
<td>UnivBApplet.java</td>
<td>The Java source code for the CORBA Java client application in the Basic sample application.</td>
</tr>
<tr>
<td>univb_utils.h univb_utils.cpp</td>
<td>The files that define database access functions for the CORBA C++ client application.</td>
</tr>
<tr>
<td>univb.icf</td>
<td>The Implementation Configuration File (ICF) for the Basic sample application.</td>
</tr>
<tr>
<td>setenvb.sh</td>
<td>A UNIX script that sets the environment variables needed to build and run the Basic sample application.</td>
</tr>
</tbody>
</table>
### Table 3-1  Files Included in the Basic Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setenvb.cmd</td>
<td>An MS-DOS command procedure that sets the environment variables needed to build and run the Basic sample application.</td>
</tr>
<tr>
<td>ubb_b.mk</td>
<td>The configuration file for the UNIX operating system platform.</td>
</tr>
<tr>
<td>ubb_b.nt</td>
<td>The configuration file for the Windows NT operating system platform.</td>
</tr>
<tr>
<td>makefileb.mk</td>
<td>The makefile for the Basic sample application on the UNIX operating system platform.</td>
</tr>
<tr>
<td>makefileb.nt</td>
<td>The makefile for the Basic sample application on the Windows NT operating system platform.</td>
</tr>
<tr>
<td>log.cpp, log.h, log_client.cpp, and log_server.cpp</td>
<td>The client and server applications that provide logging and tracing functions for the sample applications. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>oradbconn.cpp and oranoconn.cpp</td>
<td>The files that provide access to an Oracle SQL database instance. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>samplesdb.cpp and samplesdb.h</td>
<td>The files that provide print functions for the database exceptions in the sample applications. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>unique_id.cpp and unique_id.h</td>
<td>C++ Unique ID class routines for the sample applications. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>samplesdbsql.h and samplesdbsql.pc</td>
<td>C++ class methods that implement access to the SQL database. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>university.sql</td>
<td>The SQL for the University database. This file is located in the \utils directory.</td>
</tr>
</tbody>
</table>
Changing the Protection on the File for the Basic Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit the files or build the files in the Basic sample application, you need to change the protection of the files you copied into your work directory, as follows:

**Windows NT**

```prompt
attrib -r drive:\workdirectory\*.*
```

**UNIX**

```prompt
chmod u+rw /workdirectory/*.*
```

Setting the Environment Variables

Use the following command to set the environment variables used to build the client and server applications in the Basic sample application:

**Windows NT**

```prompt
setenvb
```

**UNIX**

```prompt
/bin/ksh
```

```prompt
. ./setenvb.sh
```

Initializing the University Database

Use the following command to initialize the University database used with the Basic sample application:

**Windows NT**

```prompt
nmake -f makefileb.nt initdb
```
UNIX
prompt>make -f makefileb.mk initdb

Loading the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

Windows NT
prompt>tmloadcf -y ubb_b.nt
UNIX
prompt>tmloadcf -y ubb_b.mk

Compiling the Basic Sample Application

During the development process, you would use the buildobjclient and buildobjserver commands to build the client and server applications. However, for the Basic sample application, this step has been done for you.

The directory for the Basic sample application contains a makefile that builds the client and server sample applications.

Use the following commands to build the CORBA C++ client and server applications in the Basic sample application:

Windows NT
prompt>nmake -f makefileb.nt
UNIX
prompt>make -f makefileb.mk

To build the CORBA Java client application:

Windows NT
prompt>nmake -f makefileb.nt javaclient
Running the Basic Sample Application

To run the Basic sample application, complete the following steps:

1. Start the University server application.
2. Start one or more of the client applications.

Starting the Server Application

Start the system and sample application server applications in the Basic sample application by entering the following command:

```
prompt>tmboot -y
```

This command starts the following server processes:

- **TMSYSEVT**
  The BEA Tuxedo system event broker.

- **TMFFNAME**
  The transaction management services, including the NameManager and the FactoryFinder services.

- **TMIFSRVR**
  The Interface Repository server process. This server process is used only by ActiveX client applications.

UNIX

```
prompt>make -f makefileb.mk javaclient
```

For information about building and using the ActiveX client application, see “Starting the ActiveX Client Application.”

For more information about the `buildobjclient` and `buildobjserver` commands, see the *Commands, System Processes, and MIB Reference*. 
The Basic Sample Application

- `univb_server`
  
  The University server process.

- `ISL`
  
  The IIOP Listener/Handler process.

Before using another sample application, enter the following command to stop the system and sample application server processes:

```
prompt>tmshutdown
```

Starting the CORBA C++ Client Application

Start the CORBA C++ client application in the Basic sample application by entering the following command:

```
prompt>univb_client
```

Starting the CORBA Java Client Application

To run the CORBA Java client application in the Basic sample application, complete the following steps:

1. Modify the following lines in the `UnivBApplet.html` file:

   ```html
   code="UnivBApplet.class"
   codebase=.
   ```

   to read as follows:

   ```html
   code="UnivBApplet"
   archive="UnivBApplet.jar,m3envobj.jar"
   ```

2. Copy the modified `UnivBApplet.html` file to the source directory for the Web server (the directory varies by Web server product).

3. After executing the `makefile` to build the Basic sample application, create a `UnivBApplet.jar` file, as follows:
Running the Basic Sample Application

a. Create a tmp directory under the directory where you built the sample application and copy the UniversityB subdirectory and the class files it contains into the tmp directory.

Copy the class files in the Basic sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to the tmp directory, and issue one of the following commands to create a jar file that contains all the Basic sample application classes:

```bash
jar -cf ..\UnivBApplet.jar *.* (Microsoft Windows NT systems)
jar -cf ../UnivBApplet.jar * (UNIX systems)
```

4. Copy the UnivBApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or $TUXDIR$/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Basic server application is running, start up your Web browser, and point it to the node where the Web server is running.

Note: On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivBApplet.html file, set your browser to http://SERVER/UnivBApplet.html.

Starting the ActiveX Client Application

Note: For the University sample applications, the task of loading the OMG IDL for the CORBA interfaces into the Interface Repository is automated by the makefile.

Before you can start the ActiveX client application, you must use the Application Builder to create ActiveX bindings for the CORBA interfaces.

To create an ActiveX binding for a CORBA interface, complete the following steps:

1. Click the BEA Application Builder icon in the WebLogic Enterprise program group.
The Domain logon window appears.

2. In the Domain Logon window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must match exactly the capitalization used in the UBBCONFIG file. For example: //BEANIE:2500.

The Application Builder logon window appears.

3. Highlight the UniversityB folder in the Services window and drag it to the Workstation Views window, or copy the UniversityB folder from the Services window and paste it into the Workstation Views window.

A confirmation window appears.

4. Click Create to create the ActiveX bindings for the CORBA interfaces in the Basic sample application.

The Application Builder creates the following:

- A binding for the CORBA interface. The binding is named DI modulename_interfacename. For example, the binding for the Registrar interface is named DIUniversityB_Registrar.

- A type library. By default, the type library is placed in \WLEdir\TypeLibraries.

  The type library file is named DI modulename_interfacename.tlb.

- A Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

To open the ActiveX client application, complete the following steps:

1. Open the University project in Visual Basic.

2. Run the University project.

3. From the Run menu, click Start.

   A logon window appears.

4. In the Logon window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must match exactly the capitalization used in the UBBCONFIG file.
Using the Client Applications in the Basic Sample Application

The following sections briefly explain how to use the client applications that are included in the Basic sample application.

The CORBA C++ Client Application

After starting the CORBA C++ client application, a menu with the following options appears:

<F>   Find courses
<A>   List all courses
<D>   Display course details
<E>   Exit

To find courses that match a particular curriculum subject, complete the following steps:

1. At the Options prompt, enter F.
2. Enter a text string at the Enter search string: prompt. For example, computer. You can enter any combination of uppercase and lowercase letters.
   A list of all the courses that match that search string appears.

To list all the courses in the database, complete the following steps:

1. At the Options prompt, enter A.
   A list of ten courses appears.
2. Enter y to continue viewing lists of ten courses or n to return to the Options menu.

To display the details of a particular course, complete the following steps:

1. At the Options prompt, enter D.
2. Enter a course number followed by -1 at the Course Number prompt. For example:
   100011
   100039
   -1

   A summary of that course appears.

To exit the C++ CORBA client application, enter E at the Options prompt.

The CORBA Java Client Application

To find courses that match a particular curriculum subject, complete the following steps:

1. In the text box under the search string? prompt, enter a text string. You can enter the title of a course, the name of a professor, or the description of a course. For example, computer.

2. Click the Show button.

   A list of all the courses that match that search string appears.

To list all the courses in the database, complete the following steps:

1. Place your cursor in the Course Name Search String text box.

2. Press Enter.

   A list of all the courses in the course database appears.

To display the details of a particular course, complete the following steps:

1. Select a course in the Course Number/Course Name window.

2. Click the Details button.

   A summary of details for the selected course appears.

To exit the CORBA Java client application, choose Quit from the Applet menu.
Using the Client Applications in the Basic Sample Application

The ActiveX Client Application

When you log on to the ActiveX client application, the Course Browser window appears. Use the Course Browser window to find courses available at the university.

To find courses that match a particular curriculum subject, complete the following steps:

1. In text box next to the Find Courses button, enter a text string or use the pulldown menu to choose a curriculum subject. For example, computer.

2. Click the Find Courses button.

A list of all the courses that match that search string appears.

To display the details of a particular course, complete the following steps:

1. Select a course in the window next to the Get Details button.

2. Click the Get Details button or double click the course name.

A summary of details for the selected course appears.

3. To enter the course into the schedule, double click the course name.

To exit the ActiveX client application, choose Exit from the File menu.
The Basic Sample Application
CHAPTER

4 The Security Sample Application

This topic includes the following sections:

- How the Security Sample Application Works
- The Development Process for the Security Sample Application
- Building the Security Sample Application
- Compiling the Security Sample Application
- Running the Security Sample Application
- Using the Client Applications in the Security Sample Application

Refer to Readme.txt in the \security directory for troubleshooting information and the latest information about using the Security sample application. For a full discussion of implementing security in a WebLogic Enterprise application, see Using Security.

How the Security Sample Application Works

The Security sample application enhances the Basic sample application by adding application-level security to the WebLogic Enterprise (WLE) application. Application-level security requires each student to have an ID and a password. Therefore, the concept of a Student is added to the Security sample application.

The following functionality is added to the Basic sample application:
The client applications add a logon operation. This operation uses a SecurityCurrent environmental object to invoke operations on the PrincipalAuthenticator object, which is part of the process of logging on to access the domain.

The University server application implements an additional operation, get_student_details(), on the Registrar object to return information about a student. After a proper CORBA logon is complete, the get_student_details() operation accesses the student information in the database to obtain the student information needed by the client logon operation.

The University database contains student information in addition to course information.

Figure 4-1 illustrates how the Security sample application works.

Figure 4-1  The Security Sample Application
The Development Process for the Security Sample Application

This section describes the development process required when adding security to WebLogic Enterprise client and server applications. These steps are in addition to the development steps outlined in Chapter 3, “The Basic Sample Application.”

**Note:** The steps in this section have been done for you and are included in the Security sample application.

**OMG IDL**

During the development process, you would define the `StudentDetails` struct and the `get_student_details()` operation in Object Management Group (OMG) Interface Definition Language (IDL).

**The Client Application**

During the development process, you would add the following code to your client application:

- The Bootstrap environmental object to obtain a reference to the SecurityCurrent environmental object in the specified WebLogic Enterprise domain
- The `Tobj::PrincipalAuthenticator` operation of the SecurityCurrent environmental object to return the type of authentication expected by the WebLogic Enterprise domain
- Operations to log on to the WebLogic Enterprise domain using the required security information

For the Security sample application, this code has already been added for you. For information about adding security to client applications, see *Using Security.*
The Server Application

During the development process, you would write the method implementation for the `get_student_details()` operation. For information about writing method implementations, see *Creating C++ Server Applications*.

The UBBCONFIG File

In the WebLogic Enterprise software, security levels are defined for the configuration by the system administrator. The system administrator defines the security for the WebLogic Enterprise domain by setting the `SECURITY` parameter `RESOURCES` section of the `UBBCONFIG` file to the desired security level. In the Security sample application, the `SECURITY` parameter is set to `APP_PW` for application-level security. For information about adding security to a WebLogic Enterprise domain, see *Administration Guide* and *Using Security*.

The ICF File

No changes to the Implementation Configuration File (ICF) are required.

Building the Security Sample Application

To build the Security sample application, complete the following steps:

1. Copy the files for the Security sample application.
2. Change the protection on the files for the Security sample application.
3. Set the environment variables.
4. Initialize the University database.
5. Load the `UBBCONFIG` file.
6. Build the client and server sample applications. 

The following sections describe these steps.

Note: Before you can build or run the Security sample application, you need to perform the steps in Chapter 2, “Setting Up Your Environment.”

**Copying the Files for the Security Sample Application into a Work Directory**

The files for the Security sample application are located in the following directories:

**Windows NT**

drive:\WLEdir\samples\corba\university\security

**UNIX**

/usr/WLEdir/samples/corba/university/security

In addition, you need to copy the `utils` directory into your work directory. The `utils` directory contains files that set up logging, tracing, and access to the University database.

You will use the files listed in Table 4-1 to create the Security sample application.

**Table 4-1 Files Included in the Security Sample Application**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>univs.idl</td>
<td>The OMG IDL that declares the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univs.cpp</td>
<td>The C++ source code for the University server application in the Security sample application.</td>
</tr>
<tr>
<td>univs_i.h</td>
<td>The C++ source code for method implementations of the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univs_i.cpp</td>
<td>The C++ source code for method implementations of the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univsc.cpp</td>
<td>The C++ source code for the CORBA C++ client application in the Security sample application.</td>
</tr>
</tbody>
</table>
### Table 4-1 Files Included in the Security Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmBrowser.frm</td>
<td>The Visual Basic source code for the ActiveX client application in the Security sample application.</td>
</tr>
<tr>
<td>frmOpen.frm</td>
<td>The Visual Basic source code for the ActiveX client application in the Security sample application.</td>
</tr>
<tr>
<td>University.vbp</td>
<td>The Visual Basic project file for the ActiveX client application in the Security sample application.</td>
</tr>
<tr>
<td>University.vbw</td>
<td>The Visual Basic workspace file for the ActiveX client application in the Security sample application.</td>
</tr>
<tr>
<td>modPublicDeclarations.bas</td>
<td>A Visual Basic file that contains the declarations for variables used in the sample applications.</td>
</tr>
<tr>
<td>frmTracing.frm, frmTracing.frx</td>
<td>The files that provide tracing capabilities to the ActiveX client application.</td>
</tr>
<tr>
<td>frmLogon.frm</td>
<td>The Visual Basic file that performs the security logon for the ActiveX client application.</td>
</tr>
<tr>
<td>UnivSApplet.java</td>
<td>The Java source code for the CORBA Java client application in the Security sample application.</td>
</tr>
<tr>
<td>univs_utils.h, univs_utils.cpp</td>
<td>The files that define database access functions for the CORBA C++ client application.</td>
</tr>
<tr>
<td>univs.icf</td>
<td>The Implementation Configuration File (ICF) for the Security sample application.</td>
</tr>
<tr>
<td>setenvs.sh</td>
<td>A UNIX script that sets the environment variables needed to build and run the Security sample application.</td>
</tr>
<tr>
<td>setenvs.cmd</td>
<td>An MS-DOS command procedure that sets the environment variables needed to build and run the Security sample application.</td>
</tr>
<tr>
<td>ubb_s.mk</td>
<td>The UBBCONFIG file for the UNIX operating system.</td>
</tr>
<tr>
<td>ubb_s.nt</td>
<td>The UBBCONFIG file for the Windows NT operating system.</td>
</tr>
</tbody>
</table>
Changing the Protection on the Files for the Security Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit the files or build the files in the Security sample application, you need to change the protection of the files you copied into your work directory, as follows:

Table 4-1  Files Included in the Security Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>makefiles.mk</td>
<td>The makefile for the Security sample application on the UNIX operating system.</td>
</tr>
<tr>
<td>makefiles.nt</td>
<td>The makefile for the Security sample application on the Windows NT operating system.</td>
</tr>
<tr>
<td>log.cpp, log.h, log_client.cpp, and log_server.cpp</td>
<td>The client and server applications that provide logging and tracing functions for the sample applications. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>oradbconn.cpp and oranoconn.cpp</td>
<td>The files that provide access to an Oracle SQL database instance. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>samplesdb.cpp and samplesdb.h</td>
<td>The files that provide print functions for the database exceptions in the sample applications. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>unique_id.cpp and unique_id.h</td>
<td>C++ Unique ID class routines for the sample applications. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>samplesdbsql.h and samplesdbsql.pc</td>
<td>C++ class methods that implement access to the SQL database. These files are located in the \utils directory.</td>
</tr>
<tr>
<td>university.sql</td>
<td>The SQL for the University database. This file is located in the \utils directory.</td>
</tr>
</tbody>
</table>
Setting the Environment Variables

Use the following command to set the environment variables used to build the client and server applications in the Security sample applications:

Windows NT
prompt>setenvs

UNIX
prompt>/bin/ksh
prompt>./setenvs.sh

Initializing the University Database

Use the following command to initialize the University database used with the Security sample application:

Windows NT
prompt>nmake -f makefiles.nt initdb

UNIX
prompt>make -f makefiles.mk initdb

Loading the UBBCONFIG File

Use the following command to load the UBBCONFIG file:
Compiling the Security Sample Application

Windows NT
prompt>tmloadcf -y ubb_s.nt

UNIX
prompt>tmloadcf -y ubb_s.mk

The build process for the UBBCONFIG file prompts you for an application password. This password will be used to log on to the client applications. Enter the password and press Enter. You are then prompted to verify the password by entering it again.

Compiling the Security Sample Application

During the development process, you would use the buildobjclient and buildobjserver commands to build the client and server applications. However, for the Security sample application, this step has been done for you.

The directory for the Security sample application contains a makefile that builds the client and server sample applications.

Use the following commands to build the CORBA C++ client and server applications in the Security sample application:

Windows NT
prompt>nmake -f makefiles.nt

UNIX
prompt>make -f makefiles.mk

To build the CORBA Java client application:

Windows NT
prompt>nmake -f makefiles.nt javaclient

UNIX
prompt>make -f makefiles.mk javaclient
Running the Security Sample Application

To run the Security sample application, complete the following steps:

1. Start the University server application.
2. Start one or more of the client applications.

These steps are explained in the following sections.

Starting the University Server Application

Start the system and sample application server applications in the Security sample application by entering the following command:

```
prompt>tmboot -y
```

This command starts the following server processes:

- **TMSYSEVT**
  - The BEA Tuxedo system event broker.
- **TMFFNAME**
  - The transaction management services, including the NameManager and the FactoryFinder services.
- **TMIFSRVR**
  - The Interface Repository server process. This server process is used only by ActiveX client applications.
- **univs_server**
The University server process.

- **ISL**
  The IIOP Listener/Handler process.

Before using another sample application, enter the following command to stop the system and sample application server processes:

```
prompt>tmshutdown
```

## Starting the CORBA C++ Client Application

Start the CORBA C++ client application in the Security sample application by completing the following steps:

1. At the MS-DOS prompt, enter the following command:
   ```
prompt>univs_client
   ```
2. At the Enter student id: prompt, enter any number between 100001 and 100010.
3. Press Enter.
4. At the Enter domain password: prompt, enter the password you defined when you loaded the **UBBCONFIG** file.
5. Press Enter.

## Starting the CORBA Java Client Application

To start the CORBA Java client application in the Security sample application, complete the following steps:

1. Modify the following lines in the **UnivSApplet.html** file:
   ```html
   code="UnivSApplet.class"
   codebase=.
   ```
   to read as follows:
2. Copy the modified UnivSApplet.html file to the source directory for the Web server (the directory varies by Web server product).

3. After executing the makefile to build the Security sample application, create a UnivSApplet.jar file, as follows:
   a. Create a tmp directory under the directory where you built the sample application and copy the UniversityS subdirectory and the class files it contains into the tmp directory.

      Copy the class files in the Security sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to the tmp directory, and issue one of the following commands to create a jar file that contains all the Security sample application classes:

      ```
      jar -cf ..\UnivSApplet.jar *.* (Microsoft Windows NT systems)
      jar -cf ../UnivSApplet.jar * (UNIX systems)
      ```

4. Copy the UnivSApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or $\{TUXDIR\}/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Security server application is running, start up your Web browser, and point it to the node where the Web server is running.

   **Note:** On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivSApplet.html file, set your browser to http://SERVER/UnivSApplet.html.

7. Enter a number between 100001 and 100010 in the student ID field.

8. Enter the password you defined when you loaded the UBBCONFIG file in the Domain Password field.

9. Click the Logon button.

10. Enter a search string to find a course.
Starting the ActiveX Client Application

**Note:** For the University sample applications, the task of loading the OMG IDL for the CORBA interfaces into the Interface Repository is automated by the *makefile*.

Before you can start the ActiveX client application, you must use the Application Builder to create ActiveX bindings for the CORBA interfaces.

To create an ActiveX binding for a CORBA interface, complete the following steps:

1. Click the BEA Application Builder icon in the WebLogic Enterprise program group.

   The IIOP Listener window appears.

2. In the IIOP Listener window, enter the host name and port number that you specified in the *ISL* parameter in the *UBBCONFIG* file. You must exactly match the capitalization used in the *UBBCONFIG* file.

   The Logon window appears.

3. In the Logon window, enter a student ID between 100001 and 100010 for the username and the password you defined when you loaded the *UBBCONFIG* file.

   The Application Builder window appears. All the CORBA interfaces loaded in the Interface Repository appear in the Services window of the Application Builder.

4. Highlight the UniversityS folder in the Services window and drag it to the Workstation Views window, or copy the UniversityS folder from the Services window and paste it into the Workstation Views window.

   A confirmation window appears.

5. Click Create to create ActiveX bindings for the CORBA interfaces in the Security sample application.

   The Application Builder creates the following:

   - A binding for the CORBA interface. The binding is named \texttt{DIModulename\_interfacename}. For example, the binding for the Registrar interface is named \texttt{DIUniversityS\_Registrar}.
A type library. By default, the type library is placed in \WLEdir\TypeLibraries.

The type library file is named DImodulename_interfacename.tlb.

A Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

To run the ActiveX client application, complete the following steps:

1. Open the University.vbw file in Visual Basic.
2. From the Run menu, click Start.
   
The IIOP Listener window appears.
3. In the IIOP Listener window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must exactly match the capitalization used in the UBBCONFIG file.
   
The Logon window appears.
4. In the Logon window, enter a student ID between 100001 and 100010 for the user name and the password you defined when you loaded the UBBCONFIG file.

Using the Client Applications in the Security Sample Application

The following sections briefly explain how to use the client applications in the Security sample application.

The CORBA C++ Client Application

The CORBA C++ client application in the Security sample application has the following additional option:

<List>
  <L> List your registered courses
</List>
Using the Client Applications in the Security Sample Application

This option displays the list of courses registered under the student ID that was used to log on to the CORBA C++ client application.

The CORBA Java Client Application

No additional functionality is added to the CORBA Java client application in the Security sample application.

The ActiveX Client Application

No additional functionality is added to the ActiveX client application in the Security sample application.
The Security Sample Application
CHAPTER 5

The Transactions Sample Application

This topic includes the following sections:

- How the Transactions Sample Application Works
- The Development Process for the Transactions Sample Application
- Building the Transactions Sample Application
- Compiling the Transactions Sample Application
- Running the Transactions Sample Application
- Using the Client Applications in the Transactions Sample Application

Refer to Readme.txt in the \transactions directory for troubleshooting information and the latest information about using the Transactions sample application. For a complete discussion of using transactions in a WebLogic Enterprise application, see Using Transactions.

How the Transactions Sample Application Works

In the Transactions sample application, students can register for classes. The operation of registering for courses is executed within the scope of a transaction. The Transactions sample application works in the following way:
1. Students submit a list of courses for which they want to be registered.

2. For each course in the list, the University server application checks whether:
   - The course is in the database.
   - The student is already registered for a course.
   - The student exceeds the maximum number of credits the student can take.

3. One of the following occurs:
   - If the course meets all the criteria, the University server application registers the student for the course.
   - If the course is not in the database or if the student is already registered for the course, the University server application adds the course to a list of registered courses for which the student could not be registered. After processing all the registration requests, the server application returns the list of courses for which registration failed. The client application prompts the student to either commit the transaction (thereby registering the student for the courses for which registration request succeeded) or to roll back the transaction (thus not registering the student for any of the courses).
   - If the student exceeds the maximum number of credits the student can take, the University server application returns a TooManyCredits user exception to the client application. The client application provides a brief message explaining that the request was rejected. The client application then rolls back the transaction.

Figure 5-1 illustrates how the Transactions sample application works.

Figure 5-1 The Transactions Sample Application
The Development Process for the Transactions Sample Application

This section describes the steps used to add transactions to the Transactions sample application. These steps are in addition to the development process outlined in Chapter 3, “The Basic Sample Application.”

**Note:** The steps in this section have been done for you and are included in the Transactions sample application.
OMG IDL

During the development process, you would define in Object Management Group (OMG) Interface Definition Language (IDL) the register_for_courses() operation for the Registrar. The register_for_courses() operation has a parameter, NotRegisteredList, which returns to the client application the list of courses for which registration failed. If the value of NotRegisteredList is empty, the client application commits the transaction.

You also need to define the TooManyCredits user exception.

The Client Application

During the development process, you would add the following to your client application:

- The Bootstrap environmental object to obtain a reference to the TransactionCurrent environmental object in the specified WebLogic Enterprise (WLE) domain.
- The operations of the TransactionCurrent environmental object to include a CORBA object in a transaction.
- A call to the register_for_courses() operation so that students can register for courses.

For information about using Transactions in client applications, see Getting Started and Using Transactions.

The University Server Application

During the development process, you would add the following to the University server application:

- Invocations to the TP::open_xa_rm() and TP::close_xa_rm() operations in the Server::initialize() and Server::release() operations of the Server object
The Development Process for the Transactions Sample Application

- A method implementation for the `register_for_courses()` operation

For information about these tasks, see *Creating C++ Server Applications.*

The **UBBCONFIG** File

During the development process, you need the following in the **UBBCONFIG** file:

- A server group that includes both the University server application and the server application that manages the database. This server group needs to be specified as transactional.

- The `OPENINFO` parameter defined according to the `XA` parameter for the Oracle database. The `XA` parameter for the Oracle database is described in the "Developing and Installing Applications that Use the XA Libraries" section of the *Oracle7 Server Distributed Systems* manual.

  **Note:** If you use a database other than Oracle, refer to the product documentation for information about defining the XA parameter.

- The pathname to the transaction log (`TLOG`) in the `TLOGDEVICE` parameter.

For information about the transaction log and defining parameters in the **UBBCONFIG** file, see *Administration Guide.*

The **ICF** File

During the development process, change the Transaction policy of the *Registrar* object from `optional` to `always`. The `always` Transaction policy indicates that this object must be part of a transaction. For information about defining Transaction policies for CORBA objects, see *Using Transactions.*
Building the Transactions Sample Application

To build the Transactions sample application, complete the following steps:

1. Copy the files for the Transactions sample application.
2. Change the protection on the files for the Transactions sample application files.
3. Set the environment variables.
4. Initialize the University database.
5. Load the UBBCONFIG file.
6. Create a transaction log.
7. Build the client and server sample applications.

The following sections describe these steps.

**Note:** Before you can build or run the Transactions sample application, you need to complete the steps in Chapter 2, “Setting Up Your Environment.”

Copying the Files for the Transactions Sample Application into a Work Directory

The files for the Transactions sample application are located in the following directories:

**Windows NT**

*drive:* `\WLEdir\samples\corba\university\transaction`

**UNIX**

`/usr/WLEdir/samples/corba/university/transaction`
In addition, you need to copy the `utils` directory into your work directory. The `utils` directory contains files that set up logging, tracing, and access to the University database.

You will use the files listed in Table 5-1 to create the Transactions sample application.

### Table 5-1 Files Included in the Transactions Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>univt.idl</code></td>
<td>The OMG IDL that declares the <code>CourseSynopsisEnumerator</code>, <code>Registrar</code>, and <code>RegistrarFactory</code> interfaces.</td>
</tr>
<tr>
<td><code>univts.cpp</code></td>
<td>The C++ source code for the University server application in the Transactions sample application.</td>
</tr>
<tr>
<td><code>univt_i.h</code></td>
<td></td>
</tr>
<tr>
<td><code>univt_i.cpp</code></td>
<td>The C++ source code for method implementations of the <code>CourseSynopsisEnumerator</code>, <code>Registrar</code>, and <code>RegistrarFactory</code> interfaces.</td>
</tr>
<tr>
<td><code>univtc.cpp</code></td>
<td>The C++ source code for the CORBA C++ client application in the Transactions sample application.</td>
</tr>
<tr>
<td><code>frmBrowser.frm</code></td>
<td>The Visual Basic source code for the ActiveX client application in the Transactions sample application.</td>
</tr>
<tr>
<td><code>frmOpen.frm</code></td>
<td>The Visual Basic source code for the ActiveX client application in the Transactions sample application.</td>
</tr>
<tr>
<td><code>University.vbp</code></td>
<td>The Visual Basic project file for the ActiveX client application in the Transactions sample application.</td>
</tr>
<tr>
<td><code>University.vbw</code></td>
<td>The Visual Basic workspace file for the ActiveX client application in the Transactions sample application.</td>
</tr>
<tr>
<td><code>modPublicDeclarations.bas</code></td>
<td>A Visual Basic file that contains the declarations for variables used in the sample applications.</td>
</tr>
<tr>
<td><code>frmTracing.frm</code></td>
<td>The files that provide tracing capabilities to the ActiveX client application.</td>
</tr>
<tr>
<td><code>frmTracing.frx</code></td>
<td></td>
</tr>
<tr>
<td><code>frmLogon.frm</code></td>
<td>The Visual Basic file that performs the security logon for the ActiveX client application.</td>
</tr>
</tbody>
</table>
### Table 5-1  Files Included in the Transactions Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnivTApplet.java</td>
<td>The Java source code for the CORBA Java client application in the Transactions sample application.</td>
</tr>
<tr>
<td>univt_utils.h</td>
<td>The files that define database access functions for the CORBA C++ client application.</td>
</tr>
<tr>
<td>univt_utils.cpp</td>
<td></td>
</tr>
<tr>
<td>univt.icf</td>
<td>The ICF file for the Transactions sample application.</td>
</tr>
<tr>
<td>setenvt.sh</td>
<td>A UNIX script that sets the environment variables needed to build and run the Transactions sample application.</td>
</tr>
<tr>
<td>setenvt.cmd</td>
<td>An MS-DOS command procedure that sets the environment variables needed to build and run the Transactions sample application.</td>
</tr>
<tr>
<td>ubb_t.mk</td>
<td>The UBBCONFIG file for the UNIX operating system.</td>
</tr>
<tr>
<td>ubb_t.nt</td>
<td>The UBBCONFIG file for the Windows NT operating system.</td>
</tr>
<tr>
<td>makefilet.mk</td>
<td>The makefile for the Transactions sample application on the UNIX operating system.</td>
</tr>
<tr>
<td>makefilet.nt</td>
<td>The makefile for the Transactions sample application on the Windows NT operating system.</td>
</tr>
<tr>
<td>log.cpp, log.h, log_client.cpp, and log_server.cpp</td>
<td>The client and server applications that provide logging and tracing functions for the sample applications. These files are located in \utils directory.</td>
</tr>
<tr>
<td>oradbconn.cpp and orancoconn.cpp</td>
<td>The files that provide access to an Oracle SQL database instance. These files are located in \utils directory.</td>
</tr>
<tr>
<td>samplesdb.cpp and samplesdb.h</td>
<td>The files that provide print functions for the database exceptions in the sample applications. These files are located in \utils directory.</td>
</tr>
</tbody>
</table>
Changing the Protection on the Files for the Transactions Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit the files or build the files in the Transactions sample application, you need to change the protection of the files you copied into your work directory, as follows:

**Windows NT**

```
prompt>attrib -r drive:\workdirectory\*.*
```
Setting the Environment Variables

Use the following command to set the environment variables used to build the client and server applications in the Transactions sample application:

**Windows NT**

prompt>setenv

**UNIX**

prompt>/bin/ksh

prompt>./setenv.sh

Initializing the University Database

Use the following command to initialize the University database used with the Transactions sample application:

**Windows NT**

prompt>nmake -f makefile.nt initdb

**UNIX**

prompt>make -f makefile.mk initdb

Loading the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

**Windows NT**

prompt>tmloadcf -y ubb_t.nt
UNIX

prompt>tmloadcf -y ubb_t.mk

The build process for the UBBCONFIG file prompts you for an application password. This password will be used to log on to the client applications. Enter the password and press Enter. You are then prompted to verify the password by entering it again.

Creating a Transaction Log

The transaction log records the transaction activities in a WebLogic Enterprise application. During the development process, you need to define the location of the transaction log (specified by the TLOGDEVICE parameter) in the UBBCONFIG file. For the Transactions sample application, the transaction log is placed in your work directory.

To open the transaction log for the Transactions sample application, complete the following steps:

1. Enter the following command to start the Interactive Administrative Interface:
   
   tmadmin

2. Enter the following command to create a transaction log:
   
   crdl -b blocks -z directorypath
clog -m SITE1

   where

   blocks specifies the number of blocks to be allocated for the transaction log and
   directorypath indicates the location of the transaction log. The directorypath option needs to match the location specified in the TLOGDEVICE parameter in the UBBCONFIG file. The following is an example of the command on Windows NT:

   crdl -b 500 -z c:\mysamples\university\Transaction\TLOG

3. Enter q to exit the Interactive Administrative Interface.
Compiling the Transactions Sample Application

During the development process, you would use the `buildobjclient` and `buildobjserver` commands to build the client and server applications. You would also build a database-specific transaction manager to coordinate the transactional events in the client/server application. However, for the Transactions sample application, this step has been done for you. The directory for the Transactions sample application contains a `makefile` that builds the client and server sample applications and creates a transaction manager called TMS_ORA.

**Note:** In the `makefile`, the following parameter is hard coded to build a transaction manager for the Oracle database:

```
RM=Oracle_XA
```

If you use a database other than Oracle, you need to change this parameter.

Use the following commands to build the CORBA C++ client and server applications in the Transactions sample application:

**Windows NT**
```
prompt>nmake -f makefile.t.nt
```

**UNIX**
```
prompt>make -f makefile.t.mk
```

To build the CORBA Java client application:

**Windows NT**
```
prompt>nmake -f makefile.t.nt javaclient
```

**UNIX**
```
prompt>make -f makefile.t.mk javaclient
```

For information about starting the ActiveX client application, see “Starting the ActiveX Client Application.”
Running the Transactions Sample Application

For more information about the buildobjclient and buildobjserver commands, see the C++ Programming Reference.

Running the Transactions Sample Application

To run the Transactions sample application, complete the following steps:

1. Start the server application.
2. Start one or more of the client applications.

These steps are described in the following sections.

Starting the Server Application

Start the system and sample application server applications in the Transactions sample application by entering the following command:

```shell
prompt>tmboot -y
```

This command starts the following server processes:

- **TMSYSEVT**
  The BEA Tuxedo system event broker.
- **TMFFNAME**
  The transaction management services, including the NameManager and the FactoryFinder services.
- **TMIFSRVR**
  The Interface Repository server process. This server process is used only by ActiveX client applications.
- **univt_server**
  The University server process.
ISL

The IIOP Listener/Handler process.

Before using another sample application, enter the following command to stop the system and sample application server processes:

`prompt>tmshutdown`

**Starting the CORBA C++ Client Application**

Start the CORBA C++ client application in the Transactions sample application by completing the following steps:

1. At the MS-DOS prompt, enter the following command:
   
   `prompt>univt_client`

2. At the **Enter student id:** prompt, enter any number between 100001 and 100010.

3. Press Enter.

4. At the **Enter domain password:** prompt, enter the password you defined when you loaded the **UBBCONFIG** file.

5. Press Enter.

**Starting the CORBA Java Client Application**

To run the CORBA Java client application in the Transactions sample application, complete the following steps:

1. Modify the following lines in the **UnivTApplet.html** file:

   ```
   code="UnivTApplet.class"
   codebase=.
   to read as follows:
   ```

   ```
   code="UnivTApplet"
   archive="UnivTApplet.jar,m3envobj.jar"
   ```
2. Copy the modified UnivTApplet.html file to the source directory for the Web server (the directory varies by Web server product).

3. After executing the makefile to build the Transactions sample application, create a UnivTApplet.jar file, as follows:

   a. Create a tmp directory under the directory where you built the sample application and copy the UniversityT subdirectory and the class files it contains into the tmp directory.

   Copy the class files in the Transactions sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to the tmp directory, and issue one of the following commands to create a jar file that contains all the Transactions sample application classes:

   ```
   jar -cf ..\UnivTApplet.jar *.* (Microsoft Windows NT systems)
   jar -cf ../UnivTApplet.jar * (UNIX systems)
   ```

4. Copy the UnivTApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or ${TUXDIR}/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Transactions server application is running, start up your Web browser, and point it to the node where the Web server is running.

   **Note:** On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivTApplet.html file, set your browser to http://SERVER/UnivTApplet.html.

7. Enter a number between 100001 and 100010 in the student ID field.

8. In the Domain Password field, enter the password you defined when you loaded the UBBCONFIG file.

9. Double click the Logon button.
Starting the ActiveX Client Application

**Note:** For the University sample applications, the task of loading the OMG IDL for the CORBA interfaces into the Interface Repository is automated by the makefile.

Before you can start the ActiveX client application, you must use the Application Builder to create ActiveX bindings for the CORBA interfaces.

To create an ActiveX binding for a CORBA interface, complete the following steps:

1. Click the BEA Application Builder icon in the WebLogic Enterprise program group.

   The IIOP Listener window appears.

2. In the IIOP Listener window, enter the host name and port number that you specified in the **ISL** parameter in the **UBBCONFIG** file. You must match exactly the capitalization used in the **UBBCONFIG** file.

   The Logon window appears.

3. In the Logon window, enter a student ID between 100001 and 100010 for the user name and the password you defined when you loaded the **UBBCONFIG** file.

   The Application Builder window appears. All the CORBA interfaces loaded in the Interface Repository appear in the Services window of the Application Builder.

4. Highlight the UniversityT folder in the Services window and drag it to the Workstation Views window, or copy the UniversityT folder from the Services window and paste it into the Workstation Views window.

   A confirmation window appears.

5. Click Create to create the ActiveX bindings for the CORBA interfaces in the Transactions sample application.

   The Application Builder creates the following:

   - A binding for the CORBA interface. The binding is named `DImodulename_interfaceName`. For example, the binding for the Registrar interface is named `DIUniversityT_Registrar`. 
A type library. By default, the type library is placed in \\WLEdir\TypeLibraries. The type library file is named Dimodulename_interfacename.tlb.

A Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

To run the ActiveX client application, complete the following steps:

1. Open the University.vbw file in Visual Basic.
2. From the Run menu, click Start.
   
   The IIOP Listener window appears.
3. In the IIOP Listener window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must match exactly the capitalization used in the UBBCONFIG file.
   
   The Logon window appears.
4. In the Logon window, enter a student ID between 100001 and 100010 for the user name and the password you defined when you loaded the UBBCONFIG file.

Using the Client Applications in the Transactions Sample Application

The following sections briefly explain how to use the client applications in the Transactions sample application.

The CORBA C++ Client Application

The CORBA C++ client application in the Transactions sample application has the following additional option:

<R> Register for Courses
To register for a course, complete the following steps:

1. At the Options prompt, enter R.

2. At the Course Number prompt, enter a course number followed by -1. For example:
   
   100011
   100039
   -1

3. Press Enter.

4. At the Options prompt, enter L to view a list of courses for which the student ID is registered.

To exit the C++ CORBA client application, enter E at the Options prompt.

The CORBA Java Client Application

When you log on to the CORBA Java client application, a Student Account Summary window appears. Use the Student Account Summary window to register for courses.

To register for a class, complete the following steps:

1. Obtain a list of available courses from the Student Account Summary window by entering a text string in the Search String text box. For example, computer.

2. Click the Search Catalog button.

   A list of courses matching the search string appears in the window.

3. Select a course by clicking on its name in the lower portion of the Student Account Summary window.

   If you are already registered for a course, Yes appears in the Registered field on the Student Account Summary window.

4. To register for the course, click the Register button on the Student Account Summary window.

   To view a list of courses for which the student ID is registered, double click the Show Registration button.
To exit the CORBA Java client application, click the Logoff button in the Student Account Summary window, or choose Quit from the Applet menu.

The ActiveX Client Application

When you log on to the ActiveX client application, the Course Browser window appears. Use the Course Browser window to register for courses.

To register for a class, complete the following steps:

1. In the text box next to the Find Courses button, enter a text string or use the pulldown menu to choose a curriculum subject. For example, computer.

2. Click the Find Courses button.
   A list of all the courses that match that search string appears.

3. Select a course from the list that appears in the window next to the Get Details button, or double-click the course name.
   The details for the selected course appear.

4. Click the Register for Course button or double click the course to enter the course into the schedule.
   The course appears in the student’s schedule at the bottom of the window. If the student is already registered for the course, it appears in the color green. If the course conflicts with a previously registered course, it appears in the color red.

To remove a course from the schedule, double click the course in the schedule.

To view a list of courses for which the student ID is registered, click the Get Registered Courses button.

To complete information about a course, click the Get Details button.

To exit the ActiveX client application, choose Exit from the File menu.
6 The Wrapper Sample Application

The topic includes the following sections:

- How the Wrapper Sample Application Works
- The Development Process for the Wrapper Sample Application
- Building the Wrapper Sample Application
- Compiling the Wrapper Sample Application
- Running the Wrapper Sample Application
- Using the Client Applications in the Wrapper Sample Application

Refer to Readme.txt in the \wrapper directory for troubleshooting information and the latest information about using the Wrapper sample application.

How the Wrapper Sample Application Works

In the Wrapper sample application, when a student registers for classes, the student’s account is charged for the classes and the balance of the student’s account is updated. In addition, students can get information about their account balances.
The Wrapper sample application incorporates an additional server application and a BEA Tuxedo application. The Billing server application contains a Teller object, which calls the BEA Tuxedo Teller application. The BEA Tuxedo Teller application performs the following billing operations:

- Debiting a student account
- Crediting a student account
- Obtaining the current balance of a student account

The University database is modified to include account information.

Figure 6-1 illustrates how the Wrapper sample application works.

**Figure 6-1  The Wrapper Sample Application**
The Development Process for the Wrapper Sample Application

This section describes the development process required when wrapping a BEA Tuxedo application in a WebLogic Enterprise (WLE) application. These steps are in addition to the development process outlined in Chapter 3, “The Basic Sample Application.”

Note: The steps in this section have been done for you and are included in the Wrapper sample application.

OMG IDL

When wrapping a BEA Tuxedo application, you need to define an object that interoperates with the BEA Tuxedo application and a factory that creates that object. In the Wrapper sample application, the Teller and TellerFactory objects interact with the BEA Tuxedo Teller application. During the development process, you would define the interfaces of the Teller and TellerFactory objects in Object Management Group (OMG) Interface Definition Language (IDL), as follows:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>TellerFactory</td>
<td>Returns an object reference to the Teller object</td>
<td>find_teller()</td>
</tr>
<tr>
<td>Teller</td>
<td>Interoperates with the BEA Tuxedo Teller application to perform billing and accounting operations</td>
<td>get_balance()</td>
</tr>
</tbody>
</table>

You need to add a Balance field to the StudentDetails structure. Client applications use the Balance field to show the student’s account balance. A user exception DelinquentAccount is also added.
The Client Application

During the development process, you would add code to the client application to handle the user exception \texttt{Delinquent Account} that the \texttt{register_for_courses()} operation can raise.

The Server Application

During the development process, you would write the following for the Billing server application:

- Method implementations for the \texttt{get_balance()}, \texttt{credit()}, and \texttt{debit()} operations for the \texttt{Teller} object. The method implementations need to include the code that does the following:
  - Allocates an FML message buffer
  - Fills the FML message buffer with the data you want to send to the BEA Tuxedo Teller application
  - Calls the BEA Tuxedo Teller application
  - Extracts information from the FML message buffer returned from the BEA Tuxedo Teller application
  - Returns the information from the FML message buffer to the University server application

- A method implementation for the \texttt{find_teller()} operation of the \texttt{TellerFactory} object.

- A Billing server object that creates and registers the \texttt{TellerFactory} object and calls the \texttt{open_XA_Rm} and \texttt{close_XA_Rm} functions.

During the development process, you would add the following to the University server application:

- In the server initialization portion of the code for the University server application, include the Bootstrap object to get a FactoryFinder object for the \texttt{TellerFactory} object. The University server application is using the Bootstrap and FactoryFinder objects like a client application would.
In the code for the University server application, include a reference to the TellerFactory object in the constructor of the servant for the Registrar object. Use the TellerFactory object to create a Teller object.

In the method implementations for the get_student_details() and register_for_courses() operations for the Registrar object, invoke the get_balance() and debit() operations on the Teller object.

For information about writing server applications that wrap BEA Tuxedo applications, see Creating C++ Server Applications.

The UBBCONFIG File

During the development process, you need to make the following changes to the UBBCONFIG file:

- Define the following server groups in the GROUPS section of the UBBCONFIG file:
  - ORA_GRP, which contains the University server application, the BEA Tuxedo Teller application, and the server application for the University database. This server group allows both the University server application and the BEA Tuxedo Teller application to access the University database.
  - APP_GRP, which contains the Billing server application.

- Specify the server applications in the Wrapper sample application in the order in which they should be booted in the SERVERS section of the UBBCONFIG file. Start the server applications in the following order:
  a. BEA Tuxedo Teller application
  b. Billing server application
  c. University server application

The ICF File

During the development process, you need to define activation and transaction policies for the Teller and TellerFactory objects. The Teller and TellerFactory objects have the following policies:
The Wrapper Sample Application

- The Teller object has an activation policy of process and a transaction policy of optional.

- The TellerFactory object has an activation policy of process and a transaction policy of ignore.

For information about defining activation and transaction policies for CORBA objects, see Creating C++ Server Applications.

Building the Wrapper Sample Application

To build the Wrapper sample application, complete the following steps:

1. Copy the files for the Wrapper sample application.
2. Change the protection on the files for the Wrapper sample application.
3. Set the environment variables.
4. Initialize the University database.
5. Load the UBBCONFIG file.
6. Create a transaction log.
7. Build the client and server sample applications.

The following sections describe these steps.

Note: Before you can build or run the Wrapper sample application, you need to complete the steps in Chapter 2, “Setting Up Your Environment.”

Copying the Files for the Wrapper Sample Application into a Work Directory

The files for the Wrapper sample application are located in the following directories:

Windows NT
In addition, you need to copy the `utils` directory into your work directory. The `utils` directory contains files that set up logging, tracing, and access to the University database.

You will use the files listed in Table 6-1 to create the Wrapper sample application.

**Table 6-1  Files Included in the Wrapper Sample Application**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billw.idl</td>
<td>The OMG IDL that declares the Teller and TellerFactory interfaces.</td>
</tr>
<tr>
<td>univw.idl</td>
<td>The OMG IDL that declares the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>billws.cpp</td>
<td>The C++ source code for the Billing server application in the Wrapper sample application.</td>
</tr>
<tr>
<td>univws.cpp</td>
<td>The C++ source code for the University server application in the Wrapper sample application.</td>
</tr>
<tr>
<td>billw_i.h</td>
<td>The C++ source code for the method implementations of the Teller and TellerFactory interfaces.</td>
</tr>
<tr>
<td>billw_i.cpp</td>
<td>The C++ source code for the method implementations of the Teller and TellerFactory interfaces.</td>
</tr>
<tr>
<td>univw_i.h</td>
<td>The C++ source code for the method implementations of the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univw_i.cpp</td>
<td>The C++ source code for the method implementations of the CourseSynopsisEnumerator, Registrar, and RegistrarFactory interfaces.</td>
</tr>
<tr>
<td>univwc.cpp</td>
<td>The C++ source code for the CORBA C++ client application in the Wrapper sample application.</td>
</tr>
<tr>
<td>frmBrowser.frm</td>
<td>The Visual Basic source code for the ActiveX client application in the Wrapper sample application.</td>
</tr>
<tr>
<td>university.vbp</td>
<td>The Visual Basic project file for the ActiveX client application in the Wrapper sample application.</td>
</tr>
</tbody>
</table>
### Table 6-1 Files Included in the Wrapper Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>University.vbw</td>
<td>The Visual Basic workspace file for the ActiveX client application in the Wrapper sample application.</td>
</tr>
<tr>
<td>modPublicDeclarations.bas</td>
<td>A Visual Basic file that contains the declarations for variables used in the sample applications.</td>
</tr>
<tr>
<td>frmTracing.frm</td>
<td>The files that provide tracing capabilities to the ActiveX client application.</td>
</tr>
<tr>
<td>frmTracing.frx</td>
<td></td>
</tr>
<tr>
<td>frmLogon.frm</td>
<td>The Visual Basic file that performs the security logon for the ActiveX client application.</td>
</tr>
<tr>
<td>univWApplet.java</td>
<td>The Java source code for the CORBA Java client application in the Wrapper sample application.</td>
</tr>
<tr>
<td>univw_utils.h</td>
<td>The files that define database access functions for the CORBA C++ client application.</td>
</tr>
<tr>
<td>univw_utils.cpp</td>
<td></td>
</tr>
<tr>
<td>univw.icf</td>
<td>The ICF file for the University server application in the Wrapper sample application.</td>
</tr>
<tr>
<td>billw.icf</td>
<td>The ICF file for the Billing server application in the Wrapper sample application.</td>
</tr>
<tr>
<td>setenvw.sh</td>
<td>A UNIX script that sets the environment variables needed to build and run the Wrapper sample application.</td>
</tr>
<tr>
<td>tellw_flds, tellw_u.c,</td>
<td>The files for the BEA Tuxedo Teller application.</td>
</tr>
<tr>
<td>tellw_c.h, tellws.ec</td>
<td></td>
</tr>
<tr>
<td>setenvw.cmd</td>
<td>An MS-DOS command procedure that sets the environment variables needed to build and run the Wrapper sample application.</td>
</tr>
<tr>
<td>ubb_w.mk</td>
<td>The UBBCONFIG file for the UNIX operating system.</td>
</tr>
<tr>
<td>ubb_w.nt</td>
<td>The UBBCONFIG file for the Windows NT operating system.</td>
</tr>
<tr>
<td>makefilew.mk</td>
<td>The makefile for the Wrapper sample application on the UNIX operating system.</td>
</tr>
</tbody>
</table>
Changing the Protection on the Files for the Wrapper Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit the files or build the files in the Wrapper sample application, you need to change the protection of the files you copied into your work directory, as follows:

**Windows NT**

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

prompt>chmod u+rw /workdirectory/*.*

Setting the Environment Variables

Use the following command to set the environment variables used to build the client and server applications in the Wrapper sample application:

Windows NT
prompt>setenvw

UNIX
prompt>/bin/ksh
prompt>./setenvw.sh

Initializing the University Database

Use the following command to initialize the University database used with the Wrapper sample application:

Windows NT
prompt>nmake -f makefilew.nt initdb

UNIX
prompt>make -f makefilew.mk initdb

Loading the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

Windows NT
prompt>tmloadcf -y ubb_w.nt
UNIX

prompt>tmloadcf -y ubb_w.mk

The build process for the UBBCONFIG file prompts you for an application password. This password will be used to log on to the client applications. Enter the password and press Enter. You are then prompted to verify the password by entering it again.

Creating a Transaction Log

The transaction log records the transaction activities in a WebLogic Enterprise application. During the development process, you need to define the location of the transaction log (specified by the TLOGDEVICE parameter) in the UBBCONFIG file. For the Wrapper sample application, the transaction log is placed in your work directory.

To open the transaction log for the Wrapper sample application, complete the following steps:

1. Enter the following command to start the Interactive Administrative Interface:
   tmadmin

2. Enter the following command to create a transaction log:
   
   crdl -b blocks -z directorypath
crlog -m SITE1

   where

   blocks specifies the number of blocks to be allocated for the transaction log, and directorypath indicates the location of the transaction log. The directorypath option needs to match the location specified in the TLOGDEVICE parameter in the UBBCONFIG file. The following is an example of the command on Windows NT:

   crdl -b 500 -z c:\mysamples\university\wrapper\TLOG

3. Enter q to quit the Interactive Administrative Interface.
Compiling the Wrapper Sample Application

During the development process, you would use the `buildobjclient` and `buildobjserver` commands to build the client and server applications. However, for the Wrapper sample application, this step has been done for you. The directory for the Wrapper sample application contains a `makefile` that builds the client and server sample applications.

Use the following commands to build the CORBA C++ client and server application in the Wrapper sample application:

**Windows NT**

```
prompt>nmake -f makefilew.nt
```

**UNIX**

```
prompt>make -f makefilew.mk
```

To build the CORBA Java client application:

**Windows NT**

```
prompt>nmake -f makefilew.nt javaclient
```

**UNIX**

```
prompt>make -f makefilew.mk javaclient
```

For information about starting the ActiveX client application, see “Starting the ActiveX Client Application.”

For more information about the `buildobjclient` and `buildobjserver` commands, see the *Commands, System Processes, and MIB Reference*.

Running the Wrapper Sample Application

To run the Wrapper sample application, complete the following steps:
Running the Wrapper Sample Application

1. Start the server application.
2. Start one or more of the client applications.
These steps are described in the following sections.

Starting the Server Application

Start the system and sample application server processes in the Wrapper sample application by entering the following command:

```
prompt>tmboot -y
```

This command starts the following server processes:

- **TMSYSEVT**
  The BEA Tuxedo system event broker.

- **TMFFNAME**
  The transaction management services, including the NameManager and the FactoryFinder services.

- **TMIFSRVR**
  The Interface Repository server process. This server process is used only by ActiveX client applications.

- **univw_server**
  The University server process.

- **tellw_server**
  The BEA Tuxedo Teller application process.

- **billw_server**
  The Billing server application process.

- **ISL**
  The IIOP Listener/Handler process.

Before using another sample application, enter the following command to stop the system and sample application server processes:
Starting the CORBA C++ Client Application

Start the CORBA C++ client application in the Wrapper sample application by completing the following steps:

1. At the MS-DOS prompt, enter the following command:
   
   `prompt>univw_client`

2. At the `Enter student id:` prompt, enter any number between 100001 and 100010.

3. Press Enter.

4. At the `Enter domain password:` prompt, enter the password you defined when you loaded the `UBBCONFIG` file.

5. Press Enter.

Starting the CORBA Java Client Application

To run the CORBA Java client application in the Wrapper sample application, complete the following steps:

1. Modify the following lines in the `UnivWApplet.html` file:

   ```html
   code="UnivWApplet.class"
   codebase=.
   ```

   to read as follows:

   ```html
   code="UnivWApplet"
   archive="UnivWApplet.jar,m3envobj.jar"
   ```

2. Copy the modified `UnivWApplet.html` file to the source directory for the Web server (the directory varies by Web server product).

3. After executing the `makefile` to build the Wrapper sample application, create a `UnivWApplet.jar` file, as follows:
Running the Wrapper Sample Application

a. Create a tmp directory under the directory where you built the sample application and copy the UniversityW subdirectory and the class files it contains into the tmp directory.

Copy the class files in the Wrapper sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to the tmp directory, and issue one of the following commands to create a JAR file that contains all the Wrapper sample application classes:

```
jar -cf ..\UnivWApplet.jar *.* (Microsoft Windows NT systems)
jar -cf ../UnivWpplet.jar * (UNIX systems)
```

4. Copy the UnivWApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or ${TUXDIR}/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Wrapper server application is running, start up your Web browser, and point it to the node where the Web server is running.

Note: On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivWApplet.html file, set your browser to http://SERVER/UnivWApplet.html.

7. Enter a number between 100001 and 100010 in the student ID field.

8. In the Domain Password field, enter the password you defined when you loaded the UBBCONFIG file.

9. Double click the Logon button.

Exceptions appear in the status bar of the Appletviewer window or in the MS-DOS window in which the CORBA Java client application was started.
Starting the ActiveX Client Application

**Note:** For the University sample applications, the task of loading the OMG IDL for the CORBA interfaces into the Interface Repository is automated by the makefile.

Before you can start the ActiveX client application, you must use the Application Builder to create ActiveX bindings for the CORBA interfaces.

To create an ActiveX binding for a CORBA interface, complete the following steps:

1. Click the BEA Application Builder icon in the WebLogic Enterprise program group.

   The IIOP Listener window appears.

2. In the IIOP Listener window, enter the host name and port number that you specified in the `ISL` parameter in the `UBBCONFIG` file. You must exactly match the capitalization used in the `UBBCONFIG` file.

   The Logon window appears.

3. In the Logon window, enter a student ID between 100001 and 100010 for the user name and the password you defined when you loaded the `UBBCONFIG` file.

   The Application Builder window appears. All the CORBA interfaces loaded in the Interface Repository appear in the Services window of the Application Builder.

4. Highlight the UniversityW folder in the Services window and drag it to the Workstation Views window, or copy the UniversityW folder from the Services window and paste it into the Workstation Views window.

   A confirmation window appears.

5. Click Create to create the ActiveX bindings for the CORBA interfaces in the Wrapper sample application.

   The Application Builder creates the following:

   - A binding for the CORBA interface. The binding is named `Dimodulename_interfaceName`. For example, the binding for the `Registrar` interface is named `DIUniversityW_Registrar`. 

6-16  Guide to the University Sample Applications
A type library. By default, the type library is placed in \WLEdir\TypeLibraries.

The type library file is named DImodulename_interfacename.tlb.

A Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

To run the ActiveX client application, complete the following steps:

1. Open the University.vbw file in Visual Basic.
2. From the Run menu, click Start.
   The IIOP Listener window appears.
3. In the IIOP Listener window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must match exactly the capitalization used in the UBBCONFIG file.
   The Logon window appears.
4. In the Logon window, enter a student ID between 100001 and 100010 for the user name and the password you defined when you loaded the UBBCONFIG file.

The following sections explain how to use the client applications in the Wrapper sample application.

The CORBA C++ Client Application

The CORBA C++ client application in the Wrapper sample application has the following additional option:

<B> Display Your Balance
The Display Your Balance option displays the account balance associated with the student ID used to log on to the CORBA C++ client application.

To exit the C++ CORBA client application, enter E at the Options prompt.

The CORBA Java Client Application

The CORBA Java client application in the Wrapper sample application allows you to display an account balance.

When you log on to the CORBA Java client application, a Student Account Summary window appears. The Student Account Summary window displays the account balance associated with the student ID used to log on to the CORBA Java client application. When you register for additional courses, the amount in the Balance text box in the Student Account Summary window increases.

To exit the CORBA Java client application, click the Logoff button in the Student Account Summary window, or choose Quit from the File menu.

The ActiveX Client Application

The ActiveX client application in the Wrapper sample application allows you to display an account balance.

When you log on to the ActiveX client application, a Course Browser window appears. The Course Browser window displays the registered courses and account balance for the student ID used to log on to the ActiveX client application. When you register for additional courses, the amount in the Balance text box in the Course Browser window increases.

To exit the ActiveX client application, choose Exit from the File menu.
CHAPTER

7  The Production Sample Application

This topic includes the following sections:

- How the Production Sample Application Works
- The Development Process for the Production Sample Application
- Building the Production Sample Application
- Compiling the Production Sample Application
- Running the Production Sample Application
- How the Production Sample Application Can Be Scaled Further

**Note:** The client applications in the Production sample application work in the same manner as the client applications in the Wrapper sample application.

Refer to `Readme.txt` in the `\production` directory for troubleshooting information and the latest information about using the Production sample application.
How the Production Sample Application Works

The Production sample application provides the same end-user functionality as the Wrapper sample application. The Production sample application demonstrates how to use features of the WebLogic Enterprise (WLE) software to scale an existing WebLogic Enterprise application. The Production sample application does the following:

- Replicates the University server application, the Billing server application, and the BEA Tuxedo Teller application within the ORA_GRP and APP_GRP server groups defined in the UBCONFIG file.
- Replicates the ORA_GRP1 and APP_GRP1 server groups on an additional server machine, Production Machine 2, as ORA_GRP2 and APP_GRP2 and partitions the database.
- Implements a stateless object model to scale up the number of requests from client applications the server application can manage simultaneously.
- Assigns unique object IDs (OIDs) to the following objects so that they can be instantiated multiple times simultaneously in their respective server groups, thereby making them available on a per-client-application (and not per-process) basis:
  - Registrar
  - RegistrarFactory
  - Teller
  - TellerFactory
- Implements factory-based routing to direct requests from client applications on behalf of some students to one server machine, and other students to another server machine.

Note: To make the Production sample application easy for you to use, the sample application is configured on the WebLogic Enterprise software kit to run on one machine using one database. However, the Production sample application is set up so that it can be configured to run on several machines and to use
multiple databases. Changing the configuration to multiple machines and
databases involves simply modifying the `UBBCONFIG` file and partitioning the
database.

The following sections describe how the Production sample application uses replicated
server applications, replicated server groups, object state management, and
factory-based routing to scale the Production sample application.

**Replicating Server Applications**

When you replicate server applications:

- You obtain a means to balance the load of incoming requests from client
  applications on that server application. As requests arrive in the WebLogic
  Enterprise domain for the server group, the WebLogic Enterprise system routes
  the request to the least busy server application within that group.

- You can specify how many copies of a given server application process are
  running on a server machine. The number of copies determines the extent to
  which the WebLogic Enterprise domain can process requests in-parallel from
  client applications.

- You obtain a useful failover protection in the event that one of the server
  application processes stops.

In the Production sample application, the server applications are replicated in the
following manner:

- The University server application, the BEA Tuxedo Teller application, and the
  server application for the University database are replicated within the `ORA_GRP`
group.

- The Billing server application is replicated within the `APP_GRP` group.

Figure 7-1 shows the replicated `ORA_GRP` and `APP_GRP` server groups.
In Figure 7-1, note the following:

- There can be no more than one instance of the **RegistrarFactory**, **Registrar**, **TellerFactory**, or **Teller** objects within a single server application process.

- There can be any number of **CourseSynopsisEnumerator** objects within a server application process.
Replicating Server Groups

Server groups are a feature of the WebLogic Enterprise software that allow you to add server machines to an existing WebLogic Enterprise application. When you replicate a server group, you can do the following:

- Spread the processing load for a WebLogic Enterprise application across multiple server machines.
- Use factory-based routing to send requests from client applications to a particular server machine.

The way in which server groups are configured and replicated is specified in the UBBCONFIG file.

Figure 7-2 shows the server groups in the Production sample application replicated on a second server machine. The replicated server groups are defined as ORA_GRP2 and APP_GRP2 in the UBBCONFIG file for the Production sample application.
In Figure 7-2, the only difference between the content of the server groups on Production Machine 1 and Production Machine 2 is the database. The University database is partitioned into two databases. The database on Production Machine 1 contains student and account information for students with IDs between 100001 and 100005. The database on Production Machine 2 contains student and account information for students with IDs between 100006 and 100010.
Using a Stateless Object Model

To achieve scalability gains, the Registrar and Teller objects are configured in the Production sample application to have the method activation policy. The method activation policy results in the following behavior changes:

- Whenever the objects are invoked, they are instantiated by the WebLogic Enterprise domain in the appropriate server group.
- After the invocation is complete, the WebLogic Enterprise domain deactivates the objects.

In the Basic through the Production sample applications, the Registrar object had an activation policy of process. All requests from client applications on the Registrar object went to the same object instance in the memory of the server machine. This design is adequate for a small-scale deployment. However, as client application demands increase, requests from client applications on the Registrar object eventually become queued, and response time drops.

However, when the Registrar and Teller objects have an activation policy of method and the server applications that manage these objects are replicated, the Registrar and Teller objects can process multiple requests from client applications in parallel. The only constraint is the number of server application processes that are available to instantiate the Registrar and Teller objects.

For the WebLogic Enterprise application to instantiate copies of the Registrar and Teller objects in each of the replicated server application processes, each copy of the Registrar and Teller objects have an unique object ID (OID). The factories that create these objects are responsible for assigning them unique OIDs. For information about generating unique object IDs, see Creating C++ Server Applications.

Using Factory-based Routing

Factory-based routing is a WebLogic Enterprise feature that allows you to send a request from a client application to a specific server group. Using factory-based routing, you can spread the processing load for a WebLogic Enterprise application across multiple server machines. The Production sample application uses factory-based routing in the following way:
Requests from client applications to the Registrar object are routed based on the student ID. Requests from student ID 100001 to 100005 go to Production Machine 1. Requests from student ID 100006 to 100010 go to Production Machine 2.

Requests from the Registrar object to the Teller object are routed based on account number. Billing requests for account 200010 to 200014 go to Production Machine 1. Billing requests for account 200015 to 200019 go to Production Machine 2.

For information about setting up factory-based routing, see *Creating C++ Server Applications*.

**The Development Process for the Production Sample Application**

This section describes the development process required when scaling a WebLogic Enterprise application. These steps are in addition to the development process outlined in Chapter 3, “The Basic Sample Application.”

**Note:** The steps in this section have been done for you and are included in the Production sample application.

**OMG IDL**

During the development process, to support factory-based routing, you would make modifications to the Object Management Group (OMG) Interface Definition Language (IDL) definitions for the following operations:

- The `find_registrar()` operation of the `RegistrarFactory` object to require a student ID
- The `find_teller()` operation of the `TellerFactory` object to require an account number
The Development Process for the Production Sample Application

For information about implementing factory-based routing, see *Creating C++ Server Applications.*

The Client Application

During the development process, you would specify a `STU_ID` value when creating a `Registrar` object. The `STU_ID` value defines to which server group the request from the client application is routed.

In the Production sample application, the University server application creates the `Teller` object in the same way a client application would. Therefore, an `ACT_NUM` value needs to be specified when creating a `Teller` object.

The Server Application

During the development process, you need to modify the invocation to the `TP::create_object_reference()` operation for the `RegistrarFactory` and `TellerFactory` objects to include an NVlist that specifies routing criteria. The `criteria` parameter of the `TP::create_object_reference()` operation specifies a list of named values to be used for factory-based routing, as follows:

- The `RegistrarFactory` object in the Production sample application specifies the value for `criteria` to be `STU_ID`.
- The `TellerFactory` object in the Production sample application specifies the value for `criteria` to be `ACT_NUM`.

The value of the `criteria` parameter must match exactly the routing criteria name, field, and field type specified in the `ROUTING` section of the `UBBCONFIG` file.

For information about implementing factory-based routing in a factory, see *Creating C++ Server Applications.*
The UBBCONFIG File

The UBBCONFIG file is the key to achieving scalability in a WebLogic Enterprise application. This section describes how the UBBCONFIG file for the Production sample application is modified to:

- Replicate server application processes and server groups
- Implement factory-based routing

Replicating Server Application Processes and Server Groups

During the development process, modify the UBBCONFIG file in the following way to configure replicated server application processes and server groups:

1. In the GROUPS section of the UBBCONFIG file, specify the names of the groups you want to configure. In the Production sample application, there are four server groups: APP_GRP1, APP_GRP2, ORA_GRP1, and ORA_GRP2.

2. In the SERVERS section of the UBBCONFIG file, enter the following information for the server application process you want to replicate:
   - A server application name.
   - The GROUP parameter, which specifies the name of the server group to which the server application process belongs. If you are replicating a server process across multiple groups, specify the server process once for each group.
   - The SRVID parameter, which specifies a unique administrative ID for the server machine.
   - The MIN parameter, which specifies the number of instances of the server application process to start when the WebLogic Enterprise application is started. You need to start at least two server application processes.
   - The MAX parameter, which specifies the maximum number of server application processes that can be running at any one time. You can specify no more than five server application processes.

The MIN and MAX parameters determine the degree to which a given server application can process requests in parallel on a given object. During run time, the system administrator can examine resource bottlenecks and start additional server processes, if necessary. In this sense, the application is scaled by the system administrator.
The Development Process for the Production Sample Application

The following example shows lines from the GROUPS and SERVERS sections of the UBBCONFIG file for the Production sample application.

*GROUPS

**GROUP**
LMID = SITE1
GRPNO = 2
TMSNAME = TMS

**GROUP**
LMID = SITE1
GRPNO = 3
TMSNAME = TMS

**GROUP**
LMID = SITE1
GRPNO = 4
OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir
            =.+MaxCur=5"
CLOSEINFO = ""
TMSNAME = "TMS_ORA"

**GROUP**
LMID = SITE1
GRPNO = 5
OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir
            =.+MaxCur=5"
CLOSEINFO = ""
TMSNAME = "TMS_ORA"

*SERVERS

# By default, activate 2 instances of each server
# and allow the administrator to activate up to 5
# instances of each server
DEFAULT:
MIN = 2
MAX = 5
tellp_server
SRVGRP = ORA_GRP1
SRVID = 10
RESTART = N
tellp_server
SRVGRP = ORA_GRP2
SRVID = 10
RESTART = N
billp_server
SRVGRP = APP_GRP1
SRVID = 10
RESTART = N
billp_server
SRVGRP = APP_GRP2
Implementing Factory-based Routing

For each interface for which you want to enable factory-based routing, you need to define the following information in the `UBBCONFIG` file:

- Details about the data in the routing criteria
- For each kind of criteria, the values that route to specific server groups

During the development process, make the following changes to the `UBBCONFIG` file:

1. The `INTERFACES` section lists the names of the interfaces for which you want to enable factory-based routing. For each interface, this section specifies the value on which the interface routes. The routing value is specified in the `FACTORYROUTING` identifier.

   The following example shows the `FACTORYROUTING` identifier for the Registrar and Teller objects in the Production sample application:

   ```
   INTERFACES
   "IDL:beasys.com/UniversityP/Registrar:1.0"
   FACTORYROUTING = STU_ID
   "IDL:beasys.com/BillingP/Teller:1.0"
   FACTORYROUTING = ACT_NUM
   ```

2. The `ROUTING` section specifies the following data for each routing value:

   - The `TYPE` parameter, which specifies the type of routing. In the Production sample application, the type of routing is factory-based routing. Therefore, this parameter is defined to `FACTORY`.
   - The `FIELD` parameter, which specifies the name that the factory inserts in the routing value. In the Production sample application, the field parameters are `student_id` and `account_number`. 

```
The Development Process for the Production Sample Application

- The `FIELDTYPE` parameter, which specifies the data type of the routing value. In the Production sample application, the field types for `STU_ID` and `ACT_NUM` are long.

- The `RANGES` parameter, which specifies the values that are routed to each group.

The following example shows the `ROUTING` section of the `UBBCONFIG` file used in the Production sample application:

```
*ROUTING
  STU_ID
  FIELD     = "student_id"
  TYPE      = FACTORY
  FIELDTYPE = LONG
  RANGES    = "100001-100005:ORA_GRP1,100006-100010:ORA_GRP2"

  ACT_NUM
  FIELD     = "account_number"
  TYPE      = FACTORY
  FIELDTYPE = LONG
  RANGES    = "200010-200014:APP_GRP1,200015-200019:APP_GRP2"
```

The example shows that Registrar objects for students with IDs 100001 through 100005 are instantiated in `ORA_GRP1`, and students with IDs 100006 through 100010 are instantiated in `ORA_GRP2`. Likewise, Teller objects for accounts 200010 through 200014 are instantiated in `APP_GRP1`, and accounts 200015 through 200019 are instantiated in `APP_GRP2`.

3. The groups specified by the `RANGES` identifier in the `ROUTING` section of the `UBBCONFIG` file need to be identified and configured. For example, the Production sample application specifies four groups: `ORA_GRP1`, `ORA_GRP2`, `APP_GRP1`, and `APP_GRP2`. These groups need to be configured, and the machines on which they run need to be identified.

Note: The names of the server groups in the `GROUPS` section must exactly match the group names specified in the `ROUTING` section.

The ICF File

During the development process, you need to change the activation policy of the Registrar, RegistrarFactory, Teller, and TellerFactory objects from process to method. For information about defining activation and transaction policies for CORBA objects, see Creating C++ Server Applications.
To build the Production sample application, complete the following steps:

1. Copy the files for the Production sample application into a work directory.
2. Change the protection on the files for the Production sample application files.
3. Set the environment variables.
4. Initialize the University database.
5. Load the UBCONFIG file.
6. Create a transaction log.
7. Build the client and server sample applications.

The following sections describe these steps.

Note: Before you can build or run the Production sample application, you need to complete the steps in Chapter 2, “Setting Up Your Environment.”

Copying the Files for the Production Sample Application
into a Work Directory

The files for the Production sample application are located in the following directories:

Windows NT

\drive:\WLEdir\samples\corba\university\production

UNIX

/usr/WLEdir/samples/corba/university/production

In addition, you need to copy the utils directory into your work directory. The utils directory contains files that set up logging, tracing, and access to the University database.
You will use the files in Table 7-1 to create the Production sample application.

### Table 7-1 Files Included in the Production Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>billp.idl</td>
<td>The OMG IDL that declares the <strong>Teller</strong> and <strong>TellerFactory</strong> interfaces.</td>
</tr>
<tr>
<td>univp.idl</td>
<td>The OMG IDL that declares the <strong>CourseSynopsisEnumerator</strong>, <strong>Registrar</strong>, and <strong>RegistrarFactory</strong> interfaces.</td>
</tr>
<tr>
<td>billps.cpp</td>
<td>The C++ source code for the Billing server application in the Production sample application.</td>
</tr>
<tr>
<td>univps.cpp</td>
<td>The C++ source code for the University server application in the Production sample application.</td>
</tr>
<tr>
<td>billp__i.h</td>
<td>The C++ source code for the method implementations of the <strong>Teller</strong> and <strong>TellerFactory</strong> interfaces.</td>
</tr>
<tr>
<td>billp_i.cpp</td>
<td>The C++ source code for the method implementations of the <strong>Teller</strong> and <strong>TellerFactory</strong> interfaces.</td>
</tr>
<tr>
<td>univp_i.h</td>
<td>The C++ source code for the method implementations of the <strong>CourseSynopsisEnumerator</strong>, <strong>Registrar</strong>, and <strong>RegistrarFactory</strong> interfaces.</td>
</tr>
<tr>
<td>univp_i.cpp</td>
<td>The C++ source code for the method implementations of the <strong>CourseSynopsisEnumerator</strong>, <strong>Registrar</strong>, and <strong>RegistrarFactory</strong> interfaces.</td>
</tr>
<tr>
<td>univpc.cpp</td>
<td>The C++ source code for the CORBA C++ client application in the Production sample application.</td>
</tr>
<tr>
<td>frmBrowser.frm</td>
<td>The Visual Basic source code for the ActiveX client application in the Production sample application.</td>
</tr>
<tr>
<td>university.vbp</td>
<td>The Visual Basic project file for the ActiveX client application in the Production sample application.</td>
</tr>
<tr>
<td>University.vbw</td>
<td>The Visual Basic workspace file for the ActiveX client application in the Production sample application.</td>
</tr>
<tr>
<td>modPublicDeclarations.bas</td>
<td>A Visual Basic file that contains the declarations for variables used in the sample applications.</td>
</tr>
<tr>
<td>frmTracing.frm</td>
<td>The files that provide tracing capabilities to the ActiveX client application.</td>
</tr>
<tr>
<td>frmTracing.frx</td>
<td>The files that provide tracing capabilities to the ActiveX client application.</td>
</tr>
</tbody>
</table>
### Table 7-1  Files Included in the Production Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frmLogon.frm</td>
<td>The Visual Basic file that performs the security logon for the ActiveX client application.</td>
</tr>
<tr>
<td>univPApplet.java</td>
<td>The Java source code for the CORBA Java client application in the Production sample application.</td>
</tr>
<tr>
<td>univp_utils.h</td>
<td>The files that define database access functions for the CORBA C++ client application.</td>
</tr>
<tr>
<td>univp_utils.cpp</td>
<td></td>
</tr>
<tr>
<td>univp.icf</td>
<td>The Implementation Configuration File (ICF) for the University server application in the Production sample application.</td>
</tr>
<tr>
<td>billp.icf</td>
<td>The ICF file for the Billing server application in the Production sample application.</td>
</tr>
<tr>
<td>tellw_flds, tellw_u.c, tellw_c.h, tellws.ec</td>
<td>The files for the BEA Tuxedo Teller application.</td>
</tr>
<tr>
<td>setenvp.sh</td>
<td>A UNIX script that sets the environment variables needed to build and run the Production sample application.</td>
</tr>
<tr>
<td>setenvp.cmd</td>
<td>An MS-DOS command procedure that sets the environment variables needed to build and run the Production sample application.</td>
</tr>
<tr>
<td>ubb_p.mk</td>
<td>The UBBCONFIG file for the UNIX operating system.</td>
</tr>
<tr>
<td>ubb_p.nt</td>
<td>The UBBCONFIG file for the Windows NT operating system.</td>
</tr>
<tr>
<td>makefilep.mk</td>
<td>The makefile for the Production sample application on the UNIX operating system.</td>
</tr>
<tr>
<td>makefilep.nt</td>
<td>The makefile for the Production sample application on the Windows NT operating system.</td>
</tr>
<tr>
<td>log.cpp, log.h, log_client.cpp, and log_server.cpp</td>
<td>The files for the client and server applications that provide logging and tracing functions for the sample applications. These files are located in the \utils directory.</td>
</tr>
</tbody>
</table>
Changing the Protection on the Files for the Production Sample Application

During the installation of the WebLogic Enterprise software, the sample application files are marked read-only. Before you can edit the files or build the files in the Production sample application, you need to change the protection of the files you copied into your work directory, as follows:

**Windows NT**

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

**UNIX**

```
prompt> chmod u+rw /workdirectory/*.*
```
Setting the Environment Variables

Use the following command to set the environment variables used to build the client and server applications in the Production sample application:

**Windows NT**

```bash
prompt>setenvp
```

**UNIX**

```bash
prompt>/bin/ksh
prompt>./setenvp.sh
```

Initializing the University Database

Use the following command to initialize the University database used with the Production sample application:

**Windows NT**

```bash
prompt>nmake -f makefilep.nt initdb
```

**UNIX**

```bash
prompt>make -f makefilep.mk initdb
```

Loading the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

**Windows NT**

```bash
prompt>tmloadcf -y ubb_p.nt
```

**UNIX**

```bash
prompt>tmloadcf -y ubb_p.mk
```
The build process for the `UBBCONFIG` file prompts you for an application password. This password will be used to log on to the client applications. Enter the password and press Enter. You are then prompted to verify the password by entering it again.

## Creating a Transaction Log

The transaction log records the transaction activities in a WebLogic Enterprise application. During the development process you need to define the location of the transaction log (specified by the `TLOGDEVICE` parameter) in the `UBBCONFIG` file. For the Production sample application, the transaction log is placed in your work directory.

You need to complete the following steps to open the transaction log for the Production sample application:

1. Enter the following command to start the Interactive Administrative Interface:
   ```
tadmin
   ```

2. Enter the following command to create a transaction log:
   ```
crdl -b blocks -z directorypath
crlog -m SITE1
   ```
   where
   - `blocks` specifies the number of blocks to be allocated for the transaction log,
   - `directorypath` indicates the location of the transaction log. The `directorypath` option needs to match the location specified in the `TLOGDEVICE` parameter in the `UBBCONFIG` file. The following is an example of the command on Windows NT:
   ```
crdl -b 500 -z c:\mysamples\university\production\TLOG
   ```

3. Enter `q` to quit the Interactive Administrative Interface.
Compiling the Production Sample Application

During the development process, you would use the `buildobjclient` and `buildobjserver` commands to build the client and server applications. However, for the Production sample application, this step has been done for you. The directory for the Production sample application contains a `makefile` that builds the client and server sample applications.

Use the following commands to build the CORBA C++ client and server application in the Production sample application:

**Windows NT**

prompt>`nmake -f makefilep.nt`

**UNIX**

prompt>`make -f makefilep.mk`

To build the CORBA Java client application:

**Windows NT**

prompt>`nmake -f makefilep.nt javaclient`

**UNIX**

prompt>`make -f makefilep.mk javaclient`

For information about starting the ActiveX client application, see “Starting the ActiveX Client Application.”

For more information about the `buildobjclient` and `buildobjserver` commands, see the Commands, System Processes, and MIB Reference.
Running the Production Sample Application

To run the Production sample application, complete the following steps:

1. Start the server application.
2. Start one or more of the client applications.

The following sections describe these steps in detail.

Starting the Server Application

Start the system and sample application server applications in the Production sample application by entering the following command:

```
prompt>tmboot -y
```

This command starts the following server processes:

- **TMSYSEVT**
  - The BEA Tuxedo system event broker.

- **TMFFNAME**
  - The transaction management services, including the NameManager and the FactoryFinder services.

- **TMIFSRVR**
  - The Interface Repository server process. This server process is used only by ActiveX client applications.

- **univp_server**
  - Four processes of the University server application.

- **tellp_server**
  - Four processes of the BEA Tuxedo Teller application.

- **billp_server**
  - Four processes of the Billing server application.
ISL

The IIOP Listener/Handler process.

Before using another sample application, enter the following command to stop the system and sample application server processes:

```
prompt>tmshutdown
```

Starting the CORBA C++ Client Application

Start the CORBA C++ client application in the Production sample application by completing the following steps:

1. At the MS-DOS prompt, enter the following command:
   ```
prompt>univp_client
   ```
2. At the Enter student id: prompt, enter any number between 100001 and 100010.
3. Press Enter.
4. At the Enter domain password: prompt, enter the password you defined when you loaded the UBBCONFIG file.
5. Press Enter.

**Note:** The CORBA C++ client application in the Production sample application works in the same manner as the CORBA C++ client application in the Wrapper sample application.

Starting the CORBA Java Client Application

To run the CORBA Java client application in the Production sample application, complete the following steps:

1. Modify the following lines in the UnivPApplet.html file:
   ```html
   code="UnivPApplet.class"
   codebase="",
   ```
   to read as follows:
code="UnivPApplet"
archive="UnivPApplet.jar,m3envobj.jar"

2. Copy the modified UnivPApplet.html file to the source directory for the Web server (the directory varies by Web server product).

3. After executing the makefile to build the Production sample application, create a UnivPApplet.jar file, as follows:

   a. Create a tmp directory under the directory where you built the sample application and copy the UniversityP subdirectory and the class files it contains into the tmp directory.

   Copy the class files in the Production sample application directory that were generated by the makefile into the tmp directory, set the directory (cd) to the tmp directory, and issue one of the following commands to create a jar file that contains all the Production sample application classes:

   jar -cf ..\UnivPApplet.jar *.* (Microsoft Windows NT systems)
jar -cf ../UnivPApplet.jar * (UNIX systems)

4. Copy the UnivPApplet.jar file you just created to the source directory for the Web server (the directory name varies by Web server product).

5. Copy the m3envobj.jar file from the appropriate subdirectory (%TUXDIR%\udataobj\java Microsoft Windows NT systems or ${TUXDIR}/udataobj/java UNIX systems) to the Web server source directory.

6. Make sure the Security server application is running, start up your Web browser, and point it to the node where the Web server is running.

   Note: On Microsoft Windows NT systems, the node name needs to be in all uppercase characters. For example, if the node is specified as SERVER in the UBBCONFIG file and in the UnivPApplet.html file, set your browser to http://SERVER/UnivPApplet.html.

7. Enter a number between 100001 and 100010 in the student ID field.

8. In the Domain Password field, enter the password you defined when you loaded the UBBCONFIG file.

9. Double click the Logon button.
Note: The CORBA Java client application in the Production sample application works in the same manner as the CORBA Java client application in the Wrapper sample application.

Starting the ActiveX Client Application

Note: For the University sample applications, the task of loading the OMG IDL for the CORBA interfaces into the Interface Repository is automated by the makefile.

Before you can start the ActiveX client application, you must use the Application Builder to create ActiveX bindings for the CORBA interfaces.

To create an ActiveX binding for a CORBA interface, complete the following steps:

1. Click the BEA Application Builder icon in the WebLogic Enterprise program group.
   The IIOP Listener window appears.

2. In the IIOP Listener window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must match exactly the capitalization used in the UBBCONFIG file.
   The Logon window appears.

3. In the Logon window, enter a student ID between 100001 and 100010 for the user name and the password you defined when you loaded the UBBCONFIG file.
   The Application Builder window appears. All the CORBA interfaces loaded in the Interface Repository appear in the Services window of the Application Builder.

4. Highlight the UniversityP folder in the Services window and drag it to the Workstation Views window, or copy the UniversityP folder from the Services window and paste it into the Workstation Views window.
   A confirmation window appears.

5. Click Create to create the ActiveX bindings for the CORBA interfaces in the Production sample application.
   The Application Builder creates the following:
How the Production Sample Application Can Be Scaled Further

- A binding for the CORBA interface. The binding is named DImodulename_interfacename. For example, the binding for the Registrar interface is named DIUniversityP_Registrar.

- A type library. By default, the type library is placed in \WLEdir\TypeLibraries.
  
  The type library file is named DImodulename_interfacename.tlb.

- A Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

To run the ActiveX client application, complete the following steps:

1. Open the University.vbw file in Visual Basic.
2. From the Run menu, click Start.
   
   The IIOP Listener window appears.
3. In the IIOP Listener window, enter the host name and port number that you specified in the ISL parameter in the UBBCONFIG file. You must exactly match the capitalization used in the UBBCONFIG file.
   
   The Logon window appears.
4. In the Logon window, enter a student ID between 100001 and 100010 for the username and the password you defined when you loaded the UBBCONFIG file.

Note: The ActiveX client application in the Production sample application works in the same manner as the ActiveX client application in the Wrapper sample application.

How the Production Sample Application Can Be Scaled Further

The Production sample application can be scaled even more by:

- Replicating the server groups in the Production sample application across additional machines
You need to modify the UBBCONFIG file to specify the additional server groups, the server application processes that run in the new server groups, and the server machines on which the server groups run.

- Changing the factory-based routing tables

For example, instead of routing to the two existing server groups in the Production sample application, you can modify the routing rules in the UBBCONFIG file to partition the application further among additional server groups. Any modification to the routing tables must match the information in the UBBCONFIG file.

Note: If you add capacity to an existing WebLogic Enterprise application that uses a database, you must consider how the database is set up, particularly when you are using factory-based routing. For example, if the Production sample application is spread across six machines, the database on each machine must be set up appropriately and in accordance with the routing tables in the UBBCONFIG file.
APPENDIX

A Setting Up the Database

The University sample applications use a database (the University database) to store all the data (for example, course names and course summaries) used in the sample applications. Before you can build and run the University sample applications, you need to install and set up the database.

This appendix describes the setup process for the database. For details about the steps, see the product documentation for the database you are using.

Database Support

The University sample applications shipped with version 4.2 of the WebLogic Enterprise (WLE) software can be used with Oracle version 7.3.3 or greater. Table A-1 lists what version of Oracle is supported on each operating system.

Table A-1  Database Support

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Version of Oracle Database Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris SPARC version 2.6</td>
<td>Oracle version 7.3.4 Enterprise Edition for Sun SPARC</td>
</tr>
<tr>
<td>Solaris SPARC version 7.0</td>
<td>Oracle version 7.3.4 for Solaris version 2.7</td>
</tr>
<tr>
<td>HP-UX version 10.20</td>
<td>Oracle version 7.3.3 for HP-UX version 10.20</td>
</tr>
<tr>
<td>HP-UX version 11.0</td>
<td>Oracle version 8.0 for HP-UX version 11.0</td>
</tr>
</tbody>
</table>
Setup Instructions for the Oracle Database

You can use the Oracle database in the following manner:

- As a local instance
- As a remote instance

Setup for Local Database Instance

If you are using a local instance of the Oracle database, you need to install the following Oracle components:

- Programmer/2000 Pro*C/C++
- TCP/IP Adapter
- SQL*Net Client
- SQL*Plus
Setup Instructions for the Oracle Database

- Oracle 7 Server
- Oracle 7 Utilities
- SQL*Net Server

Use the default database created by the Oracle installation program. You need the connection string for the Oracle database and the default user id and password. Refer to the Oracle product documentation for information about obtaining this information.

Once the Oracle software is installed, you need to start the daemon for the Oracle database. The daemon is generally started as part of the start-up process for your machine. You also need to enable an XA resource manager and set privileges so that the XA resource manager can manage the interaction between the Oracle database and the sample applications ($v\text{xatrans}$).

Enter the following command to enable an XA resource manager that works with the University sample applications:

SQL> grant select on $v\text{xatrans}$ to public;

**Note:** Enter the following additional command only if you are using version 8.0 of the Oracle database.

SQL> grant select on dba_pending_transactions to user;
SQL> commit;

where user is the default user of the Oracle database.

For more information about performing these steps, see the Oracle product documentation.

Instructions for initializing the Oracle database are included in the descriptions of building the individual sample applications.

Setup for Remote Database Instance

If you are using a remote instance of the Oracle database (for example, a database running on another machine), you need to install the following Oracle components:

- Programmer/2000 Pro*C/C++, Version 2.2.3.0.0
A Setting Up the Database

- TCP/IP Adapter, version 2.3.3.0.0
- SQL*Net Client, version 2.3.3.0.0
- SQL*Plus, version 3.3.3.0.0

In order to use a remote instance of the Oracle database, you need to define an alias for the database. You need the following information:

- A character string that identifies the remote instance of the Oracle database.
- The name of the machine on which the remote instance of the Oracle database is located. This name must match host name specified in the machine’s UBBCONFIG file.
- The SID of the remote instance of the Oracle database.

Use the SQL *Net Easy Configuration Utility to define an alias for the remote instance of the Oracle database. For more information about performing these steps, see the Oracle product documentation.

The database you are accessing needs an XA resource manager enabled with privileges that allow interaction between the database and the sample applications (v$xatrans$).

Enter the following command to enable an XA resource manager that works with the University sample applications:

```sql
SQL>grant select on v$xatrans$ to public;
```

**Note:** Enter the following additional command only if you are using version 8.0 of the Oracle database.

```sql
SQL>grant select on dba_pending_transactions to user;
SQL>commit;
```

where `user` is the default user of the Oracle database.

Instructions for initializing the Oracle database are included in the descriptions of building the individual sample applications.
Index

A
activation policies
  Basic sample application 3-7
  process 7-7
  Production sample application 7-7
  Wrapper sample application 6-6
ActiveX client applications
  starting
    Basic sample application 3-14
    Production sample application 7-23
    Security sample application 4-12
    Transactions sample
      application 5-14
    Wrapper sample application 6-15
  using
    Basic sample application 3-18
    Production sample application 7-1
    Security sample application 4-14
    Transactions sample
      application 5-17
    Wrapper sample application 6-17
  writing
    Basic sample application 3-6
    Production sample application 7-9
    Security sample application 4-3
    Transactions sample application 5-4
    Wrapper sample application 6-4
APPDIR parameter
  setenv file 2-4
  UBBCONFIG file 2-7

Application Builder
  using
    Basic sample application 3-15
    Production sample application 7-23
    Security sample application 4-12
    Transactions sample
      application 5-14
    Wrapper sample application 6-15

B
Basic sample application 3-12
  activation policies 3-7
  buildobjclient command 3-12
  buildobjserver command 3-12
  changing protection on files 3-11
  compiling client applications 3-12
  compiling the server application 3-12
  description 3-2
  ICF file 3-7
  illustrated 3-2
  loading the UBBCONFIG file 3-12
  makefile 3-12
  setenv file 3-11
  setting up work directory 3-8
  source files 3-8
  starting the ActiveX client
    application 3-14
  starting the CORBA C++ client
    application 3-14
starting the CORBA Java client application 3-14
starting the Oracle database 3-11
starting the server application 3-13
tmloadcf command 3-12
transaction policies 3-7
UBBCONFIG file 3-7
writing server applications 3-6

BEA TUXEDO applications
replicating 7-2, 7-3
wrapping 6-2

Bootstrap object
Basic sample application 3-6
in client applications 3-6
Security sample application 4-3
Transactions sample application 5-4
Wrapper sample application 6-4

building
Basic sample application 3-8
Production sample application 7-13
Security sample application 4-4
transaction manager 5-11
Transactions sample application 5-6
Wrapper sample application 6-6

buildobjclient command
Basic sample application 3-12
Production sample application 7-20
Security sample application 4-9
Transactions sample application 5-11
Wrapper sample application 6-12

buildobjserver command
Basic sample application 3-12
Production sample application 7-20
Security sample application 4-9
Transactions sample application 5-11
Wrapper sample application 6-12

client applications
Bootstrap object 3-6
FactoryFinder object 3-6
initializing the ORB 3-6
PrincipalAuthenticator operation 4-3
SecurityCurrent object 4-3
TransactionCurrent object 5-4
types 3-6
writing
Basic sample application 3-6
Production sample application 7-9
Security sample application 4-3, 5-4
Transactions sample application 5-4
Wrapper sample application 6-4

client stubs
generating 3-5
in sample applications 3-5

compiling
client applications
Basic sample application 3-12
Production sample application 7-20
Security sample application 4-9
Transactions sample application 5-11
Wrapper sample application 6-12

server applications
Basic sample application 3-12
Production sample application 7-20
Security sample application 4-9
Transactions sample application 5-11
Wrapper sample application 6-12

configuring
Basic sample application 3-7
factory-based routing 7-9
Production sample application 7-9
replicated server applications 7-9
replicated server groups 7-9
security 4-4
Security sample application 4-4
Transactions sample application 5-5
Guide to the University Sample Applications

Wrapper sample application 6-5
CORBA C++ client applications
  client stubs 3-5
  starting
    Basic sample application 3-14
    Production sample application 7-22
    Security sample application 4-11
    Transactions sample
      application 5-13
    Wrapper sample application 6-14
  using
    Basic sample application 3-16
    Production sample application 7-1
    Security sample application 4-14
    Transactions sample
      application 5-16
    Wrapper sample application 6-17
  writing
    Basic sample application 3-6
    Production sample application 7-9
    Security sample application 4-3
    Transactions sample application 5-3
    Wrapper sample application 6-4

CORBA Java client applications
  client stubs 3-5
  starting
    Basic sample application 3-14
    Production sample application 7-22
    Security sample application 4-11
    Transactions sample
      application 5-14
    Wrapper sample application 6-14
  using
    Basic sample application 3-17
    Production sample application 7-1
    Security sample application 4-14
    Transactions sample
      application 5-17
    Wrapper sample application 6-17
  writing
    Basic sample application 3-6

Production sample application 7-9
Security sample application 4-3
Transactions sample application 5-3
Wrapper sample application 6-4

CourseSynopsisEnumerator interface
OMG IDL 3-3
CPCMPL parameter
UBBCONFIG file 2-6
CPPINC parameter
UBBCONFIG file 2-6

D

development process
  client applications
    Production sample application 7-9
    Security sample application 4-3
    Transactions sample application 5-4
    Wrapper sample application 6-4
  factory-based routing 7-12
  ICF file
    Basic sample application 3-7
    Production sample application 7-13
    Security sample application 4-4
    Transactions sample application 5-5
    Wrapper sample application 6-5
  OMG IDL
    Basic sample application 3-3
    Production sample application 7-8
    Security sample application 4-3
    Transactions sample application 5-4
    Wrapper sample application 6-3
  replicated server applications 7-10
  replicated server groups 7-10
  server applications
    Security sample application 4-4
    Transactions sample application 5-4
    Wrapper sample application 6-4
  UBBCONFIG file 4-4, 7-9
    Basic sample application 3-7
    factory-based routing 7-12
replicated server applications 7-10
replicated server groups 7-10
Transactions sample application 5-5
Wrapper sample application 6-5

DII
in sample applications 3-5

F
factory-based routing
description 7-7
Production sample application 7-7
routing criteria 7-12
UBBCONFIG file 7-12
INTERFACES section 7-12
ROUTING section 7-12
FactoryFinder object
in client applications 3-6
FIELD parameter 7-12
FIELDTYPE parameter 7-12
file protections
Basic sample application 3-11
Production sample application 7-17
Security sample application 4-7
Transactions sample application 5-9
Wrapper sample application 6-9
FML message buffer
in server applications 6-4

G
genicf command 3-7
GROUP parameter 7-10
GROUPS section
replicating server applications 7-10
replicating server groups 7-10

I
ICF file
Basic sample application 3-7

Production sample application 7-13
Security sample application 4-4
Transactions sample application 5-5
Wrapper sample application 6-5
idl command 3-5
Implementation Configuration File
see ICF file
initialize the ORB 3-6
INTERFACES section
implementing factory-based routing 7-12
ISL parameter
Basic sample application 3-14, 3-15
Production sample application 7-22
Security sample application 4-11
Transactions sample application 4-12, 5-14, 5-15, 6-15, 7-23
UBBCONFIG file 2-8

J
JDKDIR parameter
setenv file 2-5

L
LD_LIBRARY_PATH parameter
UBBCONFIG file 2-6
LIBPATH parameter
UBBCONFIG file 2-6
loading the UBBCONFIG file
Basic sample application 3-12
Production sample application 7-18
Security sample application 4-8
Transactions sample application 5-10
Wrapper sample application 6-10
M
makefile
   Basic sample application 3-12
   client stubs 3-5
   Production sample application 7-20
   Security sample application 4-9
   skeletons 3-5
   Transactions sample application 5-11
   Wrapper sample application 6-10
MAX parameter 7-10
method implementations
   Basic sample application 3-6
   Security sample application 4-4
   Transactions sample application 5-4
   Wrapper sample application 6-4
MIN parameter 7-10
MY_SERVER_MACHINE parameter
   UBBCONFIG file 2-7

N
naming conventions
   sample application code 1-3
   setenv file 2-3
   UBBCONFIG file 2-3
NETSCAPE parameter
   setenv file 2-5

O
object IDs  7-2
OIDs  7-2
OMG IDL
   Basic sample application 3-3
   compiling 3-5
   CourseSynopsisEnumerator
      interface 3-3
   generating client stubs 3-5
   generating skeletons 3-5
   Production sample application 7-8
   Registrar interface 3-3
   RegistrarFactory interface 3-3
   Security sample application 4-3
   Teller interface 6-3
   TellerFactory interface 6-3
   Transactions sample application 5-4
   user exception 5-4
   Wrapper sample application 6-3
OPENINFO parameter
   Transactions sample application 5-5
   UBBCONFIG file 2-8
Oracle database
   setting the XA parameter 5-5
   starting
      Basic sample application 3-11
      Production sample application 7-18
      Security sample application 4-8
      Transactions sample
         application 5-10
      Wrapper sample applications 6-10
ORACLE_SID parameter
   setenv file 2-6
ORADIR parameter
   setenv file 2-5

P
PrincipalAuthenticator object
   using in sample applications 4-2
Production sample application
   activation policies 7-7
   buildobjclient command 7-20
   buildobjserver command 7-20
   changing protection on files 7-17
   compiling client applications 7-20
   compiling server applications 7-20
   description 7-2
   development process 7-8
   factory-based routing 7-2, 7-7
   ICF file 7-13
   illustrated 7-2
loading the UBBCONFIG file 7-18
makefile 7-20
replicating server applications 7-2
replicating server groups 7-2
server groups 7-5
setenv file 7-18
setting up a work directory 7-14
source files 7-14
starting the ActiveX client
application 7-23
starting the CORBA C++ client
application 7-22
starting the CORBA Java client
application 7-22
starting the Oracle database 7-18
starting the server application 7-21
stateless objects 7-7
tmloadcf command 7-18
UBBCONFIG file 7-9
writing client applications 7-9
writing server applications 7-9

R
RANGES parameter 7-12
Registrar interface
OMG IDL 3-3
RegistrarFactory interface
OMG IDL 3-3
replicating
server applications 7-3
description 7-2
server groups 7-5
ROUTING section
FIELD parameter 7-12
FIELDTYPE parameter 7-12
implementing factory-based
routing 7-12
RANGES parameter 7-12
TYPE parameter 7-12

S
sample applications
naming conventions 1-3
overview
scaling
Production sample application 7-2, 7-22
stateless objects 7-7
security
adding to sample applications 4-1
application-level 4-1
SECURITY parameter 4-4
Security sample application
buildobjclient command 4-9
buildobjserver command 4-9
changing protection on files 4-7
client applications 4-3
compiling
client applications 4-9
compiling the server application 4-9
description 4-1
development process 4-3
ICF file 4-4
illustrated 4-2
initializing the database 4-8
ISL parameter 4-11
loading the UBBCONFIG file 4-8
makefile 4-9
PrincipalAuthenticator object 4-2
SecurityCurrent object 4-2
server application 4-4
setenv file 4-8
setting up the work directory 4-5
source files 4-5
tmloadcf command 4-8
UBBCONFIG file 4-4
writing client applications 4-3
writing server applications 4-4
SecurityCurrent object
using in client applications 4-2
using in Security sample application 4-3
server applications
  configuring in groups 7-5
  ICF file 3-7
  ISL parameter 3-13
  method implementations 3-6
  replicating 7-3
    description 7-2
    UBBCONFIG file 7-10
server object 3-6
starting
  Basic sample application 3-13
  Production sample application 7-21
  Security sample application 4-10
  Transactions sample
    application 5-12
  Wrapper sample application 6-13
TMFFNAME 3-13
TMIFSRVR 3-13
TMSYSEVT 3-13
UBBCONFIG file
  GROUPS section 7-10
  SERVERS section 7-10
using FML message buffers 6-4
writing
  Basic sample application 3-6
  Production sample application 7-9
  Security sample application 4-4
  Transactions sample
    application 5-4
  Wrapper sample application 6-4
server groups
  creating 7-5
  Production sample application 7-5
  replicating 7-5, 7-10
  Transactions sample application 5-5
UBBCONFIG file
  GROUPS section 7-10
  SERVERS section 7-10
  Wrapper sample application 6-5
server object
  Basic sample application 3-6
  Transactions sample application 5-4
SERVERS section
  GROUP parameter 7-10
  MAX parameter 7-10
  MIN parameter 7-10
  replicating server applications 7-10
  replicating server groups 7-10
  SRVID parameter 7-10
setenv file
  Basic sample application 3-11
  description 2-2
  parameters 2-4
  Transactions sample application 5-9
SHLIB_PATH parameter
  UBBCONFIG file 2-6
skeletons
  generating 3-5
  in sample applications 3-5
software requirements
  C++ 2-1
  Java 2-1
  Visual Basic 2-1
source files
  Basic sample application 3-8
  Production sample application 7-14
  Security sample application 4-5
  Transactions sample application 5-6
  Wrapper sample application 6-6
SRVID parameter 7-10
stateless objects 7-7
support
  documentation xvi
  technical xvi
system environment variables
  Basic sample application 3-11
  Production sample application 7-18
  Security sample application 4-8
  setting 2-9
  Transactions sample application 5-9
T

Teller interface
  OMG IDL 6-3
TellerFactory interface
  OMG IDL 6-3
TLOGDEVICE parameter 5-5, 7-19
tmloadcf command
  Basic sample application 3-12
  Production sample application 7-18
  Security sample application 4-9
  Transactions sample application 5-10
  Wrapper sample application 6-10
TMS_ORA 5-11
TOBJADDR parameter
  setenv file 2-5
transaction manager
  building 5-11
  TMS_ORA 5-11
transaction policies
  Basic sample application 3-7
  Transactions sample application 5-5
  Wrapper sample application 6-6
TransactionCurrent object
  using in client applications 5-4
transactions
  description 5-1
  ICF file 5-5
  in client applications 5-4
  OMG IDL 5-1
  TransactionCurrent object 5-4
  UBBCONFIG file 5-5
transactions log
  creating
    Production sample application 7-19
    Transactions sample application 5-10
    Wrapper sample application 6-11
Transactions sample application
  buildobjclient command 5-11
  buildobjserver command 5-11

changing protection on files 5-9
compiling client applications 5-11
compiling server application 5-11
description 5-1
development process 5-3
illustrated 5-2
initializing the database 5-10
setenv file 5-9
setting up work directory 5-6
source files 5-7
starting server application 5-4
tmloadcf command 5-10
transaction policies 5-5
UBBCONFIG file 5-5
writing client applications 5-4
writing server applications 5-4
TUXCONFIG file
  description 3-7
TUXCONFIG parameter
  setenv file 2-4
  UBBCONFIG file 2-7
TUXDIR parameter
  setenv file 2-4
  UBBCONFIG file 2-8
TYPE parameter 7-12

U

UBBCONFIG file
  Basic sample application 3-7, 3-12
  description 2-2
  parameters 2-6
  replicating server groups 7-5
  security 4-4
  SECURITY parameter 4-4
  Security sample application 4-4
  Transactions sample application 5-5
  Wrapper sample application 6-5
UNIX
  APPDIR parameter 2-4
  naming conventions for setenv file 2-3
naming conventions for UBBCONFIG
file 2-3
setenv parameters 2-4
UBBCONFIG parameters 2-6
user exceptions
Transactions sample application 5-2
Wrapper sample application 6-3
USERID parameter
setenv file 2-6

W
Windows NT
APPDIR parameter 2-4
naming conventions for setenv file 2-3
naming conventions for UBBCONFIG
file 2-3
setenv parameters 2-4
UBBCONFIG parameters 2-6
WLE domain
adding security to 4-2
Wrapper sample application
activation policies 6-6
buildobjclient command 6-12
buildobjserver command 6-12
changing protection on files 6-9
compiling client applications 6-12
compining the server application 6-12
description 6-1
development process 6-3
ICF file 6-5
illustrated 6-2
loading the UBBCONFIG file 6-10
makefile 6-10
setting up work directory 6-6
source files 6-6
starting CORBA C++ client application
6-14
starting the ActiveX client application
6-15
starting the CORBA Java client

application 6-14
starting the Oracle database 6-10
starting the server application 6-13
tmloadcf command 6-10
transactions log 6-11
UBBCONFIG file 6-5
writing client applications 6-4
writing server applications 6-4
wrapping
BEA TUXEDO applications 6-2

X
XA parameter 5-5