BEA Tuxedo®

Guide to CORBA University Sample Applications

Version 9.1
Document Released: May 16, 2006

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

About This Document

Introduction

Setting Up Your Environment

The Basic Sample Application

The Security Sample Application

The Transactions Sample Application

The Wrapper Sample Application

The Production Sample Application

Index

About This Document

This document describes the University sample applications that are provided with the BEA Tuxedo® 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 4, "The Security Sample Application," describes the Security 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.

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 CORBA environment in the BEA Tuxedo product.

e-docs Web Site

The BEA Tuxedo 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 BEA Tuxedo 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 BEA Tuxedo 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, distributed object computing, transaction processing, C++ client programming, and C++ server programming, see the *CORBA Bibliography* in the BEA Tuxedo online documentation.

Contact Us!

Your feedback on the BEA Tuxedo 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 BEA Tuxedo documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 9.1 release.

If you have any questions about this version of BEA Tuxedo, or if you have problems installing and running BEA Tuxedo, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	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`** `( )`

Convention	Item
monospace italic text	Identifies variables in code. *Example*: `String expr`
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	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 for the CORBA environment in the BEA Tuxedo product. The sample applications provide client and server programmers with the basic concepts of developing distributed client/server applications using the CORBA environment and introduces many of the more advanced CORBA features of the BEA Tuxedo product.

This topic includes the following sections:

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

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

An Overview of the University Sample Applications

The BEA Tuxedo software kit includes a CORBA sample application suite based on client and server applications implemented at a university. Each University sample application demonstrates a new set of CORBA 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 CORBA feature of the BEA Tuxedo product.

Table 1-1 describes the University sample applications.

Table 1-1 The University Sample Applications

University Sample Application	Description
Basic	Describes how to create CORBA client and server applications, configure a CORBA application, and build and run the client and server applications included in the Basic sample application. CORBA C++ client applications are provided as well as a CORBA C++ server application.
Security	Adds application-level security to the CORBA client applications in the Basic sample application and to the configuration of the CORBA application.
Transactions	Adds transactional objects to the CORBA 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.
Wrapper	Demonstrates how to wrap an ATMI application as a CORBA object.
Production	Demonstrates replicating CORBA server applications, creating stateless CORBA objects, and implementing factory-based routing in CORBA server applications.

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

- *Getting Started with BEA Tuxedo CORBA Applications*
- *Creating CORBA Client Applications*
- *Creating CORBA 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

Convention	Description
`crs`	The abbreviation for course.
`syn`	The abbreviation for synopsis.
`det`	The abbreviation for details.
`lst`	The abbreviation for list.
`enum`	The abbreviation for enumerator.
`stu`	The abbreviation for student.
`num`	The abbreviation for number.
`cur`	The abbreviation for current.
`_oref`	A `CORBA::Object` reference.
`_ref`	A typed object reference.
`p_`	The abbreviation for ptr.
`v_`	The abbreviation for var.
`s_`	The abbreviation for file static data.
`m_`	The abbreviation for class member data.
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).
Type names	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`.

CHAPTER 2

Setting Up Your Environment

This topic describes how to configure your CORBA application so that you can run the University sample applications.

This topic includes the following sections:

- System Prerequisites
- Editing the setenv and UBBCONFIG Files

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

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.

Note: The user is assumed to know how to setup RDBMS (e.g., Oracle) before running the sample

For details about the setting up a database, see the product documentation for the database you are using.

System Prerequisites

For information about the operating system platforms supported by the product, see *Installing the BEA Tuxedo System*.

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

- Visual C++ .NET 2003

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 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 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. The bold letter is the identifying letter for the sample application.

Table 2-1 Naming Conventions for setenv and UBBCONFIG Files

University Sample Application	Naming Convention
Basic	• `setenv`**`b`**`.cmd`—the `setenv` file for Windows • `setenv`**`b`**`.sh`—the `setenv` file for UNIX • `ubb_`**`b`**`.nt`—the `UBBCONFIG` file for Windows • `ubb_`**`b`**`.mk`—the `UBBCONFIG` file for UNIX
Security	• `setenv`**`s`**`.cmd`—the `setenv` file for Windows • `setenv`**`s`**`.sh`—the `setenv` file for UNIX • `ubb_`**`s`**`.nt`—the `UBBCONFIG` file for Windows • `ubb_`**`s`**`.mk`—the `UBBCONFIG` file for UNIX
Transactions	• `setenv`**`t`**`.cmd`—the `setenv` file for Windows • `setenv`**`t`**`.sh`—the `setenv` file for UNIX • `ubb_`**`t`**`.nt`—the `UBBCONFIG` file for Windows • `ubb_`**`t`**`.mk`—the `UBBCONFIG` file for UNIX
Wrapper	• `setenv`**`w`**`.cmd`—the `setenv` file for Windows • `setenv`**`w`**`.sh`—the `setenv` file for UNIX • `ubb_`**`w`**`.nt`—the `UBBCONFIG` file for Windows • `ubb_`**`w`**`.mk`—the `UBBCONFIG` file for UNIX
Production	• `setenv`**`p`**`.cmd`—the `setenv` file for Windows • `setenv`**`p`**`.sh`—the `setenv` file for UNIX • `ubb_`**`p`**`.nt`—the `UBBCONFIG` file for Windows • `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

Parameter	Description
APPDIR	The directory path where you copied the sample application files. For example: **Windows** APPDIR=c:*work*\university\basic **UNIX** APPDIR=/usr/*work*/university/basic
TUXCONFIG	The directory path and name of the configuration file. For example: **Windows** TUXCONFIG=c:\work\university\basic\tuxconfig **UNIX** TUXCONFIG=/usr/*work*/university/basic/tuxconfig
TUXDIR	The directory path where you installed the BEA Tuxedo software. For example: **Windows** TUXDIR=c:\Tux8 **UNIX** TUXDIR=/usr/local/Tux8
ORACLE_HOME	The directory path where you installed the Oracle software. For example: **Windows** ORADIR=c:\Orant **UNIX** ORACLE_HOME=/usr/local/oracle
TOBJADDR	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

Table 2-2 Parameters in the setenv File (Continued)

Parameter	Description
USERID	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
ORACLE_SID	The instance ID of the Oracle database. On Windows, you do not need to specify the ORACLE_SID, the parameter automatically defaults to ORCL.
CCMPL	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.
CPPCMPL	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.
CPPINC	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.
SHLIB_PATH, LD_LIBRARY_PATH, or LIBPATH	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.
PROC	The directory location of the Oracle Programmer C/C++ SQL Precompiler. You only need to specify this parameter if you are using the Windows operating system.
PRODIR	The directory location of the Oracle Programmer C/C++ SQL Precompiler. You only need to specify this parameter if you are using the Windows operating system.

Setting the UBBCONFIG Parameters

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

Table 2-3 Parameters in the UBBCONFIG File

Parameter	Description

MY_SERVER_MACHINE	Delete this parameter and replace it with the name of the server machine.
	On Windows, 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
	`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
	`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.

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

Parameter	Description

TUXDIR	The full directory path where you installed the BEA Tuxedo software. The directory path needs to be included in quotation marks. For example:
	Windows
	`TUXDIR="c:\Tux8"`
	UNIX
	`TUXDIR="/usr/local/Tux8"`
	This parameter needs to match the `TUXDIR` parameter in the `setenv` file.

`CLOPT` for the `ISL` process	Enter the host name and port number of the machine on which the server application is installed. For example:
	`ISL`
	`SRVGRP = SYS_GRP`
	`SRVID =`
	`CLOPT = "-A --n //BEANIE:2500"`

OPENINFO	If you are using the Transactions, Wrapper, or Production sample applications, you need to specify this parameter for the Oracle database.
	If you are using a remote instance of the Oracle database, the `OPENINFO` parameter is specified as follows:
	`OPENINFO = "Oracle_XA:Oracle_XA+SqlNet=`*`aliasname`*`+Acc=P/`*`account`*`/`*`password`*`+SesTM=100+LogDir=.+MaxCur=5"`
	For example, on Windows:
	`OPENINFO = "Oracle_XA:Oracle_XA+SqlNet=ORCL+Acc=P/scott/tiger+SesTM=100+LogDir=.+MaxCur=5"`
	If you are using a local instance of the Oracle database, the `OPENINFO` parameter is specified as follows:
	`OPENINFO = "Oracle_XA:Oracle_XA+Acc=P/`*`account`*`/`*`password`*`+SesTM=100+LogDir=.+MaxCur=5"`
	For example, on Windows:
	`OPENINFO = "Oracle_XA:Oracle_XA+Acc=P/scott/tiger+SesTM=100+LogDir=.+MaxCur=5"`

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 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++.NET 2003 is installed in the following location on Windows:

```
c:\Program Files\Microsoft Visual Studio.NET 2003\vc7
```

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:\Program Files\Microsoft Visual Studio.NET 2003\Common7\Tools\
vsvars32.bat
```

CHAPTER 3

The Basic Sample Application

This topic includes the following sections:

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

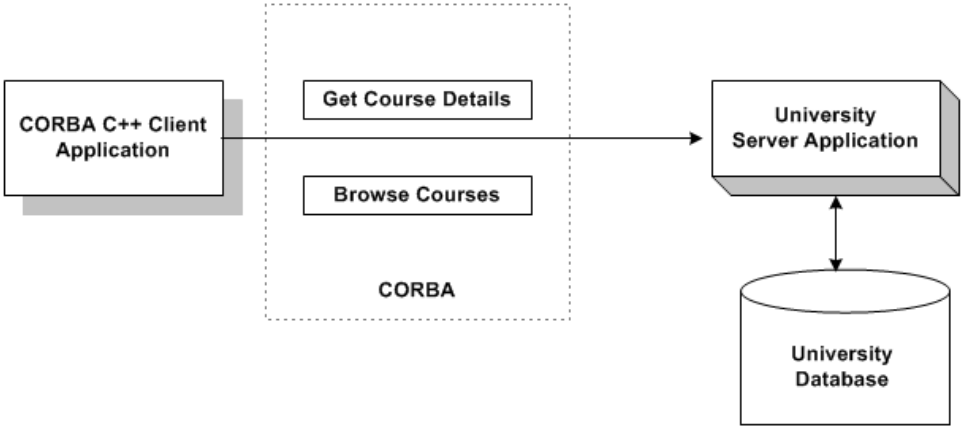
Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

For an explanation of concepts associated with CORBA applications and a description of the development process for CORBA applications, see *Getting Started with BEA Tuxedo CORBA Applications*.

How the Basic Sample Application Works

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 CORBA client and server applications
- Defining the configuration information for a CORBA application
- Building client and server applications using the CORBA commands and tools provided by the BEA Tuxedo product.

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:

Interface	Description	Operations
`RegistrarFactory`	Creates object references to the `Registrar` object.	`find_registrar()`
`Registrar`	Obtains course information from the database.	`get_courses_synopsis()` `get_courses_details()`
`CourseSynopsisEnumerator`	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.	`get_next_n()` `destroy()`

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

```
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();
};
  typedef 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,
           in unsigned long             number_to_get, // 0 = all
           out unsigned long            number_remaining,
           out CourseSynopsisEnumerator rest
        );
    CourseDetailsList get_courses_details(in CourseNumberList
         courses);

    interface RegistrarFactory
    {
         Registrar find_registrar(
         );
    };
};
```

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 CORBA Client Applications*.

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 with BEA Tuxedo CORBA Applications*.

During the development process, you would use the `idl` command to compile the OMG IDL file and produce client stubs and skeletons. This task has been automated in the `makefile` for the Basic sample application. For a description of the `idl` command, see the *BEA Tuxedo Command Reference*.

Writing the Client Application

The CORBA environment in BEA Tuxedo only supports CORBA C++ types of client applications.

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

- Initializes the ORB
- Uses the Bootstrap environmental object or the standard CORBA mechanism to establish communication with the BEA Tuxedo 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++ versions of the client application code in the Basic sample application are provided. For information about writing client applications, see *Getting Started with BEA Tuxedo CORBA Applications* and *Creating CORBA 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 BEA Tuxedo domain.
- The method implementations for the operations on the `Registrar`, `RegistrarFactory`, and `CourseSynopsisEnumerator` objects.

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 CORBA Server Applications*.

Configuring the Basic Sample Application

A key part of any CORBA application is the `UBBCONFIG` file. Although creating a `UBBCONFIG` file is the task of the administrator, it is important for programmers to understand that the file

exists and how the file is used. When system administrators create a configuration file, they are describing the CORBA application using a set of parameters that the BEA Tuxedo 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. "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 *Setting Up a BEA Tuxedo Application* and the *BEA Tuxedo Command Reference*.

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 "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

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

UNIX

```
/usr/TUXDIR/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

File	Description
`univb.idl`	The OMG IDL that declares the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univbs.cpp`	The C++ source code for the University server application in the Basic sample application.
`univb_i.h` `univb_i.cpp`	The C++ source code for method implementations of the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univbc.cpp`	The C++ source code for the CORBA C++ client application in the Basic sample application.
`univb_utils.h` `univb_utils.cpp`	The files that define database access functions for the CORBA C++ client application.
`univb.icf`	The Implementation Configuration File (ICF) for the Basic sample application.
`setenvb.sh`	A UNIX script that sets the environment variables needed to build and run the Basic sample application.
`setenvb.cmd`	An MS-DOS command procedure that sets the environment variables needed to build and run the Basic sample application.
`ubb_b.mk`	The configuration file for the UNIX operating system platform.
`ubb_b.nt`	The configuration file for the Windows operating system platform.

Table 3-1 Files Included in the Basic Sample Application (Continued)

File	Description
`makefileb.mk`	The `makefile` for the Basic sample application on the UNIX operating system platform.
`makefileb.nt`	The `makefile` for the Basic sample application on the Windows operating system platform.
`log.cpp`, `log.h`, `log_client.cpp`, and `log_server.cpp`	The client and server applications that provide logging and tracing functions for the sample applications. These files are located in the `\utils` directory.
`oradbconn.cpp` and `oranoconn.cpp`	The files that provide access to an Oracle SQL database instance. These files are located in the `\utils` directory.
`samplesdb.cpp` and `samplesdb.h`	The files that provide print functions for the database exceptions in the sample applications. These files are located in the `\utils` directory.
`unique_id.cpp` and `unique_id.h`	C++ Unique ID class routines for the sample applications. These files are located in the `\utils` directory.
`samplesdbsql.h` and `samplesdbsql.pc`	C++ class methods that implement access to the SQL database. These files are located in the `\utils` directory.
`university.sql`	The SQL for the University database. This file is located in the `\utils` directory.

Changing the Protection on the File for the Basic Sample Application

During the installation of the BEA Tuxedo 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

```
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

```
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

```
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

```
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 application in the Basic sample application:

Windows

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

UNIX

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

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 EventBroker.

- `TMFFNAME`

  The transaction management services, including the NameManager and the FactoryFinder services.

- `TMIFSRVR`

  The Interface Repository server process.

- `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
```

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.

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

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

For a full discussion of implementing security in a CORBA application, see *Using Security in CORBA Applications*.

How the Security Sample Application Works

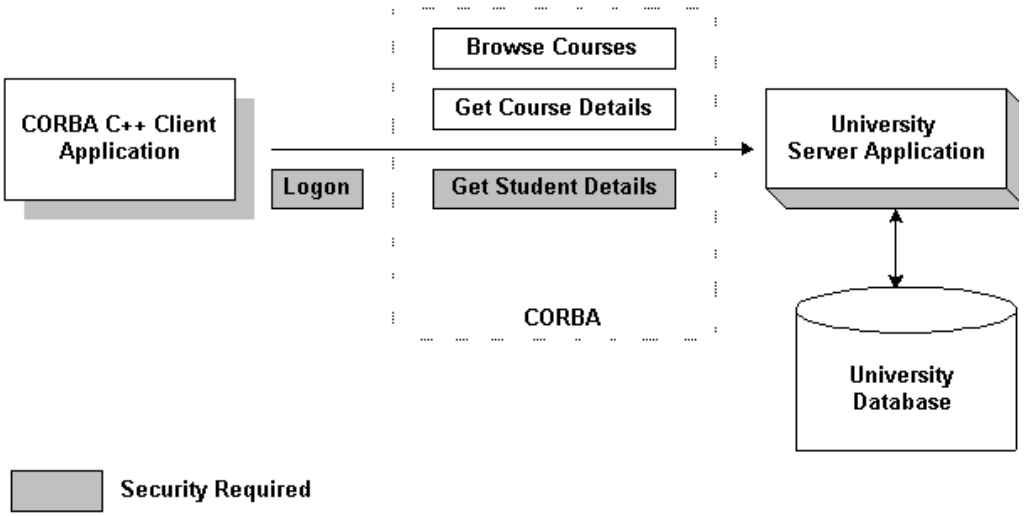
The Security sample application enhances the Basic sample application by adding application-level security to the CORBA application. Application-level security requires each student to have an ID and a password. Therefore, the concept of a Student is added to the Security sample application.

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 CORBA 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 BEA Tuxedo domain.
- The `Tobj::PrincipalAuthenticator` operation of the SecurityCurrent environmental object to return the type of authentication expected by the BEA Tuxedo domain.
- Operations to log on to the BEA Tuxedo 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 CORBA client applications, see *Using Security in CORBA Applications.*

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 CORBA Server Applications*.

The UBBCONFIG File

In the BEA Tuxedo software, security levels are defined for the configuration by the system administrator. The system administrator defines the security for the BEA Tuxedo domain by setting the `SECURITY` parameter `RESOURSES` 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 BEA Tuxedo domain, see *Setting Up a BEA Tuxedo Application* and *Using Security in CORBA Applications*.

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

drive:*TUXDIR*\samples\corba\university\security

UNIX

/usr/*TUXDIR*/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

File	Description
`univs.idl`	The OMG IDL that declares the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univss.cpp`	The C++ source code for the University server application in the Security sample application.
`univs_i.h` `univs_i.cpp`	The C++ source code for method implementations of the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univsc.cpp`	The C++ source code for the CORBA C++ client application in the Security sample application.
`univs_utils.h` `univs_utils.cpp`	The files that define database access functions for the CORBA C++ client application.

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

File	Description
`univs.icf`	The Implementation Configuration File (ICF) for the Security sample application.
`setenvs.sh`	A UNIX script that sets the environment variables needed to build and run the Security sample application.
`setenvs.cmd`	An MS-DOS command procedure that sets the environment variables needed to build and run the Security sample application.
`ubb_s.mk`	The `UBBCONFIG` file for the UNIX operating system.
`ubb_s.nt`	The `UBBCONFIG` file for the Windows operating system.
`makefiles.mk`	The `makefile` for the Security sample application on the UNIX operating system.
`makefiles.nt`	The `makefile` for the Security sample application on the Windows operating system.
`log.cpp`, `log.h`, `log_client.cpp`, and `log_server.cpp`	The client and server applications that provide logging and tracing functions for the sample applications. These files are located in the `\utils` directory.
`oradbconn.cpp` and `oranoconn.cpp`	The files that provide access to an Oracle SQL database instance. These files are located in the `\utils` directory.
`samplesdb.cpp` and `samplesdb.h`	The files that provide print functions for the database exceptions in the sample applications.These files are located in the `\utils` directory.
`unique_id.cpp` and `unique_id.h`	C++ Unique ID class routines for the sample applications.These files are located in the `\utils` directory.

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

File	Description
`samplesdbsql.h` and `samplesdbsql.pc`	C++ class methods that implement access to the SQL database. These files are located in the `\utils` directory.
`university.sql`	The SQL for the University database. This file is located in the `\utils` directory.

Changing the Protection on the Files for the Security Sample Application

During the installation of the BEA Tuxedo 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:

Windows

```
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 Security sample applications:

Windows

```
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

```
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:

Windows

```
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

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

UNIX

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

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 EventBroker.

- `TMFFNAME`

  The transaction management services, including the NameManager and the FactoryFinder services.

- `TMIFSRVR`

  The Interface Repository server process.

- `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.

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:

```
<L>    List your registered courses
```

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

CHAPTER 5

The Transactions Sample Application

This topic includes the following sections:

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

For a complete discussion of using transactions in a CORBA application, see *Using CORBA Transactions*.

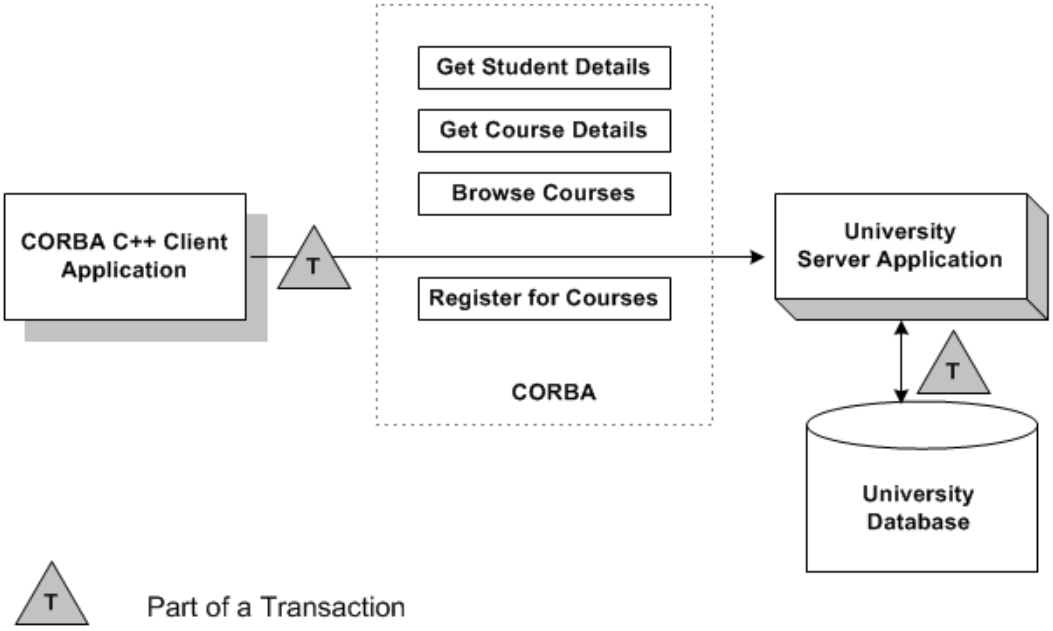
How the Transactions Sample Application Works

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 BEA Tuxedo 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 with BEA Tuxedo CORBA Applications* and *Using CORBA 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
- A method implementation for the `register_for_courses()` operation

For information about these tasks, see *Creating CORBA 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 *Setting Up a BEA Tuxedo Application*.

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 CORBA 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

drive:*TUXDIR*\samples\corba\university\transaction

UNIX

/usr/*TUXDIR*/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

File	Description
`univt.idl`	The OMG IDL that declares the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univts.cpp`	The C++ source code for the University server application in the Transactions sample application.
`univt_i.h` `univt_i.cpp`	The C++ source code for method implementations of the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univtc.cpp`	The C++ source code for the CORBA C++ client application in the Transactions sample application.
`univt_utils.h` `univt_utils.cpp`	The files that define database access functions for the CORBA C++ client application.
`univt.icf`	The ICF file for the Transactions sample application.
`setenvt.sh`	A UNIX script that sets the environment variables needed to build and run the Transactions sample application.
`setenvt.cmd`	An MS-DOS command procedure that sets the environment variables needed to build and run the Transactions sample application.
`ubb_t.mk`	The `UBBCONFIG` file for the UNIX operating system.
`ubb_t.nt`	The `UBBCONFIG` file for the Windows operating system.

Table 5-1 Files Included in the Transactions Sample Application (Continued)

File	Description
`makefilet.mk`	The `makefile` for the Transactions sample application on the UNIX operating system.
`makefilet.nt`	The `makefile` for the Transactions sample application on the Windows operating system.
`log.cpp`, `log.h`, `log_client.cpp`, and `log_server.cpp`	The client and server applications that provide logging and tracing functions for the sample applications. These files are located in the `\utils` directory.
`oradbconn.cpp` and `oranoconn.cpp`	The files that provide access to an Oracle SQL database instance. These files are located in the `\utils` directory.
`samplesdb.cpp` and `samplesdb.h`	The files that provide print functions for the database exceptions in the sample applications. These files are located in the `\utils` directory.
`unique_id.cpp` and `unique_id.h`	C++ Unique ID class routines for the sample applications. These files are located in the `\utils` directory.
`samplesdbsql.h` and `samplesdbsql.pc`	C++ class methods that implement access to the SQL database. These files are located in the `\utils` directory.
`university.sql`	The SQL for the University database. This file is located in the `\utils` directory.

Changing the Protection on the Files for the Transactions Sample Application

During the installation of the BEA Tuxedo 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

```
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 Transactions sample application:

Windows

```
prompt>setenvt
```

UNIX

```
prompt>/bin/ksh
```

```
prompt>. ./setenvt.sh
```

Initializing the University Database

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

Windows

```
prompt>nmake -f makefilet.nt initdb
```

UNIX

```
prompt>make -f makefilet.mk initdb
```

Loading the UBBCONFIG File

Use the following command to load the `UBBCONFIG` file:

Windows

```
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 CORBA 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:

   ```
   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

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

UNIX

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

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:

```
prompt>tmboot -y
```

This command starts the following server processes:

- `TMSYSEVT`

  The BEA Tuxedo system EventBroker.

- `TMFFNAME`

  The transaction management services, including the NameManager and the FactoryFinder services.

- `TMIFSRVR`

  The Interface Repository server process.

- `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.

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.

CHAPTER 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

Notes: The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

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.

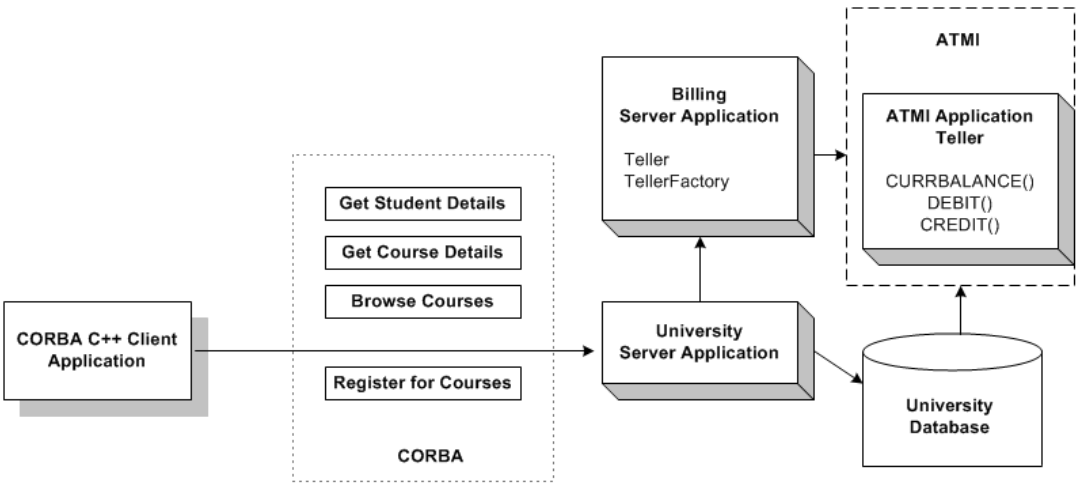
The Wrapper sample application incorporates a ATMI service. The Billing server application contains a `Teller` object, which calls the ATMI application Teller. The Teller application performs the following billing operations:

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

Object	Description	Operations
`TellerFactory`	Returns an object reference to the Teller object	`find_teller()`

`Teller`	Interoperates with the ATMI application Teller to perform billing and accounting operations	`get_balance()`
		`credit()`
		`debit()`

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 `Delinquent Account` that the `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 `get_balance()`, `credit()`, and `debit()` operations for the `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 ATMI application Teller.
  - Calls the ATMI application Teller.
  - Extracts information from the FML message buffer returned from the ATMI application Teller.
  - Returns the information from the FML message buffer to the University server application.
- A method implementation for the `find_teller()` operation of the `TellerFactory` object.
- A Billing server object that creates and registers the `TellerFactory` object and calls the `open_XA_RM` and `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 `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 ATMI services, see *Creating CORBA 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 Teller application, and the server application for the University database. This server group allows both the University server application and the 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. ATMI application Teller
  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 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 CORBA 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

drive:*TUXDIR*\samples\corba\university\wrapper

UNIX

/usr/*TUXDIR*/samples/corba/university/wrapper

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

File	Description
`billw.idl`	The OMG IDL that declares the `Teller` and `TellerFactory` interfaces.
`univw.idl`	The OMG IDL that declares the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`billws.cpp`	The C++ source code for the Billing server application in the Wrapper sample application.
`univws.cpp`	The C++ source code for the University server application in the Wrapper sample application.
`billw__i.h` `billw_i.cpp`	The C++ source code for the method implementations of the `Teller` and `TellerFactory` interfaces.
`univw_i.h` `univw_i.cpp`	The C++ source code for the method implementations of the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univwc.cpp`	The C++ source code for the CORBA C++ client application in the Wrapper sample application.
`univw_utils.h` `univw_utils.cpp`	The files that define database access functions for the CORBA C++ client application.
`univw.icf`	The ICF file for the University server application in the Wrapper sample application.

Table 6-1 Files Included in the Wrapper Sample Application (Continued)

File	Description
billw.icf	The ICF file for the Billing server application in the Wrapper sample application.
setenvw.sh	A UNIX script that sets the environment variables needed to build and run the Wrapper sample application.
tellw_flds, tellw_u.c, tellw_c.h, tellws.ec	The files for the ATMI application Teller.
setenvw.cmd	An MS-DOS command procedure that sets the environment variables needed to build and run the Wrapper sample application.
ubb_w.mk	The UBBCONFIG file for the UNIX operating system.
ubb_w.nt	The UBBCONFIG file for the Windows operating system.
makefilew.mk	The makefile for the Wrapper sample application on the UNIX operating system.
makefilew.nt	The makefile for the Wrapper sample application on the Windows operating system.
log.cpp, log.h, log_client.cpp, and log_server.cpp	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.
oradbconn.cpp and oranoconn.cpp	The files that provide access to an Oracle SQL database instance. These files are located in the \utils directory.
samplesdb.cpp and samplesdb.h	The files that provide print functions for the database exceptions in the sample applications. These files are located in the \utils directory.
unique_id.cpp and unique_id.h	C++ Unique ID class routines for the sample applications.These files are located in the \utils directory.

Table 6-1 Files Included in the Wrapper Sample Application (Continued)

File	Description
`samplesdbsql.h` and `samplesdbsql.pc`	C++ class methods that implement access to the SQL database. These files are located in the `\utils` directory.
`university.sql`	The SQL for the University database. This file is located in the `\utils` directory.

Changing the Protection on the Files for the Wrapper Sample Application

During the installation of the BEA Tuxedo 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

```
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

```
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

```
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

```
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 CORBA 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:

```
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

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

UNIX

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

Running the Wrapper Sample Application

To run the Wrapper 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 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 EventBroker.

- `TMFFNAME`

The transaction management services, including the NameManager and the FactoryFinder services.

- `TMIFSRVR`

  The Interface Repository server process.

- `univw_server`

  The University server process.

- `tellw_server`

  The application process for the ATMI application Teller.

- `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:

```
prompt>tmshutdown
```

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.

Using the Client Applications in the Wrapper Sample Application

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.

CHAPTER 7

The Production Sample Application

This topic includes the following sections:

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

The BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB were deprecated in Tuxedo 8.1 and are no longer supported. All BEA Tuxedo CORBA Java client and BEA Tuxedo CORBA Java client ORB text references, associated code samples, should only be used to help implement/run third party Java ORB libraries, and for programmer reference only.

Technical support for third party CORBA Java ORBs should be provided by their respective vendors. BEA Tuxedo does not provide any technical support or documentation for third party CORBA Java ORBs.

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 CORBA features of the BEA Tuxedo software to scale a CORBA application. The Production sample application does the following:

- Replicates the University server application, the Billing server application, and the ATMI Teller application within the `ORA_GRP` and `APP_GRP` server groups defined in the `UBBCONFIG` 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 BEA Tuxedo 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:

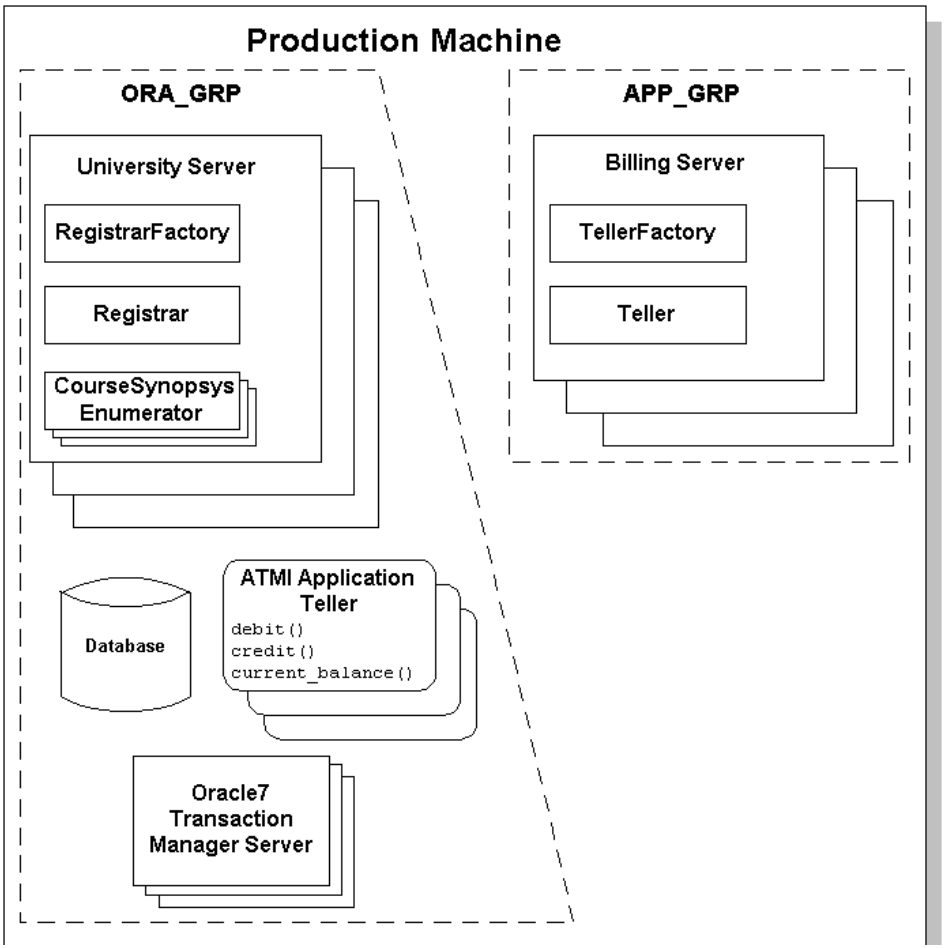
- You obtain a means to balance the load of incoming requests from client applications on that server application. As requests arrive in the Tuxedo domain for the server group, the BEA Tuxedo system routes the request to the least busy server application within that group.
- You can specify how many copies of a given server application process are running on a server machine. The number of copies determines the extent to which the Tuxedo 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 ATMI 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.

Figure 7-1 Replicated Server Groups in the Production Sample Application



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

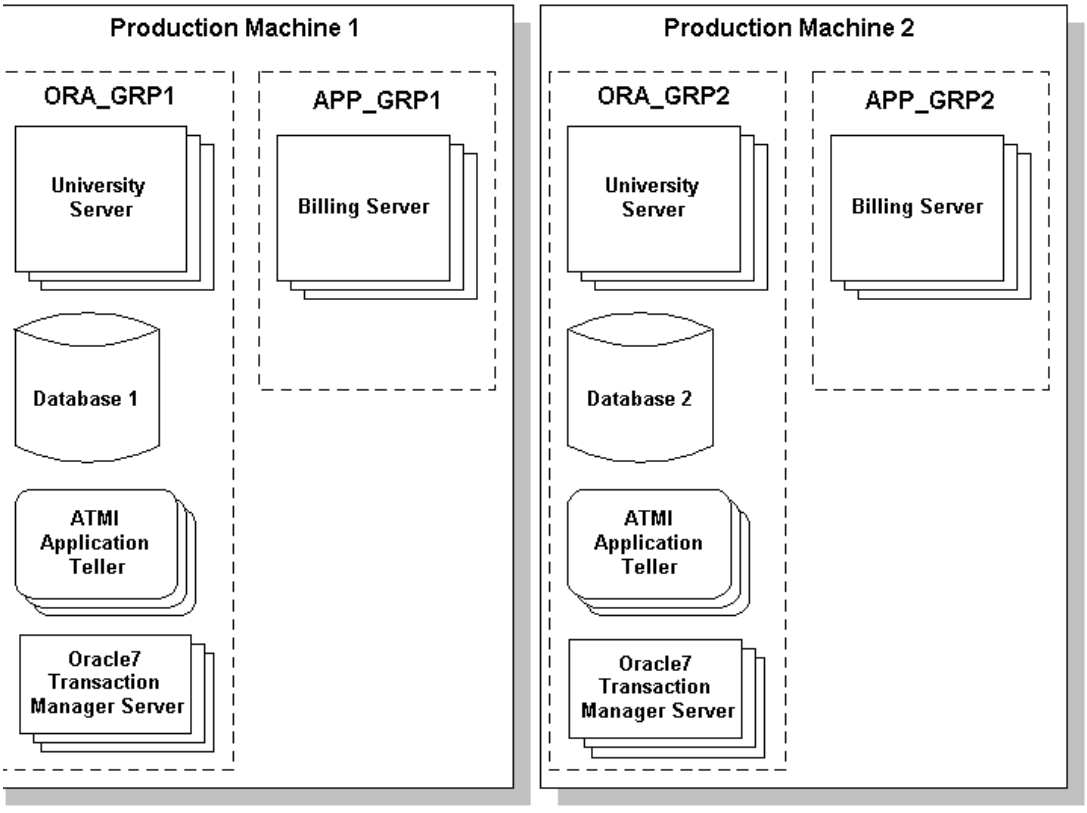
Server groups are a feature of the BEA Tuxedo software that allow you to add server machines to an existing CORBA application. When you replicate a server group, you can do the following:

- Spread the processing load for a CORBA 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.

Figure 7-2 Replicating Server Groups Across Server Machines



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 Tuxedo domain in the appropriate server group.
- After the invocation is complete, the Tuxedo 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 CORBA 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 CORBA Server Applications*.

Using Factory-based Routing

Factory-based routing is a CORBA 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 CORBA 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 CORBA Server Applications*.

The Development Process for the Production Sample Application

This section describes the development process required when scaling a CORBA 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.

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

The UBBCONFIG File

The UBBCONFIG file is the key to achieving scalability in a CORBA 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 CORBA 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 following example shows lines from the GROUPS and SERVERS sections of the UBBCONFIG file for the Production sample application.

```
*GROUPS
    APP_GRP1
        LMID      = SITE1
        GRPNO     = 2
        TMSNAME   = TMS
    APP_GRP2
        LMID      = SITE1
        GRPNO     = 3
        TMSNAME   = TMS
    ORA_GRP1
        LMID      = SITE1
        GRPNO     = 4
        OPENINFO  = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir
                     =.+MaxCur=5"
        CLOSEINFO = ""
        TMSNAME   = "TMS_ORA"
    ORA_GRP2
        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
    SRVID   = 10
    RESTART = N
univp_server
    SRVGRP  = ORA_GRP1
    SRVID   = 20
    RESTART = N
univp_server
    SRVGRP  = ORA_GRP2
    SRVID   = 20
    RESTART = N
```

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 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 CORBA Server Applications*.

Building the Production Sample Application

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 `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 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

drive`:\`*TUXDIR*`\samples\corba\university\production`

UNIX

`/usr/`*TUXDIR*`/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

File	Description
`billp.idl`	The OMG IDL that declares the `Teller` and `TellerFactory` interfaces.
`univp.idl`	The OMG IDL that declares the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`billps.cpp`	The C++ source code for the Billing server application in the Production sample application.
`univps.cpp`	The C++ source code for the University server application in the Production sample application.
`billp__i.h` `billp_i.cpp`	The C++ source code for the method implementations of the `Teller` and `TellerFactory` interfaces.
`univp_i.h` `univp_i.cpp`	The C++ source code for method implementations of the `CourseSynopsisEnumerator`, `Registrar`, and `RegistrarFactory` interfaces.
`univpc.cpp`	The C++ source code for the CORBA C++ client application in the Production sample application.
`univp_utils.h` `univp_utils.cpp`	The files that define database access functions for the CORBA C++ client application.
`univp.icf`	The Implementation Configuration File (ICF) for the University server application in the Production sample application.
`billp.icf`	The ICF file for the Billing server application in the Production sample application.
`tellw_flds, tellw_u.c, tellw_c.h, tellws.ec`	The files for the ATMI application Teller.
`setenvp.sh`	A UNIX script that sets the environment variables needed to build and run the Production sample application.

Table 7-1 Files Included in the Production Sample Application (Continued)

File	Description
`setenvp.cmd`	An MS-DOS command procedure that sets the environment variables needed to build and run the Production sample application.
`ubb_p.mk`	The `UBBCONFIG` file for the UNIX operating system.
`ubb_p.nt`	The `UBBCONFIG` file for the Windows operating system.
`makefilep.mk`	The `makefile` for the Production sample application on the UNIX operating system.
`makefilep.nt`	The `makefile` for the Production sample application on the Windows operating system.
`log.cpp`, `log.h`, `log_client.cpp`, and `log_server.cpp`	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.
`oradbconn.cpp` and `oranoconn.cpp`	The files that provide access to an Oracle SQL database instance.These files are located in the `\utils` directory.
`samplesdb.cpp` and `samplesdb.h`	The files that provide print functions for the database exceptions in the sample applications. These files are located in the `\utils` directory.
`unique_id.cpp` and `unique_id.h`	C++ Unique ID class routines for the sample applications.These files are located in the `\utils` directory.
`samplesdbsql.h` and `samplesdbsql.pc`	C++ class methods that implement access to the SQL database. These files are located in the `\utils` directory.
`university.sql`	The SQL for the University database. This file is located in the `\utils` directory.

Changing the Protection on the Files for the Production Sample Application

During the installation of the BEA Tuxedo 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

```
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

```
prompt>setenvp
```

UNIX

```
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

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

UNIX

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

Loading the UBBCONFIG File

Use the following command to load the `UBBCONFIG` file:

Windows

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

UNIX

```
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 CORBA 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:

   ```
   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:

   ```
   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

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

UNIX

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 EventBroker.

- `TMFFNAME`

  The transaction management services, including the NameManager and the FactoryFinder services.

- `TMIFSRVR`

  The Interface Repository server process.

  Four processes of the University server application.

- `tellp_server`

  Four processes of the ATMI application Teller.

- `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 A type library. By default, the type library is placed in `\`*`TUXDIR`*`\TypeLibraries`.

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 CORBA 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.

Index

C

D

M

N

O

P

R

S

T

U

W

X