BEA WebLogic Enterprise

Guide to the Java Sample Applications

WebLogic Enterprise 5.0
Document Edition 5.0
December 1999
Contents

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

1. Introduction
Overview of the Sample Applications ........................................ 1-1

2. The Java Simpapp Sample Application
How the Java Simpapp Sample Application Works...................... 2-2
Software Prerequisites .................................................................. 2-3
The OMG IDL Code for the Java Simpapp Sample Application ....... 2-3
Building and Running the Java Simpapp Sample Application ........ 2-4
  Copying the Files for the Java Simpapp Sample Application into a Work
  Directory .................................................................................. 2-5
  Changing the Protection Attribute on the Files for the Java Simpapp Sample
  Application .............................................................................. 2-7
  Verifying the Settings of the Environment Variables ...................... 2-7
  Executing the runme Command .................................................. 2-9
Using the Java Simpapp Sample Application .................................................. 2-15
Using the C++ Client Application with the Java Simpapp Sample Application .... 2-16
Stopping the Java Simpapp Sample Application ............................................. 2-17

3. The JDBC Bankapp Sample Application

How the JDBC Bankapp Sample Application Works ...................................... 3-2
  Java Server Objects .................................................................................... 3-2
  Application Workflow ................................................................................ 3-2
  JDBC Connection Pooling ......................................................................... 3-3
Development Process for the JDBC Bankapp Sample Application .......... 3-4
  Object Management Group (OMG) Interface Definition Language (IDL) .... 3-4
    BankApp.idl File ................................................................................. 3-5
    BankDB.idl File ................................................................................. 3-6
    Bank.idl File ...................................................................................... 3-7
  Client Application ................................................................................... 3-8
  Server Application .................................................................................... 3-8
  Server Description File (BankApp.xml) .................................................. 3-9
  UBBCONFIG File .................................................................................... 3-9
    Enabling Multithreaded Support ......................................................... 3-10
    Setting Up the Connection Pool .......................................................... 3-10
Setting Up the Database for the JDBC Bankapp Sample Application ........ 3-12
  Setting Up an Oracle Database ............................................................. 3-12
  Setting Up a Microsoft SQL Server Database ........................................... 3-13
Building the JDBC Bankapp Sample Application ....................................... 3-13
  Step 1: Copy the Files for the JDBC Bankapp Sample Application into a
    Work Directory .................................................................................. 3-14
    Source File Directories ..................................................................... 3-14
    Copying Source Files to the Work Directory ......................................... 3-15
    Source Files Used to Build the JDBC Bankapp Sample Application .... 3-15
  Step 2: Change the Protection Attribute on the Files for the JDBC Bankapp
    Sample Application ............................................................................ 3-17
  Step 3: Verify the Settings of the Environment Variables ...................... 3-18
    Environment Variables ....................................................................... 3-18
    Verifying Settings .............................................................................. 3-19
Guide to the Java Sample Applications

4. The XA Bankapp Sample Application

How the XA Bankapp Sample Application Works ........................................... 4-2
Server Applications ...................................................................................... 4-2
Application Workflow .................................................................................. 4-2
Software Prerequisites ................................................................................... 4-3
Development Process for the XA Bankapp Sample Application ......................... 4-4
Object Management Group (OMG) Interface Definition Language (IDL) ... 4-4
Client Application ......................................................................................... 4-4
Server Application ......................................................................................... 4-5
Server Description File .................................................................................... 4-5
Implementation Configuration File ................................................................. 4-5
UBBCONFIG File .................................................................................................. 4-6

Guide to the Java Sample Applications v
Setting Up the Database for the XA Bankapp Sample Application .......... 4-6
Building the XA Bankapp Sample Application ........................................... 4-7
Step 1: Copy the Files for the XA Bankapp Sample Application into a Work Directory ............................................................................................ 4-7
Source File Directories ........................................................................ 4-7
Copying Source Files to the Work Directory ........................................ 4-8
Source Files Used to Build the XA Bankapp Sample Application .... 4-9
Step 2: Change the Protection Attribute on the Files for the XA Bankapp Sample Application ................................................................. 4-10
Step 3: Verify the Settings of the Environment Variables ...................... 4-11
Environment Variables ..................................................................... 4-11
Verifying Settings ............................................................................ 4-12
Changing Settings ............................................................................ 4-12
Step 4: Run the setupX Command .......................................................... 4-13
Step 5: Load the UBBCONFIG File ....................................................... 4-13
Step 6: Create a Transaction Log ............................................................ 4-14
Compiling the Client and Server Applications ......................................... 4-14
Initializing the Oracle Database .......................................................... 4-15
Starting the Server Application in the XA Bankapp Sample Application .... 4-15
Files Generated by the XA Bankapp Sample Application ...................... 4-16
Starting the ATM Client Application in the XA Bankapp Sample Application ... 4-20
Stopping the XA Bankapp Sample Application ...................................... 4-21
Using the ATM Client Application ......................................................... 4-21

Index
This document describes the Java sample applications that are provided with the BEA WebLogic Enterprise (sometimes referred to as WLE) software and is intended to be used with the following documents:

♦ Getting Started
♦ Creating Client Applications
♦ Creating Java Server Applications

Note: Effective February 1999, the BEA M3 product is renamed. The new name of the product is BEA WebLogic Enterprise (WLE).

This document covers the following topics:

♦ Chapter 1, “Introduction,” provides an overview of the sample applications.
♦ Chapter 2, “The Java Simpapp Sample Application,” describes how to build and use the Java Simpapp sample application.
♦ Chapter 3, “The JDBC Bankapp Sample Application,” describes how to build and use the JDBC Bankapp sample application.
♦ Chapter 4, “The XA Bankapp Sample Application,” describes how to build and use the XA Bankapp sample application.
What You Need to Know

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

e-docs Web Site

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

How to Print the Document

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

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

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

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

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.0 release.

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

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

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

The following documentation conventions are used throughout this document.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface text</strong></td>
<td>Indicates terms defined in the glossary.</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Indicates that you must press two or more keys simultaneously.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Indicates emphasis or book titles.</td>
</tr>
<tr>
<td><strong>monospace text</strong></td>
<td>Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.</td>
</tr>
<tr>
<td>Examples:</td>
<td></td>
</tr>
<tr>
<td>#include &lt;iostream.h&gt; void main ( ) the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</td>
<td></td>
</tr>
<tr>
<td><strong>monospace boldface text</strong></td>
<td>Identifies significant words in code.</td>
</tr>
<tr>
<td>Example:</td>
<td>void commit ( )</td>
</tr>
<tr>
<td><strong>monospace italic text</strong></td>
<td>Identifies variables in code.</td>
</tr>
<tr>
<td>Example:</td>
<td>String expr</td>
</tr>
<tr>
<td>UPPERCASE TEXT</td>
<td>Indicates device names, environment variables, and logical operators.</td>
</tr>
<tr>
<td>Examples:</td>
<td>LPT1 SIGNON OR</td>
</tr>
</tbody>
</table>
### Documentation Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ }</code></td>
<td>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</td>
</tr>
</tbody>
</table>
| `[ ]`      | Indicates optional items in a syntax line. The brackets themselves should never be typed.  

*Example:*

```
buildobjclient [-v] [-o name ] [-f file-list]...
[-l file-list]...
```

| `|` | Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. |
| `...` | Indicates one of the following in a command line:  
  n That an argument can be repeated several times in a command line  
  n That the statement omits additional optional arguments  
  n 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. |
This chapter provides an overview of the Java sample applications

Overview of the Sample Applications

The sample applications provide client and server programmers with the basic concepts of developing Java server applications for the Weblogic Enterprise (WLE) software.

The following sample applications are provided:

- Java Simpapp—provides a Java client application and a Java server application. The Java server application contains two operations that manipulate strings received from the Java client application.

- JDBC Bankapp—implements an automatic teller machine (ATM) interface and uses Java Database Connectivity (JDBC) to access a database that stores account and customer information.

- XA Bankapp—implements the same ATM interface as JDBC Bankapp; however, XA Bankapp uses a database XA library to demonstrate using the Transaction Manager to coordinate transactions.

The chapters in this manual describe how to build and run the sample applications. For a description of the development process used to create the sample applications, see Getting Started.
Introduction
The chapter discusses the following topics:

- How the Java Simpapp sample application works
- Software prerequisites
- The Object Management Group (OMG) Interface Definition Language (IDL) for the Java Simpapp sample application
- Building and running the Java Simpapp sample application
- Using the Java Simpapp sample application
- Using the C++ client application with the Java Simpapp sample application
- Stopping the Java Simpapp Sample Application

Refer to Readme.txt in the \WLEdir\samples\corba\simpapp_java directory for troubleshooting information and the latest information about using the Java Simpapp sample application.
How the Java Simpapp Sample Application Works

The Java Simpapp sample application consists of a Java client application that sends requests to a Java server application. The Java server application provides an implementation of a CORBA object that has the following two methods:

- The `upper` method accepts a string from the Java client application and converts the string to uppercase letters.
- The `lower` method accepts a string from the Java client application and converts the string to lowercase letters.

Figure 2-1 illustrates how the Java Simpapp sample application works.

Figure 2-1   The Java Simpapp Sample Application
Software Prerequisites

To run the `idltojava` compiler used by the Java Simpapp sample application, you need to install Visual C++ Version 5.0 with Service Pack 3 for Visual Studio.

The OMG IDL Code for the Java Simpapp Sample Application

The Java Simpapp sample application implements the CORBA interfaces listed in Table 2-1:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory</td>
<td>Creates object references to the Simple object</td>
<td><code>find_simple()</code></td>
</tr>
</tbody>
</table>
| Simple      | Converts the case of a string | `to_upper()`  
|             |                              | `to_lower()`  |

Listing 2-1 shows the `simple.idl` file that defines the CORBA interfaces in the Java Simpapp sample application. This is the same OMG IDL file used by the C++ Simpapp sample application shipped with version 4.2 of the WebLogic Enterprise (WLE) software. The `runme` command automatically copies it from the `\corba\simpapp_java` directory.
Listing 2-1  OMG IDL Code for the Java Simpapp Sample Application

```idl
#pragma prefix "beasys.com"

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

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

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

Building and Running the Java Simpapp Sample Application

Perform the following steps to build and run the Java Simpapp sample application:

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

The following sections describe these steps.
Copying the Files for the Java Simpapp Sample Application into a Work Directory

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

**Windows NT**

`drive:\WLEdir\samples\corba\simpapp_java`

**UNIX**

`/usr/local/WLedir/samples/corba/simapp_java`

You will use the files listed in Table 2-2 to build and run the Java Simpapp sample application.

**Table 2-2  Files Included in the Java Simpapp Sample Application**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.idl</td>
<td>The OMG IDL code that declares the Simple and SimpleFactory interfaces. This file is copied from the WLE simpapp_java directory by the runme command file.</td>
</tr>
<tr>
<td>ServerImpl.java</td>
<td>The Java source code that overrides the Server.initialize and Server.release methods.</td>
</tr>
<tr>
<td>SimpleClient.java</td>
<td>The Java source code for the client application in the Java Simpapp sample application.</td>
</tr>
<tr>
<td>SimpleFactoryImpl.java</td>
<td>The Java source code that implements the SimpleFactory methods.</td>
</tr>
<tr>
<td>SimpleImpl.java</td>
<td>The Java source code that implements the Simple methods.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Simple.xml</td>
<td>The Server Description File used to associate activation and transaction policy values with CORBA interfaces. For the Java Simpapp sample application, the Simple and SimpleFactory interfaces have an activation policy of method and a transaction policy of optional.</td>
</tr>
<tr>
<td>Readme.txt</td>
<td>Provides the latest information about building and running the Java Simpapp sample application.</td>
</tr>
<tr>
<td>runme.cmd</td>
<td>The Windows NT batch file that builds and runs the Java Simpapp sample application.</td>
</tr>
<tr>
<td>runme.ksh</td>
<td>The UNIX Korn shell script that builds and executes the Java Simpapp sample application.</td>
</tr>
<tr>
<td>makefile.mk</td>
<td>The make file for the Java Simpapp sample application on the UNIX operating system. This file is used to manually build the Java Simpapp sample application. Refer to the Readme.txt file for information about manually building the Java Simpapp sample application. The UNIX make command needs to be in the path of your machine.</td>
</tr>
<tr>
<td>makefiles.nt</td>
<td>The make file for the Java Simpapp sample application on the Windows NT operating system. This make file can be used directly by the Visual C++ nmake command. This file is used to manually build the Java Simpapp sample application. Refer to the Readme.txt file for information about manually building the Java Simpapp sample application. The Windows NT nmake command needs to be in the path of your machine.</td>
</tr>
<tr>
<td>smakefile.nt</td>
<td>The make file for the Java Simpapp sample application that is used with Visual Cafe smake command.</td>
</tr>
</tbody>
</table>

**Note:** makefile.nt is included by smakefile.nt.
Changing the Protection Attribute on the Files for the Java Simpapp Sample Application

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

**Windows NT**

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

**UNIX**

`prompt>/bin/ksh`

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

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

`ksh prompt>chmod +x runme.ksh`

Verifying the Settings of the Environment Variables

Before building and running the Java Simpapp sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure they reflect correct information.
Table 2-3 lists the environment variables required to run the Java Simpapp sample application.

Table 2-3  Required Environment Variables for the Java Simpapp Sample Application

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUXDIR</td>
<td>The directory path where you installed the WLE software. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>TUXDIR=c:\WLEdir</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>TUXDIR=/usr/local/WLEdir</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>The directory path where you installed the JDK software. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>JAVA_HOME=c:\JDK1.2</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>JAVA_HOME=/usr/local/JDK1.2</td>
</tr>
</tbody>
</table>

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

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.
3. Click the System icon.
   The System Properties window appears.
4. Click the Environment tab.
   The Environment page appears.
5. Check the settings for TUXDIR and JAVA_HOME.
UNIX

ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME

To change the settings, perform the following steps:

Windows NT

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

UNIX

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

Executing the runme Command

The runme command automates the following steps:

1. Setting the system environment variables
2. Loading the UBBCONFIG file
3. Compiling the code for the client application
4. Compiling the code for the server application
5. Starting the server application using the tmboot command
6. Starting the client application
7. Stopping the server application using the tmshutdown command

Note: You can also run the Java Simpapp sample application manually. The steps for manually running the Java Simpapp sample application are described in the Readme.txt file.
To build and run the Java Simpapp sample application, enter the `runme` command, as follows:

**Windows NT**
```
prompt>cd workdirectory
prompt>runme
```

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

The Java Simpapp sample application runs and prints the following messages:
```
Testing simpapp
  cleaned up
  prepared
  built
  loaded ubb
  booted
  ran
  shutdown
  saved results
PASSED
```

**Note:** After executing the `runme` command, you may get a message indicating the Host, Port, and IPCKEY parameters in the UBBCONFIG file conflict with an existing UBBCONFIG file. If this occurs, you need to set these parameters to different values to get the Java Simpapp sample application running on your machine. See the `Readme.txt` file for information about how to set these parameters.

The `runme` command starts the following application processes:

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

- **TMFFNAME**
  The following three TMFFNAME server processes are started:

  - The TMFFNAME server process started with the `-N` and `-M` options is the master NameManager service. The NameManager service maintains a mapping of the application-supplied names to object references.
The TMFFNAME server process started with only the \texttt{-N} option is the slave NameManager service.

The TMFFNAME server process started with the \texttt{-F} option contains the FactoryFinder object.

\textbf{JavaServer}

The Java Simpapp sample application server process. The JavaServer process has one option, \texttt{simple}, which is the Java Archive (JAR) file that was created for the application.

\textbf{ISL}

The IIOP Listener process.

Table 2-4 lists the files in the work directory generated by the \texttt{runme} command.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{File} & \textbf{Description} \\
\hline
SimpleFactory.java & Generated by the \texttt{m3idltojava} command for the \texttt{SimpleFactory} interface. The \texttt{SimpleFactory} interface contains the Java version of the OMG IDL interface. It extends \texttt{org.omg.CORBA.Object}. \\
\hline
SimpleFactoryHolder.java & Generated by the \texttt{m3idltojava} command for the \texttt{SimpleFactory} interface. This class holds a public instance member of type \texttt{SimpleFactory}. The class provides operations for \texttt{out} and \texttt{inout} arguments that are included in CORBA, but that do not map exactly to Java. \\
\hline
SimpleFactoryHelper.java & Generated by the \texttt{m3idltojava} command for the \texttt{SimpleFactory} interface. This class provides auxiliary functionality, notably the \texttt{narrow} method. \\
\hline
_SimpleFactoryStub.java & Generated by the \texttt{m3idltojava} command for the \texttt{SimpleFactory} interface. This class is the client stub that implements the \texttt{SimpleFactory.java} interface. \\
\hline
\end{tabular}
\caption{Files Generated by the \texttt{runme} Command}
\end{table}
Table 2-4  Files Generated by the  runme  Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SimpleFactoryImplBase.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This abstract class is the server skeleton. It implements the SimpleFactory.java interface. The user-written server class SimpleFactoryImpl extends _SimpleFactoryImplBase.</td>
</tr>
<tr>
<td>Simple.java</td>
<td>Generated by the m3idltojava command for the Simple interface. The Simple interface contains the Java version of the OMG IDL interface. It extends org.omg.CORBA.Object.</td>
</tr>
<tr>
<td>SimpleHolder.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class holds a public instance member of type Simple. The class provides operations for out and inout arguments that CORBA has but that do not match exactly to Java.</td>
</tr>
<tr>
<td>SimpleHelper.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class provides auxiliary functionality, notably the narrow method.</td>
</tr>
<tr>
<td>_SimpleStub.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class is the client stub that implements the Simple.java interface.</td>
</tr>
<tr>
<td>_SimpleImplBase.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This abstract class is the server skeleton. It implements the Simple.java interface. The user-written server class SimpleImpl extends _SimpleImplBase.</td>
</tr>
<tr>
<td>Simple.ser</td>
<td>The Server Descriptor File generated by the buildjobjserver command in the runme command.</td>
</tr>
<tr>
<td>Simple.jar</td>
<td>The server Java Archive file generated by the buildjavaserver command in the runme command.</td>
</tr>
</tbody>
</table>
Table 2-4  Files Generated by the *runme* Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleClient.jar</td>
<td>The Java Archive file for the client application. It can be used to verify. This file is used during the installation of the WLE software to insure the client application is installed properly. For information about verifying the installation of the WLE software, see <em>Installing the WebLogic Enterprise Software</em>.</td>
</tr>
<tr>
<td>.adm/.keybd</td>
<td>A file that contains the security encryption key database. The subdirectory is created by the <code>tmloadcf</code> command in the <code>runme</code> command.</td>
</tr>
<tr>
<td>results</td>
<td>A directory generated by the <code>runme</code> command.</td>
</tr>
</tbody>
</table>

Table 2-5 lists files in the `results` directory generated by the `runme` command.

Table 2-5  Files in the `results` Directory Generated by the `runme` Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Contains the input that the <code>runme</code> command provides to the Java client application.</td>
</tr>
<tr>
<td>output</td>
<td>Contains the output produced when the <code>runme</code> command executes the Java client application.</td>
</tr>
<tr>
<td>expected_output</td>
<td>Contains the output that is expected when the Java client application is executed by the <code>runme</code> command. The data in the <code>output</code> file is compared to the data in the <code>expected_output</code> file to determine whether or not the test passed or failed.</td>
</tr>
<tr>
<td>log</td>
<td>Contains the output generated by the <code>runme</code> command. If the <code>runme</code> command fails, check this file for errors.</td>
</tr>
</tbody>
</table>
### Table 2-5 Files in the results Directory Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setenv.cmd</td>
<td>Contains the commands to set the environment variables needed to build and run the Java Simpapp sample application on the Windows NT operating system platform.</td>
</tr>
<tr>
<td>setenv.ksh</td>
<td>Contains the commands to set the environment variables needed to build and run the Java Simpapp sample application on the UNIX operating system platform.</td>
</tr>
<tr>
<td>stderr</td>
<td>Generated by the tmboot command, which is executed by the runme command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.err.println method sends the output to the stderr file instead of to the ULOG file.</td>
</tr>
<tr>
<td>stdout</td>
<td>Generated by the tmboot command, which is executed by the runme command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.out.println method sends the output to the stdout file instead of to the ULOG file.</td>
</tr>
<tr>
<td>tmsysevt.dat</td>
<td>Contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. This file is generated by the tmboot command in the runme command.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>A binary version of the UBBCONFIG file.</td>
</tr>
<tr>
<td>ubb</td>
<td>The UBBCONFIG file for the Java Simpapp sample application.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command.</td>
</tr>
</tbody>
</table>
Using the Java Simpapp Sample Application

This section describes how to use the Java Simpapp sample application after the runme command is executed.

Run the Java server application in the Java Simpapp sample application, as follows:

Windows NT

prompt>tmboot

UNIX

ksh prompt>tmboot

Run the Java client application in the Java Simpapp sample application, as follows:

Windows NT

prompt>java -classpath .;%TUXDIR%/udataobj\java\jdk\m3envobj.jar
-DTOBJADDR=%TOBJADDR% SimpleClient
String?
Hello World
HELLO WORLD
hello world

UNIX

ksh prompt>java -classpath .:$TUXDIR/udataobj/java/jdk/
/m3envobj.jar -DTOBJADDR=$TOBJADDR SimpleClient
String?
Hello World
HELLO WORLD
hello world

Note: The Java Simpapp sample client application uses the client-only JAR file m3envobj.jar. However, you could also use the m3.jar file to run the client application.
Using the C++ Client Application with the Java Simpapp Sample Application

A C++ client application is provided with the Java Simpapp sample application to demonstrate interoperability between a Java server application and a C++ client application. This section describes the process of building and running the C++ client application.

Build the C++ client application in the Java Simpapp sample application as follows:

1. Copy the files from the following directory to a work directory:

   **Windows NT**
   \WLEdir\samples\CORBA\simpapp_java

   **UNIX**
   /usr/local/WLEdir/samples/corba/simapp_java

   **Note:** The work directory for the Java Simpapp sample application cannot be the same as the work directory for the C++ Simpapp sample application.

2. Change the protection on the files using the following commands:

   **Windows NT**
   prompt>attrib -r drive:\workdirectory\*.*

   **UNIX**
   prompt>/bin/ksh
   ksh prompt>chmod u+w /workdirectory/.*

3. Make sure the UNIX `make` command or the Windows NT `nmake` command is in the path of your machine.

4. Set the M3SIMPDIR environment variable to your work directory.

5. Build the C++ client application, as follows:

   **Windows NT**
   prompt>cd %M3SIMPDIR
prompt>nmake -f makefile.nt simple_client.exe

UNIX
  ksh prompt>cd %M3SIMPDIR
  ksh prompt>make -f makefile.mk simple_client

Run the Java server application in the Java Simpapp sample application, as follows:

Windows NT
  prompt>tmboot

UNIX
  ksh prompt>tmboot

Run the C++ client application in the Java Simpapp sample application, as follows:

Windows NT
  prompt>\%M3SIMPDIR\simple_client
  String? Hello
  HELLO
  hello

UNIX
  ksh prompt>$M3SIMPDIR/simple_client
  String? Hello
  HELLO
  hello

Stopping the Java Simpapp Sample Application

Before using another sample application, enter the following commands to stop the Java Simpapp sample application and to remove unnecessary files from the work directory:
The Java Simpapp Sample Application

Windows NT

prompt>tmshutdown -y
prompt>nmake -f makefile.nt clean

UNIX

ksh prompt>tmshutdown -y
ksh prompt>make -f makefile.mk clean
CHAPTER

3 The JDBC Bankapp Sample Application

This topic includes the following sections:
- How the JDBC Bankapp Sample Application Works
- Development Process for the JDBC Bankapp Sample Application
- Setting Up the Database for the JDBC Bankapp Sample Application
- Building the JDBC Bankapp Sample Application
- Compiling the Client and Server Applications
- Initializing the Database
- Starting the Server Application in the JDBC Bankapp Sample Application
- Files Generated by the JDBC Bankapp Sample Application
- Starting the ATM Client Application in the JDBC Bankapp Sample Application
- Stopping the JDBC Bankapp Sample Application
- Using the ATM Client Application

Refer to the Readme.txt file in the \WLEdir\samples\corba\bankapp_java\JDBC directory for troubleshooting information and for the latest information about using the JDBC Bankapp sample application.
The JDBC Bankapp sample application implements an automatic teller machine (ATM) interface and uses Java Database Connectivity (JDBC) to access a database that stores account and customer information. This topic includes the following sections:

- Java Server Objects
- Application Workflow
- JDBC Connection Pooling

### Java Server Objects

The JDBC Bankapp sample application consists of a Java server application that contains the objects listed in Table 3-1.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TellerFactory</td>
<td>The TellerFactory object creates the object references to the Teller object.</td>
</tr>
<tr>
<td>Teller</td>
<td>The Teller object receives and processes requests for banking operations from the ATM client application.</td>
</tr>
<tr>
<td>DBAccesss</td>
<td>The DBAccess object receives and processes requests from the Teller object to the database.</td>
</tr>
</tbody>
</table>

### Application Workflow

Figure 3-1 illustrates how the JDBC Bankapp sample application works.
How the JDBC Bankapp Sample Application Works

Figure 3-1  The JDBC Bankapp Sample Application

JDBC Connection Pooling

The JDBC Bankapp sample application demonstrates how to use JDBC database connection pooling running in a multithreaded server application. In the JDBC Bankapp sample application, WLE creates and initializes a pool of database connections that the sample application uses. All DBAccess objects share this pool. For more information about JDBC connection pools, see Using JDBC Connection Pooling.

A minimum number of database connections is established when the server is initialized. The number of connections is increased on demand. When a worker thread receives a request for a DBAccess object, the corresponding DBAccess method gets an available database connection from the pool. When the call to the DBAccess method completes, the database connection is returned to the pool. If there is no database connection available and the maximum number of database connections has been established, the worker thread waits until a database connection becomes available.
Development Process for the JDBC Bankapp Sample Application

This topic includes the following sections:

- Object Management Group (OMG) Interface Definition Language (IDL)
- Client Application
- Server Application
- Server Description File (BankApp.xml)
- UBBCONFIG File

This topic describes the development process for the JDBC Bankapp sample application.

Note: The steps in this topic have been done for you and are included in the JDBC Bankapp sample application.

Object Management Group (OMG) Interface Definition Language (IDL)

Table 3-2 lists the CORBA interfaces defined in the OMG IDL for the JDBC Bankapp sample application:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>TellerFactory</td>
<td>Creates object references to the Teller object</td>
<td>create_Teller()</td>
</tr>
</tbody>
</table>

Table 3-2  CORBA Interfaces Defined in the JDBC Bankapp OMG IDL
Development Process for the JDBC Bankapp Sample Application

Table 3-2  CORBA Interfaces Defined in the JDBC Bankapp OMG IDL  (Continued)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Methods</th>
</tr>
</thead>
</table>
| Teller     | Performs banking operations   | verify_pin_number()  
|            |                               | deposit()  
|            |                               | withdraw()  
|            |                               | inquiry()  
|            |                               | transfer()  
|            |                               | report()  |
| DBAccess   | Accesses the Oracle database  | get_valid_accounts()  
|            | on behalf of the Teller object| read_account()  
|            |                               | update_account()  
|            |                               | transfer_funds()  |

BankApp.idl File

Listing 3-1 shows the BankApp.idl file that defines the TellerFactory and Teller interfaces in the JDBC Bankapp sample application. A copy of this file is included in the directory for the JDBC Bankapp sample application.

Listing 3-1  OMG IDL Code for the TellerFactory and Teller Interfaces

```idl
#pragma prefix "beasys.com"
#pragma javaPackage "com.beasys.samples"

#include "Bank.idl"

module BankApp{
    exception IOException {};
    exception TellerInsufficientFunds {};

    struct BalanceAmounts{
        float fromAccount;
        float toAccount;
    };

    struct TellerActivity {
        long totalRequests;
        long totalSuccesses;
        long totalFailures;
        float currentBalance;
    };
}
```
// Process Object
interface Teller {
    void verify_pin_number(in short pinNo,
        out Bank::CustAccounts accounts)
        raises(Bank::PinNumberNotFound, IOException);
    float deposit(in long accountNo, in float amount)
        raises(Bank::AccountRecordNotFound, IOException);
    float withdraw(in long accountNo, in float amount)
        raises(Bank::AccountRecordNotFound,
            Bank::InsufficientFunds, IOException, TellerInsufficientFunds);
    float inquiry(in long accountNo)
        raises(Bank::AccountRecordNotFound, IOException);
    void transfer(in long fromAccountNo,
        in long toAccountNo, in float amount,
        out BalanceAmounts balAmounts)
        raises(Bank::AccountRecordNotFound,
            Bank::InsufficientFunds, IOException);
    void report(out TellerActivity tellerData)
        raises(IOException);
};

interface TellerFactory{
    Teller createTeller(in string tellerName);
};

BankDB.idl File

Listing 3-2 shows the BankDB.idl file that defines the DBAccess interface in the JDBC Bankapp sample application. A copy of this file is included in the directory for the JDBC Bankapp sample application.

Listing 3-2 OMG IDL Code for the DBAccess Interface

#pragma prefix "beasys.com"
#pragma javaPackage "com.beasys.samples"
#include "Bank.idl"

module BankDB{
    struct AccountData{
        long accountID;
    }
Development Process for the JDBC Bankapp Sample Application

Listing 3-3 shows the Bank.idl file that defines common exceptions and structures. It is included by both BankApp.idl and BankDB.idl. A copy of this file is included in the directory for the JDBC Bankapp sample application.

Listing 3-3   OMG IDL Code for the Exceptions and Structures in JDBC Bankapp

#pragma prefix "beasys.com"
#pragma javaPackage "com.beasys.samples"

module Bank{

    exception DataBaseException ();
    exception PinNumberNotFoundException ();
    exception AccountRecordNotFoundException ();
    exception InsufficientFunds ();

    struct CustAccounts{
        long checkingAccountID;
    }

    interface DBAccess{
        void get_valid_accounts(in short, pinNo,
            out Bank::CustAccounts accounts)
            raises(Bank::DatabaseException,
                Bank::PinNumberNotFoundException);
        void read_account(inout AccountData data)
            raises(Bank::DatabaseException,
                Bank::AccountRecordNotFoundException);
        void update_account(inout AccountData data)
            raises(Bank::DatabaseException,
                Bank::AccountRecordNotFoundException,
                Bank::InsufficientFunds);
        void transfer_funds(in float_amount,
            inout AccountData fromAcct,
            inout AccountData toAcct,
            raises(Bank::DatabaseException,
                Bank::AccountRecordNotFoundException,
                Bank::InsufficientFunds);
    }

    interface DBAccess{

    }

Bank.idl File

float balance;

};
Client Application

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

- Initializes the ORB.
- Uses the Bootstrap environmental object to establish communication with the WebLogic Enterprise (WLE) domain.
- Resolves initial references to the FactoryFinder environmental object.
- Uses a factory to get an object reference for the Teller object.
- Invokes the verify_pin_number, deposit, withdraw, inquiry, transfer, and report methods on the Teller object.

A Java client application, referred to as the ATM client application, is included in the JDBC Bankapp sample application. For more information about writing Java client applications that use transactions, see Using Transactions.

Server Application

During the development of the server application, you would write the following:

- The Server object, which initializes the server application in the JDBC Bankapp sample application and registers a factory for the Teller object with the WLE domain. The Server object also obtains a reference to the JDBC connection pool from JNDI.
- The implementations for the methods of the Teller and DBAccess objects.
  The implementations for the Teller object include invoking operations on the DBAccess object.
Because the Teller object has durable state (for example, ATM statistics) that is stored in an external source (a flat file), the method implementations must also include the activate_object and deactivate_object methods to ensure the Teller object is initialized with its state.

The JDBC Bankapp server application is configured to be multithreaded. Writing a multithreaded WLE Java server application is the same as writing a single-threaded Java server application; you cannot establish multiple threads programmatically in your object implementations. Instead, you establish the number of threads for a Java server application in the UBBCONFIG file. For information about writing Java server applications and using threads in Java server applications, see Using Transactions.

**Server Description File (BankApp.xml)**

During development, you create a Server Description File (BankApp.xml) that defines the activation and transaction policies for the TellerFactory, Teller, and DBAccess interfaces. Table 3-3 shows the activation and transaction policies for the JDBC Bankapp sample application.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Activation Policy</th>
<th>Transaction Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TellerFactory</td>
<td>Process</td>
<td>Never</td>
</tr>
<tr>
<td>Teller</td>
<td>Method</td>
<td>Never</td>
</tr>
<tr>
<td>DBAccess</td>
<td>Method</td>
<td>Never</td>
</tr>
</tbody>
</table>

A Server Description File for the JDBC Bankapp sample application is provided. For information about creating Server Description Files and defining activation and transaction policies on objects, see Creating Java Server Applications.

**UBBCONFIG File**

When using the WLE software, the server application is represented by a Java Archive (JAR). The JAR must be loaded into the Java Virtual Machine (JVM) to be executed. The JVM must execute in a WLE server application to be integrated in an WLE.
Enabling Multithreaded Support

If your Java server application is multithreaded, you can establish the number of threads by using the command-line option (CLOPT) -M in the SERVERS section of the UBBCONFIG file. In Listing 3-4, the -M 100 option enables multithreading for the JavaServer and specifies 100 as the maximum number of worker threads that a particular instance of JavaServer can support. The largest number that you can specify is 500.

Listing 3-4   Enabling Multithreaded Support in UBBCONFIG

JavaServer
SRVGRP = BANK_GROUP1
SRVID = 2
SRVTYP = JAVA
CLOPT = "-A -- -M 100 Bankapp.jar TellerFactory_1 bank_pool"
RESTART = N

Notes: The SRVTYP=JAVA line is required when using JDBC connection pooling.

The information for the CLOPT parameter needs to be entered on one line.

You also need to set the MAXACCESSERS parameter in the RESOURCES section of the UBBCONFIG file to account for the number of worker threads that each server application is configured to run. The MAXACCESSERS parameter specifies the number of processes that can attach to a WLE application.

Setting Up the Connection Pool

For the JDBC Bankapp sample application, you need to include the name of the connection pool on the command-line option (CLOPT) in the SERVERS section of the UBBCONFIG file, as shown in Listing 3-5.

3-10   Guide to the Java Sample Applications
Listing 3-5  Specifying the Connection Pool Name (bank_pool) in UBBCONFIG

CLOPT = "-A -- -M 100 Bankapp.jar TellerFactory_1 bank_pool"

Note: The information for the CLOPT parameter needs to be entered on one line.

In addition, you need to include the following information on the JDBCCONNPOOLS section of the UBBCONFIG file:

- The server group and server ID of the server.
- The class name of JDBC driver:
  - JdbcOracle734 for the jdbcKona/Oracle driver
  - JdbcMSSQL4 for the jdbcKona/MSSQLServer driver
- Either the JDBC URL for the Oracle database, or the name of the machine where the Microsoft SQL Server database is installed
- Optionally, either the user id and password for the Oracle database, or the user name and password you defined for the master instance of the Microsoft SQL Server database
- The initial number of database connections in the pool
- The maximum number of database connections in the pool

Listing 3-6 provides an example of the JDBCCONNPOOLS section in the UBBCONFIG.

Listing 3-6  Specifying JDBCCONNPOOLS Information in UBBCONFIG

```
JDBCCONNPOOLS
bank_pool
  SRVGRP       = BANK_GROUP1
  SRVID        = 2
  DRIVER       = "weblogic.jdbc20.oci815.Driver"
  URL          = "jdbc:weblogic:oracle:Beq-local"
  PROPS        = "user=scott;password=tiger;server=Beq-Local"
  ENABLEXA     = N
  INITCAPACITY = 2
  MAXCAPACITY  = 10
  CAPACITYINCR = 1
  CREATEONSTARTUP = Y
```
Setting Up the Database for the JDBC Bankapp Sample Application

The JDBC Bankapp sample application uses a database to store all the bank data. You can use either the Oracle or the Microsoft SQL Server database with the JDBC Bankapp sample application.

Before you can build and run the JDBC Bankapp sample application, you need to follow the steps in the product documentation to install the desired database.

The jdbcKona/Oracle and jdbcKona/MSSQLServer4 drivers are installed as part of the WLE installation. For more information about the jdbcKona drivers, refer to the JDBC Driver Programming Reference and the BEA WebLogic Installation Guide.

Note: The jdbcKona/Oracle driver supports Oracle Version 7.3.4 and Oracle8i (for Solaris and Windows NT) and versions 8.04 and 8i (for HP-UX). By default, this sample application supports Oracle Version 7.3.4 on NT/Solaris and Version 8.0.4 on HP. You can use a different Oracle version by specifying command line parameters, as described in “Step 4: Run the setupJ Command” on page 3-20.

Setting Up an Oracle Database

If you are using Oracle as the database for the JDBC Bankapp sample application, you need to install the following software:

- Visual C++ Version 5.0 with Service Pack for Visual Studio (Windows NT only)
- Sun SparcWorks Compiler 4.2 (Solaris only)
- Oracle Version 7.3.4
When using the Oracle database, you use the default database created by the Oracle installation program. You need the connection string you defined for the Oracle database and the default user id and password. Refer to the Oracle product documentation for details about obtaining this information.

Setting Up a Microsoft SQL Server Database

If you are using the Microsoft SQL Server as the database for the JDBC Bankapp sample application, you need to install the following software:

- Visual C++ Version 5.0 with Service Pack for Visual Studio (Windows NT only)
- Sun SparcWorks Compiler 4.2 (Solaris only)
- Microsoft SQL Server Version 7.0

When using the Microsoft SQL Server database, you use the master database instance. You need the name of the machine where the Microsoft SQL Server database is installed and the user name and password you defined for the master instance of the Microsoft SQL Server database. Refer to the Microsoft product documentation for details about obtaining this information.

Building the JDBC Bankapp Sample Application

This topic describes the following steps, which are required to build the JDBC Bankapp sample application:

- Step 1: Copy the Files for the JDBC Bankapp Sample Application into a Work Directory.
- Step 2: Change the Protection Attribute on the Files for the JDBC Bankapp Sample Application.
- Step 3: Verify the Settings of the Environment Variables.
Step 1: Copy the Files for the JDBC Bankapp Sample Application into a Work Directory

You need to copy the files for the JDBC Bankapp sample application into a work directory on your local machine.

Source File Directories

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

**Windows NT**
- drive:\WLEdir\samples\corba\bankapp_java\JDBC
- drive:\WLEdir\samples\corba\bankapp_java\client
- drive:\WLEdir\samples\corba\bankapp_java\shared

**UNIX**
- /usr/local/WLEdir/samples/corba/bankapp_java/JDBC
- /usr/local/WLEdir/samples/corba/bankapp_java/client
- /usr/local/WLEdir/samples/corba/bankapp_java/shared

Table 3-4 describes the contents of these directories.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC</td>
<td>Source files and commands needed to build and run the JDBC Bankapp sample application.</td>
</tr>
<tr>
<td>client</td>
<td>Files for the ATM client application. The images subdirectory contains .gif files used by the graphical user interface in the ATM client application.</td>
</tr>
</tbody>
</table>
Copying Source Files to the Work Directory

You need to manually copy only the files in the \JDBC directory. The other sample application files are automatically copied from the \client and \shared directories when you execute the setupJ command. For example:

Windows NT

prompt> cd c:\mysamples\bankapp_java\JDBC
prompt> copy c:\WLEdir\samples\corba\bankapp_java\JDBC\* 

UNIX

ksh prompt> cd /usr/mysamples/bankapp_java/JDBC/*
ksh prompt> cp $TUXDIR/samples/bankapp_java/JDBC/* .

Note: You cannot run the JDBC Bankapp sample application in the same work directory as the XA Bankapp sample application, because some of the files for the JDBC Bankapp sample application have the same name as files for the XA Bankapp sample application.

Source Files Used to Build the JDBC Bankapp Sample Application

Table 3-5 lists the files used to build and run the JDBC Bankapp sample application.

Table 3-5  Files Included in the JDBC Bankapp Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank.idl</td>
<td>The OMG IDL code that declares common structures and extensions for the JDBC Bankapp sample application.</td>
</tr>
<tr>
<td>BankApp.idl</td>
<td>The OMG IDL code that declares the TellerFactory and Teller interfaces.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BankDB.idl</td>
<td>The OMG IDL code that declares the DBAccess interface.</td>
</tr>
<tr>
<td>TellerFactoryImpl.java</td>
<td>The Java source code that implements the createTeller method. This file is in the com.beasys.samples package. It is automatically moved to the com/beasys/samples directory by the setupJ command.</td>
</tr>
<tr>
<td>TellerImpl.java</td>
<td>The Java source code that implements the verify, deposit, withdraw, inquiry, transfer, and report methods. This file is in the com.beasys.samples package. It is automatically moved to the com/beasys/samples directory by the setupJ command.</td>
</tr>
<tr>
<td>BankAppServerImpl.java</td>
<td>The Java source code that overrides the Server.initialize and Server.release methods.</td>
</tr>
<tr>
<td>DBAccessImpl.java</td>
<td>The Java source code that implements the get_valid_accounts, read_account, update_account, and transfer methods. This file is in the com.beasys.samples package. It is automatically moved to the com/beasys/samples directory by the setupJ command.</td>
</tr>
<tr>
<td>Atm.java</td>
<td>The Java source code for the ATM client application.</td>
</tr>
<tr>
<td>BankStats.java</td>
<td>Contains methods to initialize, read from, and write to the flat file that contains the ATM statistics.</td>
</tr>
<tr>
<td>BankApp.xml</td>
<td>The Server Description File used to associate activation and transaction policy values with CORBA interfaces.</td>
</tr>
<tr>
<td>InitDB.java</td>
<td>A Java program that initializes the database and ensures that JDBC is working properly.</td>
</tr>
</tbody>
</table>
Step 2: Change the Protection Attribute on the Files for the JDBC Bankapp Sample Application

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

**Windows NT**

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

**UNIX**

```
prompt>/bin/ksh
ksh prompt> chmod u+w /workdirectory/*.*
```
Step 3: Verify the Settings of the Environment Variables

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

Environment Variables

Table 3-6 lists the environment variables required to run the JDBC Bankapp sample application.

Table 3-6 Required Environment Variables for the JDBC Bankapp Sample Application

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUXDIR</td>
<td>The directory path where you installed the WLE software. For example:</td>
</tr>
</tbody>
</table>
|                      |  **Windows NT**  
|                      | TUXDIR=c:\WLEdir  
|                      |  **UNIX**  
|                      | TUXDIR=/usr/local/WLEdir  
| JAVA_HOME             | The directory path where you installed the JDK software. For example: |
|                      |  **Windows NT**  
|                      | JAVA_HOME=c:\JDK1.2  
|                      |  **UNIX**  
|                      | JAVA_HOME=/usr/local/JDK1.2  
| ORACLE_HOME           | The directory path where you installed the Oracle software. For example: |
|                      |  **Windows NT**  
|                      | ORACLE_HOME=d:\orant  
|                      |  **UNIX**  
|                      | ORACLE_HOME=/usr/local/oracle  
|                      | **Note:** This environment variable applies only if you are using the Oracle database on the Solaris operating system.
Verifying Settings

To verify that the information defined during installation is correct:

**Windows NT**
1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.
3. Click the System icon.
   The System Properties window appears.
4. Click the Environment tab.
   The Environment page appears.
5. Check the settings for TUXDIR and JAVA_HOME.

**UNIX**

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

Changing Settings

To change the settings:

**Windows NT**
1. On the Environment page in the System Properties window, click the environment variable you want to change, or enter the name of the environment variable in the Variable field.
2. In the Value field, enter the correct information for the environment variable.
3. Click OK to save the changes.

**UNIX**

```
ksh prompt>TUXDIR=directorypath; export TUXDIR
ksh prompt>JAVA_HOME=directorypath; export JAVA_HOME
```
The JDBC Bankapp Sample Application

ksh prompt>JAVA_HOME=directorypath; export ORACLE_HOME

**Note:** If you are running multiple WLE applications concurrently on the same machine, you also need to set the IPKEY and PORT environment variables. See the Readme.txt file for information about how to set these environment variables.

### Step 4: Run the setupJ Command

The `setupJ` command automates the following steps:

1. Copy the required files from the \client and \shared directories.
2. Set the PATH, TOBJADDR, APPDIR, TUXCONFIG, and CLASSPATH system environment variables.
3. Create the UBBCONFIG file (ubb_jdbc).
4. Create a `setenvJ.cmd` or `setenvJ.ksh` file that can be used to reset the system environment variables.

### Syntax

The syntax for the `setupJ` command is:

```
prompt>setupJ DB_DRIVER DB_SERVER DB_USER DB_PASSWORD
```

where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB_DRIVER</strong></td>
<td>Name of the database driver. Valid values include:</td>
</tr>
<tr>
<td></td>
<td>- jdbcOracle734</td>
</tr>
<tr>
<td></td>
<td>- jdbcOracle804</td>
</tr>
<tr>
<td></td>
<td>- jdbcOracle815</td>
</tr>
<tr>
<td></td>
<td>- jdbcMSSQL4</td>
</tr>
<tr>
<td></td>
<td>Default values are jdbcOracle734 (on Solaris or NT) or jdbcOracle804 (on Hewlett-Packard).</td>
</tr>
</tbody>
</table>
Note: SetupJ uses default values unless you explicitly specify arguments. For example, to use Microsoft SQL Server, you must specify all command line parameters.

Command

Follow these steps to enter the setupJ command:

Windows NT

prompt> cd c:\mysamples\bankapp_java\JDBC
prompt> setupJ jdbcOracle815 Beq-Local scott tiger

UNIX

prompt> /bin/ksh
prompt> cd /usr/mysamples/bankapp_java/JDBC
prompt> . ./setupJ.ksh jdbcOracle815 null scott tiger

Step 5: Load the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

prompt> tmloadcf -y ubb_jdbc
Compiling the Client and Server Applications

The directory for the JDBC Bankapp sample application contains a make file that builds the client and server sample applications. During development, you use the buildjavaserver command to build the server application, and your Java product’s development commands to build the client application. However, for the JDBC Bankapp sample application, these steps are included in the make file.

Use the following commands to build the client and server applications in the JDBC Bankapp sample application:

**Windows NT**

`prompt>nmake -f makefileJ.nt`

**UNIX**

`prompt>make -f makefileJ.mk`

Initializing the Database

This topic includes the following sections:

- Initializing an Oracle Database
- Initializing a Microsoft SQL Server Database

Initializing an Oracle Database

To initialize an Oracle database using the default arguments, enter the following command:

`prompt>java InitDB`
To initialize the Oracle database with user-defined attributes, enter the following command:

```
prompt>java InitDB driver_name connect_string username password
```

where

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver_name</td>
<td>Name of the database driver. Valid values include:</td>
</tr>
<tr>
<td></td>
<td>jdbcOracle734</td>
</tr>
<tr>
<td></td>
<td>jdbcOracle804</td>
</tr>
<tr>
<td></td>
<td>jdbcOracle815</td>
</tr>
<tr>
<td></td>
<td>jdbcMSSQL4</td>
</tr>
<tr>
<td></td>
<td>Default values are jdbcOracle734 (on Solaris or NT) or jdbcOracle804 (on Hewlett-Packard).</td>
</tr>
<tr>
<td>connect_string</td>
<td>Connection string for the instance of the Oracle database being used with the JDBC Bankapp sample application. Default values are Beq-Local (on NT) or null (on Solaris or Hewlett-Packard).</td>
</tr>
<tr>
<td>username</td>
<td>User name for the Oracle database. Default value is scott.</td>
</tr>
<tr>
<td>password</td>
<td>Password for the Oracle database. Default value is tiger.</td>
</tr>
</tbody>
</table>

### Initializing a Microsoft SQL Server Database

To initialize a Microsoft SQL Server database, enter the following command:

```
prompt>java InitDB JdbcMSSQL4 db_server username password
```

where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JdbcMSSQL4</td>
<td>Name of the database driver for Microsoft SQL Server. This is the only valid value.</td>
</tr>
<tr>
<td>db_server</td>
<td>Name of the machine on which the Microsoft SQL Server database is installed.</td>
</tr>
</tbody>
</table>
Starting the Server Application in the JDBC Bankapp Sample Application

Start the server application in the JDBC Bankapp sample application by entering the following command:

```
prompt>tmboot -y
```

The `tmboot` command starts the application processes listed in Table 3-7.

Table 3-7 Application Processes Started by `tmboot` Command

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMSYSEVT</td>
<td>BEA TUXEDO system event broker.</td>
</tr>
<tr>
<td>TMFFNAME</td>
<td>Three TMFFNAME server processes are started:</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process started with the -N and -M options is the master NameManager service. The NameManager service maintains a mapping of the application-supplied names to object references.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process started with the -N option is the slave NameManager service.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process started with the -F option contains the FactoryFinder object.</td>
</tr>
<tr>
<td>JavaServer</td>
<td>Server process that implements the TellerFactory, Teller, and DBAccess interfaces.</td>
</tr>
<tr>
<td>ISL</td>
<td>IIOP Listener process.</td>
</tr>
</tbody>
</table>
Files Generated by the JDBC Bankapp Sample Application

Table 3-8 lists the files generated by the JDBC Bankapp sample application.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubb_jdbc</td>
<td>The UBBCONFIG file for the JDBC Bankapp sample application. This file is generated by the setupJ command.</td>
</tr>
<tr>
<td>setenvJ.cmd and setenvJ.ksh</td>
<td>Contains the commands to set the environment variables needed to build and run the JDBC Bankapp sample application. setenvJ.cmd is the Windows NT version and setenvJ.ksh is the UNIX Korn shell version of the file.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>A binary version of the UBBCONFIG file. Generated by the tmloadcf command.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command. The log file also contains messages generated by the server applications and by the tmshutdown command.</td>
</tr>
<tr>
<td>.adm/.keybd</td>
<td>A file that contains the security encryption key database. The subdirectory is created by the tmloadcf command.</td>
</tr>
<tr>
<td>Atm$1.class Atm.class AtmAppletStub.class AtmArrow.class AtmButton.class AtmCenterTextCanvas.class AtmClock.class AtmScreen.class AtmServices.class AtmStatus.class</td>
<td>Used by the Java client application. Created when the Atm.java file is compiled.</td>
</tr>
<tr>
<td>InitDB.class</td>
<td>Initializes the database used by the JDBC Bankapp sample application. Created when InitDB.java is compiled.</td>
</tr>
</tbody>
</table>
### The JDBC Bankapp Sample Application

#### Table 3-8  Files Generated by the JDBC Bankapp Sample Application  (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountRecordNotFound.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the Bank.idl file. These files are created in the com/beasys/samples/Bank directory.</td>
</tr>
<tr>
<td>AccountRecordNotFoundHelper.java</td>
<td></td>
</tr>
<tr>
<td>AccountRecordNotFoundHolder.java</td>
<td></td>
</tr>
<tr>
<td>CustAccounts.java</td>
<td></td>
</tr>
<tr>
<td>CustAccountsHelper.java</td>
<td></td>
</tr>
<tr>
<td>CustAccountsHolder.java</td>
<td></td>
</tr>
<tr>
<td>DataBaseException.java</td>
<td></td>
</tr>
<tr>
<td>DataBaseExceptionHelper.java</td>
<td></td>
</tr>
<tr>
<td>DataBaseExceptionHolder.java</td>
<td></td>
</tr>
<tr>
<td>InsufficientFunds.java</td>
<td></td>
</tr>
<tr>
<td>InsufficientFundsHelper.java</td>
<td></td>
</tr>
<tr>
<td>InsufficientFundsHolder.java</td>
<td></td>
</tr>
<tr>
<td>PinNumberNotFound.java</td>
<td></td>
</tr>
<tr>
<td>PinNumberNotFoundHelper.java</td>
<td></td>
</tr>
<tr>
<td>PinNumberNotFoundHolder.java</td>
<td></td>
</tr>
<tr>
<td>BalanceAmounts.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the BankApp.idl file. These files are created in the com/beasys/samples/BankApp subdirectory.</td>
</tr>
<tr>
<td>BalanceAmountsHelper.java</td>
<td></td>
</tr>
<tr>
<td>BalanceAmountsHolder.java</td>
<td></td>
</tr>
<tr>
<td>IOException.java</td>
<td></td>
</tr>
<tr>
<td>IOExceptionHelper.java</td>
<td></td>
</tr>
<tr>
<td>IOExceptionHolder.java</td>
<td></td>
</tr>
<tr>
<td>Teller.java</td>
<td></td>
</tr>
<tr>
<td>TellerActivity.java</td>
<td></td>
</tr>
<tr>
<td>TellerActivityHelper.java</td>
<td></td>
</tr>
<tr>
<td>TellerActivityHolder.java</td>
<td></td>
</tr>
<tr>
<td>TellerFactory.java</td>
<td></td>
</tr>
<tr>
<td>TellerFactoryHelper.java</td>
<td></td>
</tr>
<tr>
<td>TellerFactoryHolder.java</td>
<td></td>
</tr>
<tr>
<td>TellerInsufficientFunds.java</td>
<td></td>
</tr>
<tr>
<td>TellerInsufficientFundsHelper.java</td>
<td></td>
</tr>
<tr>
<td>TellerInsufficientFundsHolder.java</td>
<td></td>
</tr>
<tr>
<td>_TellerFactoryImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_TellerFactoryStub.java</td>
<td></td>
</tr>
<tr>
<td>_TellerImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_TellerStub.java</td>
<td></td>
</tr>
</tbody>
</table>
### Starting the ATM Client Application in the JDBC Bankapp Sample Application

Start the ATM client application by entering the following command:

**Note:** The following command sets the Java CLASSPATH to include the current directory and the client JAR file (`m3envobj.jar`). The full WLE JAR file (`m3.jar`) can be specified instead of the client JAR file.

---

#### Table 3-8 Files Generated by the JDBC Bankapp Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountData.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the BankDB.idl file. These files are created in the com/beasys/samples/BankDB subdirectory.</td>
</tr>
<tr>
<td>AccountDataHelper.java</td>
<td></td>
</tr>
<tr>
<td>AccountDataHolder.java</td>
<td></td>
</tr>
<tr>
<td>DBAccessHelper.java</td>
<td></td>
</tr>
<tr>
<td>DBAccessHolder.java</td>
<td></td>
</tr>
<tr>
<td>_DBAccessImplBase.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the BankDB.idl file. These files are created in the com/beasys/samples/BankDB subdirectory.</td>
</tr>
<tr>
<td>_DBAccessStub.java</td>
<td></td>
</tr>
<tr>
<td>Bankapp.ser</td>
<td>The Server Descriptor File and Server Java Archive file generated by the buildjavaserver command in the make file.</td>
</tr>
<tr>
<td>Bankapp.jar</td>
<td></td>
</tr>
<tr>
<td>stderr</td>
<td>Generated by the tmboot command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.err.println method sends the output to the stderr file instead of the ULOG file.</td>
</tr>
<tr>
<td>stdout</td>
<td>Generated by the tmboot command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.out.println method sends the output to the stdout file instead of the ULOG file.</td>
</tr>
<tr>
<td>tmsysevt.dat</td>
<td>Contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. This file is generated by the tmboot command.</td>
</tr>
</tbody>
</table>

---

Starting the ATM Client Application in the JDBC Bankapp Sample Application

Start the ATM client application by entering the following command:

**Note:** The following command sets the Java CLASSPATH to include the current directory and the client JAR file (`m3envobj.jar`). The full WLE JAR file (`m3.jar`) can be specified instead of the client JAR file.
Windows NT

prompt>java -classpath .;%TUXDIR%;udataobj\java\jdk\m3envobj.jar -DTOBJADDR=%TOBJADDR% Atm Teller1

UNIX

ksh prompt>java -classpath .:$TUXDIR/udataobj/java/jdk/m3envobj.jar -DTOBJADDR=$TOBJADDR Atm Teller1

The GUI for the ATM client application appears. Figure 3-2 shows the GUI for the ATM client application.

Figure 3-2   GUI for ATM Client Application
Stopping the JDBC Bankapp Sample Application

Before using another sample application, enter the following commands to stop the JDBC Bankapp sample application and to remove unnecessary files from the work directory:

**Windows NT**

prompt>tmshutdown -y
prompt>nmake -f makefileJ.nt clean

**UNIX**

ksh prompt>tmshutdown -y
ksh prompt>make -f makefileJ.mk clean

Using the ATM Client Application

This topic includes the following sections:

- Available Banking Operations
- Available Statistics
- Keypad Functions
- Steps for Using the ATM Client Application

Available Banking Operations

In the ATM client application, a customer enters a personal identification number (PIN) and performs one of the following banking operations:
The JDBC Bankapp Sample Application

- Withdraws money from the account
- Deposits money in the account
- Inquires about the balance of the account
- Transfers money between checking and savings accounts

Available Statistics

One special PIN number (999) allows customers to receive statistics about the ATM machine. The following statistics are available:

- Total number of requests received by the ATM machine
  For example, an inquiry is one request, and a withdrawal is one request.
- Total number of successful requests
- Total number of failed requests
  For example, when a customer attempts to withdraw more money than is in his account, the request fails.
- Total amount of cash remaining in the ATM machine
  The ATM machine starts with $10,000 and the amount decreases with each withdrawal request.

Keypad Functions

Use the keypad in the ATM client application to enter a PIN and amounts for deposit, transfer, and withdrawal. Table 3-9 describes the functions available on the keypad in the ATM client application.

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>Use this key to cancel the current operation and exit the view.</td>
</tr>
</tbody>
</table>

Guide to the Java Sample Applications
Steps for Using the ATM Client Application

To use the ATM client application in the JDBC Bankapp sample application:

1. Enter one of the following PINs: 100, 110, 120, or 130.

2. Click OK.
   The Operations view appears. Figure 3-3 shows the Operations view in the ATM client application.

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Use this key to accept the entered data. After you enter a PIN or an amount for deposit, transfer, or withdrawal, you need to click the OK button to have the action take effect.</td>
</tr>
<tr>
<td>Numerics (0 through 9)</td>
<td>Use these keys to enter your PIN and an amount for deposit, transfer, and withdrawal amounts.</td>
</tr>
<tr>
<td>Period (.)</td>
<td>Use this key to enter decimal amounts for deposit, transfer, and withdrawal.</td>
</tr>
</tbody>
</table>
From the Operations view, you can perform the following banking operations:

- Inquiry
- Transfer
- Deposit
- Withdrawal

3. Click the desired banking operation.
4. Click either the Checking Acct or Savings Acct button.
5. Enter a dollar amount.
6. Click OK.
   An updated account balance appears.

**Note:** After you click OK, you cannot cancel the operation. If you enter an amount and then select Cancel, the ATM client application cancels your operation and displays the previous screen.

7. Click OK.
8. Click Cancel to return to the main window of the ATM client application.
CHAPTER

4 The XA Bankapp Sample Application

This topic includes the following sections:

- How the XA Bankapp Sample Application Works
- Development Process for the XA Bankapp Sample Application
- Setting Up the Database for the XA Bankapp Sample Application
- Building the XA Bankapp Sample Application
- Compiling the Client and Server Applications
- Initializing the Oracle Database
- Starting the Server Application in the XA Bankapp Sample Application
- Files Generated by the XA Bankapp Sample Application
- Starting the ATM Client Application in the XA Bankapp Sample Application
- Stopping the XA Bankapp Sample Application
- Using the ATM Client Application

For troubleshooting information and the most recent information about using the XA Bankapp sample application, see the Readme.txt file in the \WLEdir\samples\corba\bankapp_java\XA directory.
How the XA Bankapp Sample Application Works

The XA Bankapp sample application is a CORBA application that implements the same automatic teller machine (ATM) interface as the JDBC Bankapp sample application. However, the XA Bankapp sample application uses the Oracle XA library and the WebLogic Enterprise (WLE) Transaction Manager to coordinate transactions between the WLE application and the Oracle database that stores account and customer information.

This topic includes the following sections:
- Server Applications
- Application Workflow

Server Applications

The XA Bankapp sample application consists of two server applications:
- A Java server application, which implements the TellerFactory and Teller objects.
- A C++ server application, which processes requests on objects that implement the DBAccesss interface.

Application Workflow

Figure 4-1 illustrates how the XA Bankapp sample application works.
In the XA Bankapp sample application, transactions are started and stopped in the Teller object using the Java Transaction Service (JTS) API. In the JDBC Bankapp sample application, transactions are started and stopped in the DBAccess object using the Java Database Connectivity (JDBC) API.

In the XA Bankapp sample application, the DBAccess object is implemented in C++ instead of Java and resides in its own server application. The object reference for the DBAccess object is generated in its Server::initialize method and is registered with the FactoryFinder environmental object.

Software Prerequisites

To run the XA Bankapp sample application, you need to install the following software:

- Visual C++ Version 5.0 with Service Pack 3 for Visual Studio
- Oracle Version 7.3.4
Development Process for the XA Bankapp Sample Application

This topic includes the following sections:

- Object Management Group (OMG) Interface Definition Language (IDL)
- Client Application
- Server Application
- Server Description File
- Implementation Configuration File
- UBBCONFIG File

These sections describe the development process for the XA Bankapp sample application.

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

Object Management Group (OMG) Interface Definition Language (IDL)

The `BankApp.idl` file used in the XA Bankapp sample application defines the `TellerFactory` and `Teller` interfaces and the `Bank.idl` file defines exceptions and structures. The `transfer_funds` interface has been removed from the `BankDB.idl` because transactions are now started and stopped by the `Teller` object.

Client Application

The XA Bankapp sample application uses the same client application as the JDBC Bankapp sample application.
Server Application

For the XA Bankapp sample application, you would write the following:

- The Java Server object, which initializes the Java server application in the XA Bankapp sample application and registers a factory for the Teller object with the WLE domain.

Server Description File

During development, you create a Server Description File (BankApp.xml) that defines the activation and transaction policies for the TellerFactory and Teller objects. Table 4-1 shows the activation and transaction policies for the XA Bankapp sample application.

Table 4-1  Activation and Transaction Policies for XA Bankapp Sample Application

<table>
<thead>
<tr>
<th>Interface</th>
<th>Activation Policy</th>
<th>Transaction Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TellerFactory</td>
<td>Process</td>
<td>Never</td>
</tr>
<tr>
<td>Teller</td>
<td>Method</td>
<td>Never</td>
</tr>
</tbody>
</table>

A Server Description File for the XA Bankapp sample application is provided. For information about creating Server Description Files and defining activation and transaction policies on objects, see Creating Java Server Applications.

Implementation Configuration File

When writing WLE C++ server applications, you create an Implementation Configuration File (ICF), which is similar to the Server Description File. This file has been created for you and defines an activation policy of transaction and a transaction policy of always for the DBAccess interface.

For information about creating ICF files and defining activation and transaction policies on objects, see Creating C++ Server Applications.
UBBCONFIG File

During development, you need to include the following information in the UBBCONFIG file:

- 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 Distributed Systems manual.

- 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 Using Transactions.

Setting Up the Database for the XA Bankapp Sample Application

The XA Bankapp sample application uses an Oracle database to store all the bank data. Before using the XA Bankapp sample application, you need to install the following Oracle components:

- Oracle Server, Version 7.3.4

- Pro*C/C++ release 7.3.4 (for more information about supported compilers, see the Oracle product documentation)

Note: When installing the specified Oracle components, other Oracle components are also installed. However, you will not use these additional components with the XA Bankapp sample application.

You also need to start the Oracle database daemon and enable an XA resource manager.

For information about installing the Oracle database and performing the necessary setup tasks, see the product documentation for the Oracle database.
Building the XA Bankapp Sample Application

This topic includes the following sections:

- Step 1: Copy the Files for the XA Bankapp Sample Application into a Work Directory
- Step 2: Change the Protection Attribute on the Files for the XA Bankapp Sample Application
- Step 3: Verify the Settings of the Environment Variables
- Step 4: Run the setupX Command
- Step 5: Load the UBBCONFIG File
- Step 6: Create a Transaction Log

These sections describe how to build the XA Bankapp sample application.

Step 1: Copy the Files for the XA Bankapp Sample Application into a Work Directory

You need to copy the files for the XA Bankapp sample application into a work directory on your local machine.

Source File Directories

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

Windows NT

drive:\WLEdir\samples\corba\bankapp_java\XA
drive:\WLEdir\samples\corba\bankapp_java\client
The XA Bankapp Sample Application

drive:\WLEdir\samples\corba\bankapp_java\shared

UNIX

/usr/local/WLEdir/samples/corba/bankapp_java/XA
/usr/local/WLEdir/samples/corba/bankapp_java/client
/usr/local/WLEdir/samples/corba/bankapp_java/shared

Table 4-2 describes the contents of these directories:

Table 4-2  Source File Directories in the XA Bankapp Sample Application

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XA</td>
<td>Source files and commands needed to build and run the XA Bankapp sample application.</td>
</tr>
<tr>
<td>client</td>
<td>Files for the ATM client application. The images subdirectory contains .gif files used by the graphical user interface in the ATM client application.</td>
</tr>
<tr>
<td>shared</td>
<td>Common files for the JDBC Bankapp and XA Bankapp sample applications.</td>
</tr>
</tbody>
</table>

Copying Source Files to the Work Directory

You need only to copy the files manually in the XA directory. The other files are automatically copied from the client and shared directories when you execute the setupX command. For example:

Windows NT

prompt> cd c:\mysamples\bankapp_xa\XA
prompt> copy c:\WLEdir\samples\corba\bankapp_xa\XA/*

UNIX

ksh prompt> cd /usr/mysamples/bankapp_xa/XA/*
ksh prompt> cp $TUXDIR/samples/bankapp_xa/XA/*

Note: You cannot run the XA Bankapp sample application in the same work directory as the JDBC Bankapp sample application, because some of the files for the JDBC Bankapp sample application have the same name as files for the XA Bankapp sample application.
Source Files Used to Build the XA Bankapp Sample Application

Table 4-3 lists the files used to build and run the XA Bankapp sample application.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank.idl</td>
<td>The OMG IDL code that declares common structures and extensions for the XA Bankapp sample application.</td>
</tr>
<tr>
<td>BankApp.idl</td>
<td>The OMG IDL code that declares the TellerFactory and Teller interfaces.</td>
</tr>
<tr>
<td>BankDB.idl</td>
<td>The OMG IDL code that declares the DBAccess interface.</td>
</tr>
<tr>
<td>BankDB.icf</td>
<td>The ICF file that defines activation and transaction policies for the DBAccess interface.</td>
</tr>
<tr>
<td>BankDBServer.cpp</td>
<td>The C++ source code that implements the Server::initialize and Server::release methods for the C++ server application.</td>
</tr>
<tr>
<td>TellerFactoryImpl.java</td>
<td>The Java source code that implements the createTeller method.</td>
</tr>
<tr>
<td>TellerImpl.java</td>
<td>The Java source code that implements the verify, deposit, withdraw, inquiry, transfer, and report methods. In addition, it includes a reference to the TransactionCurrent environmental object and invokes operations on the DBAccess object within a transaction.</td>
</tr>
<tr>
<td>BankAppServerImpl.java</td>
<td>The Java source code that overrides the Server.initialize and Server.release methods.</td>
</tr>
<tr>
<td>Atm.java</td>
<td>The Java source code for the ATM client application.</td>
</tr>
<tr>
<td>BankStats.java</td>
<td>Contains methods to initialize, read from, and write to the flat file that contains the ATM statistics.</td>
</tr>
</tbody>
</table>
Step 2: Change the Protection Attribute on the Files for the XA Bankapp Sample Application

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

Windows NT
prompt>attrib -r drive:\workdirectory\*. *

UNIX
prompt>/bin/ksh
ksh prompt>chmod u+w /workdirectory/.*

Step 3: Verify the Settings of the Environment Variables

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

Environment Variables

Table 4-4 lists the environment variables required to run the XA Bankapp sample application.

Table 4-4 Required Environment Variables for the XA Bankapp Sample Application

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUXDIR</td>
<td>The directory path where you installed the WLE software. For example: Windows NT TUXDIR=c:\WLEdir UNIX TUXDIR=/usr/local/WLEdir</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>The directory path where you installed the JDK software. For example: Windows NT JAVA_HOME=c:\JDK1.2 UNIX JAVA_HOME=/usr/local/JDK1.2</td>
</tr>
<tr>
<td>ORACLE_HOME</td>
<td>The directory path where you installed the Oracle software. For example: ORACLE_HOME=/usr/local/oracle You need to set this environment variable on the Solaris operating system only.</td>
</tr>
</tbody>
</table>
Verifying Settings

To verify that the information defined during installation is correct:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.
3. Click the System icon.
   The System Properties window appears.
4. Click the Environment tab.
   The Environment page appears.
5. Check the settings for `TUXDIR`, `ORACLE_HOME`, and `JAVA_HOME`.

**UNIX**

ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME
ksh prompt>printenv ORACLE_HOME

Changing Settings

To change the settings:

**Windows NT**

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

**UNIX**

ksh prompt>TUXDIR=directorypath; export TUXDIR
ksh prompt>JAVA_HOME=directorypath; export JAVA_HOME
Building the XA Bankapp Sample Application

ksh prompt>JAVA_HOME=directorypath; export ORACLE_HOME

Note: If you are running multiple WLE applications concurrently on the same machine, you also need to set the IPCKEY and PORT environment variables. See the Readme.txt file for information about how to set these environment variables.

Step 4: Run the setupX Command

The setupX command automates the following steps:

1. Copy the required files from the \client and \shared directories.
2. Set the PATH, TOBJADDR, APPDIR, TUXCONFIG, and CLASSPATH system environment variables.
3. Create the UBBCONFIG file.
4. Create a setenvX.cmd or setenvX.ksh file that can be used to reset the system environment variables.

Enter the setupX command, as follows:

Windows NT
prompt> cd c:\mysamples\bankapp_xa\XA
prompt>setupX

UNIX
prompt>/bin/ksh
prompt> cd /usr/mysamples/bankapp_xa/XA/*
prompt>./setupX.ksh

Step 5: Load the UBBCONFIG File

Use the following command to load the UBBCONFIG file:

prompt>tmloadcf -y ubb_xa
Step 6: Create a Transaction Log

The transaction log records the transaction activities in a WLE session. 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 XA Bankapp sample application, the transaction log is placed in your work directory.

To open the transaction log for the XA Bankapp 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 TLOG
   crlog -m SITE1
   ```
   
   where
   
   - `blocks` specifies the number of blocks to be allocated for the transaction log and
   - `directorypath` indicates the location of the transaction log. The `directorypath` option needs to match the location specified in the `TLOGDEVICE` parameter in the `UBBCONFIG` file. The following is an example of the command on Windows NT:
   
   ```
   crdl -b 500 -z c:\mysamples\bankapp_java\XA\TLOG
   ```

3. Enter `quit` to exit the Interactive Administrative Interface.

Compiling the Client and Server Applications

The directory for the XA Bankapp sample application contains a make file that builds the client and server applications. During the development process, you use the `buildjavaserver` command to build the server application, and your Java product’s development commands to build the client application. However, for the XA Bankapp sample application, this step is included in the make file.
Use the following commands to build the client and server applications in the XA Bankapp sample application:

**Windows NT**

prompt>nmake -f makefileX.nt

**UNIX**

prompt>make -f makefileX.mk

### Initializing the Oracle Database

Use the following command to initialize the Oracle database used with the XA Bankapp sample application:

**Windows NT**

prompt>nmake -f makefileX.nt InitDB

**UNIX**

ksh prompt>make -f makefileX.mk InitDB

### Starting the Server Application in the XA Bankapp Sample Application

Start the server application in the XA Bankapp sample application by entering the following command:

prompt>tmboot -y
The `tmboot` command starts the application processes listed in Table 4-5.

### Table 4-5  Application Processes Started by `tmboot` Command

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMSYSEVT</td>
<td>BEA TUXEDO system event broker.</td>
</tr>
<tr>
<td>TMFFNAME</td>
<td>Three TMFFNAME server processes are started:</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process with the <code>-N</code> and <code>-M</code> options is the master NameManager service. The NameManager service maintains a mapping of the application-supplied names to object references.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process started with the <code>-N</code> option only is the slave NameManager service.</td>
</tr>
<tr>
<td></td>
<td>- The TMFFNAME server process started with the <code>-F</code> option contains the FactoryFinder object.</td>
</tr>
<tr>
<td>TMS_ORA</td>
<td>Transaction manager service.</td>
</tr>
<tr>
<td>BankDataBase</td>
<td>WLE server process that implements the <code>DBAccess</code> interface.</td>
</tr>
<tr>
<td>JavaServerXA</td>
<td>Server process that implements the <code>TellerFactory</code> and <code>Teller</code> interfaces. The JavaServer process has two options:</td>
</tr>
<tr>
<td></td>
<td>- <code>BankApp.jar</code>, which is the Java Archive (JAR) file that was created by the <code>buildjavaserver</code> command.</td>
</tr>
<tr>
<td></td>
<td>- <code>TellerFactory_1</code>, which is passed to the <code>Server.initialize</code> method.</td>
</tr>
<tr>
<td>JavaServerXA</td>
<td><code>JavaServerXA</code> is a special version of <code>JavaServer</code> that uses the same XA switch as the <code>BankDataBase</code> server process. It is created by the <code>buildXAJS</code> command.</td>
</tr>
<tr>
<td>ISL</td>
<td>IIOP Listener process.</td>
</tr>
</tbody>
</table>

### Files Generated by the XA Bankapp Sample Application

Table 4-6 lists the files generated by the XA Bankapp sample application.
Files Generated by the XA Bankapp Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubb_xa</td>
<td>The UBBCONFIG file for the XA Bankapp sample application. This file is generated by the setupX command.</td>
</tr>
<tr>
<td>setenvX.cmd and setenvX.ksh</td>
<td>Contains the commands to set the environment variables needed to build and run the XA Bankapp sample application. setenvX.cmd is the Windows NT version and setenvX.ksh is the UNIX Korn shell version of the file.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>A binary version of the UBBCONFIG file. Generated by the tmloadcf command.</td>
</tr>
<tr>
<td>TLOG</td>
<td>The transaction log.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command. The log file also contains messages generated by the server applications and the tmshutdown command.</td>
</tr>
<tr>
<td>.adm/.keybd</td>
<td>A file that contains the security encryption key database. The subdirectory is created by the tmloadcf command.</td>
</tr>
<tr>
<td>Atm$1.class</td>
<td>Used by the Java client application. Created when the Atm.java file is compiled.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AccountRecordNotFound.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the Bank.idl file. These files are created in the \com\beasys\samples\Bank subdirectory.</td>
</tr>
<tr>
<td>AccountRecordNotFoundException.java</td>
<td></td>
</tr>
<tr>
<td>AccountRecordNotFoundExceptionHolder.java</td>
<td></td>
</tr>
<tr>
<td>CustAccounts.java</td>
<td></td>
</tr>
<tr>
<td>CustAccountsHelper.java</td>
<td></td>
</tr>
<tr>
<td>CustAccountsHolder.java</td>
<td></td>
</tr>
<tr>
<td>DataBaseException.java</td>
<td></td>
</tr>
<tr>
<td>DataBaseExceptionHelper.java</td>
<td></td>
</tr>
<tr>
<td>DataBaseExceptionHolder.java</td>
<td></td>
</tr>
<tr>
<td>InsufficientFunds.java</td>
<td></td>
</tr>
<tr>
<td>InsufficientFundsHelper.java</td>
<td></td>
</tr>
<tr>
<td>InsufficientFundsHolder.java</td>
<td></td>
</tr>
<tr>
<td>PinNumberNotFound.java</td>
<td></td>
</tr>
<tr>
<td>PinNumberNotFoundHelper.java</td>
<td></td>
</tr>
<tr>
<td>PinNumberNotFoundHolder.java</td>
<td></td>
</tr>
<tr>
<td>BalanceAmounts.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the BankApp.idl file. These files are created in the \com\beasys\samples\BankApp subdirectory.</td>
</tr>
<tr>
<td>BalanceAmountsHelper.java</td>
<td></td>
</tr>
<tr>
<td>BalanceAmountsHolder.java</td>
<td></td>
</tr>
<tr>
<td>IOException.java</td>
<td></td>
</tr>
<tr>
<td>IOExceptionHelper.java</td>
<td></td>
</tr>
<tr>
<td>IOExceptionHolder.java</td>
<td></td>
</tr>
<tr>
<td>Teller.java</td>
<td></td>
</tr>
<tr>
<td>TellerActivity.java</td>
<td></td>
</tr>
<tr>
<td>TellerActivityHelper.java</td>
<td></td>
</tr>
<tr>
<td>TellerActivityHolder.java</td>
<td></td>
</tr>
<tr>
<td>TellerFactory.java</td>
<td></td>
</tr>
<tr>
<td>TellerFactoryHelper.java</td>
<td></td>
</tr>
<tr>
<td>TellerFactoryHolder.java</td>
<td></td>
</tr>
<tr>
<td>TellerInsufficientFunds.java</td>
<td></td>
</tr>
<tr>
<td>TellerInsufficientFundsHelper.java</td>
<td></td>
</tr>
<tr>
<td>TellerInsufficientFundsHolder.java</td>
<td></td>
</tr>
<tr>
<td>_TellerFactoryImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_TellerFactoryStub.java</td>
<td></td>
</tr>
<tr>
<td>_TellerImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_TellerStub.java</td>
<td></td>
</tr>
</tbody>
</table>
### Table 4-6 Files Generated by the XA Bankapp Sample Application (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountData.java</td>
<td>Generated by the m3idltojava command for the interfaces defined in the BankDB.idl file. These files are created in the \com\beasy\samples\BankDB subdirectory.</td>
</tr>
<tr>
<td>AccountDataHelper.java</td>
<td></td>
</tr>
<tr>
<td>AccountDataHolder.java</td>
<td></td>
</tr>
<tr>
<td>DBAccessHelper.java</td>
<td></td>
</tr>
<tr>
<td>_DBAccessImplBase.java</td>
<td></td>
</tr>
<tr>
<td>_DBAccessStub.java</td>
<td></td>
</tr>
<tr>
<td>Bankapp.ser</td>
<td>The Server Descriptor file and Server Java Archive file generated by the buildjavaserver command in the make file.</td>
</tr>
<tr>
<td>Bankapp.jar</td>
<td></td>
</tr>
<tr>
<td>Bank_c.cpp</td>
<td>Generated by the idl command for the interfaces defined in the Bank.idl file.</td>
</tr>
<tr>
<td>Bank_c.h</td>
<td></td>
</tr>
<tr>
<td>Bank_s.cpp</td>
<td></td>
</tr>
<tr>
<td>Bank_s.h</td>
<td></td>
</tr>
<tr>
<td>BankDB_c.cpp</td>
<td>Generated by the idl command for the interfaces defined in the BankDB.idl file.</td>
</tr>
<tr>
<td>BankDB_c.h</td>
<td></td>
</tr>
<tr>
<td>BankDB_s.cpp</td>
<td></td>
</tr>
<tr>
<td>BankDB_s.h</td>
<td></td>
</tr>
<tr>
<td>dbaccess_i.cpp</td>
<td>Generated from the DBAccess_i.pc file by the Oracle Pro*C/C++ compiler.</td>
</tr>
<tr>
<td>BankDataBase.exe</td>
<td>The WLE server application that implements the DBAccess interface.</td>
</tr>
<tr>
<td>TMS_ORA.exe</td>
<td>The server process for the Transaction Manager service.</td>
</tr>
<tr>
<td>JavaServerXA</td>
<td>The special version of the JavaServer that uses the same XA switches as the BankDataBase server process.</td>
</tr>
<tr>
<td>stderr</td>
<td>Generated by the tmboot command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.err.println method sends the output to the stderr file instead of to the ULOG file.</td>
</tr>
</tbody>
</table>
Starting the ATM Client Application in the XA Bankapp Sample Application

Start the ATM client application by entering the following command:

**Note:** The following command sets the Java CLASSPATH to include the current directory and the client JAR file (m3envobj.jar). The full WLE JAR file (m3.jar) can be specified instead of the client JAR file.

**Windows NT**

cmd prompt> java -classpath .;%TUXDIR%\udataobj\java\jdk\m3envobj.jar -DTOBJADDR=%TOBJADDR% Atm Teller2

**UNIX**

csh prompt> java -classpath .:$TUXDIR/udataobj/java/jdk/m3envobj.jar -DTOBJADDR=$TOBJADDR Atm Teller2

The GUI for the ATM client application appears.
Stopping the XA Bankapp Sample Application

Before using another sample application, enter the following commands to stop the XA Bankapp sample application and to remove unnecessary files from the work directory:

**Windows NT**
```
prompt>tmshutdown -y
prompt>nmake -f makefileX.nt clean
```

**UNIX**
```
ksh prompt>tmshutdown -y
ksh prompt>make -f makefileX.mk clean
```

Using the ATM Client Application

The ATM client application in the XA Bankapp sample application works as it does in the JDBC Bankapp sample application. For instructions, see “Using the ATM Client Application” on page 3-29.
Index

A
activation policies
  DBAccess interface 3-8
defining in Implementation
    Configuration file 4-5
defining in Server
    Description file 1-5
  JDBC Bankapp sample application 3-8
  Teller interface
    JDBC Bankapp
      sample application 3-8
    XA Bankapp
      sample application 4-5
  TellerFactory interface
    JDBC Bankapp
      sample application 3-8
    XA Bankapp
      sample application 4-5
ATM client application
  starting
    JDBC Bankapp
      sample application 3-24
    XA Bankapp
      sample application 4-20
  using
    JDBC Bankapp
      sample application 3-26
    XA Bankapp
      sample application 4-21

B
Bootstrap object
  use in client applications 1-4
building
  Java Simpapp sample application 2-4
  JDBC Bankapp sample application 3-12
  XA Bankapp sample application 4-6

C
client stubs
  generating 1-4
  in sample applications 1-4
compiling
  client applications
    Java Simpapp
      sample application 2-9
    JDBC Bankapp
      sample application 3-19
    XA Bankapp
      sample application 4-14
  server applications
    Java Simpapp
      sample application 2-9
    JDBC Bankapp
      sample application 3-19
    XA Bankapp
      sample application 4-14
D

database
  initializing Microsoft SQL Server 3-16
  initializing Oracle 3-16
  supported with JDBC Bankapp
    sample application 3-11
  use in JDBC Bankapp
    sample application 3-11
  use in XA Bankapp
    sample application 4-2, 4-6

database instance
  setting up local 4-6
  setting up remote 4-6

DBAccess interface
  activation policy 3-4
  description 3-4
  OMG IDL 3-6
  transaction policy 3-4
  use in JDBC Bankapp
    sample application 3-3
  use in XA Bankapp
    sample application 4-16

development process
  activation policies 1-5
  client applications 1-2
  client stubs 1-4
  illustrated 1-2
  Java server applications 1-2
  JDBC Bankapp sample application 3-4
  m3idltotojava command 1-4
  obtaining the OMG IDL code 1-4
  Server Description file 1-5
  skeletons 1-4
  transaction policies 1-5
  UBBCONFIG file 1-5
  writing Java server application code 1-5
  writing the client application code 1-4

directory location of source files
  Java Simpapp sample application 2-4
  JDBC Bankapp sample application 3-12

XA Bankapp sample application 4-7

E

evironment variables
  Java Simpapp sample application 2-7
  JDBC Bankapp sample application 3-16
  XA Bankapp sample application 4-11

F

FactoryFinder object
  in client applications 1-4

file protections
  Java Simpapp sample application 2-6
  JDBC Bankapp sample application 3-15
  XA Bankapp sample application 4-10

J

Java Simpapp sample application
  building 2-4
  changing protection on files 2-6
  compiling the C++
    client application 2-15
  compiling the Java
    client application 2-8
  compiling the Java
    server application 2-8
  description 2-2
  illustrated 2-2
  loading the UBBCONFIG file 2-8
  OMG IDL 2-3
  required environment variables 2-7
  runme command 2-8
  setting up the work directory 2-4
  source files 2-4, 2-5
  starting the C++ client application 2-15
  starting the Java client application 2-14
  starting the Java server application 2-14
  stopping 2-16

I-2  Guide to the Java Sample Applications
Guide to the Java Sample Applications

using the client applications 2-14

Java_HOME parameter
Java Simpapp sample application 2-7
JDBC Bankapp sample application 3-16
XA Bankapp sample application 4-11

JavaServer application process
description 3-9
Java Simpapp sample application 2-10
JDBC Bankapp sample application 3-21

JavaServerXA application process
description 4-16
XA Bankapp sample application 4-16

JDBC Bankapp sample application
building 3-12
changing protection on files 3-15
compiling the Java
client application 3-19
compiling the server application 3-19
description 3-2
development process 3-4
generated files 3-21
illustrated 3-3
initializing the database 3-19
loading the UBBCONFIG file 3-18
OMG IDL 3-4
required environment variables 3-16
setting up a work directory 3-12
setting up the database 3-11
setUpJ command 3-18
software requirements 3-16
starting the ATM
client application 3-24
starting the Java
server application 3-20
stopping 3-26
using JDBC drivers with 3-11
using the ATM
client application 3-26
jdbcKona/MSSQLServer driver
use in JDBC Bankapp
sample application 3-16

jdbcKona/Oracle driver
use in JDBC Bankapp
sample application 3-16

m3idltoljava command
generating client stubs 1-4
generating skeletons 1-4
method implementations
use in Java server applications 1-5

OMG IDL
changes for XA Bankapp
sample application 4-4
compiling 1-4
DBAccess interface 3-4
generating client stubs 1-4
generating skeletons 1-4
Java Simpapp sample application 2-3
Simple interface 2-3
SimpleFactory interface 2-3
Teller interface 3-4
TellerFactory interface 3-4

OPENINFO parameter 4-5
Oracle database 4-6
setting the XA parameter 4-5
setting up remote instance 4-6

ORACLE_HOME parameter
JDBC Bankapp sample application 3-16
XA Bankapp sample application 4-11

runme command
description 2-8
files generated by 2-10
**S**

Server object
  in Java server applications 1-5
  setting up local instance 4-6
  setupJ command
    files generated by 3-16
    use in JDBC Bankapp
      sample application 3-16
  setupX command
    files generated 4-12
    use in XA Bankapp
      sample application 4-12
  skeletons
    generating 1-4
    in sample applications 1-4
  software requirements
    JDBC Bankapp sample application 3-16
    XA Bankapp sample application 4-3
  source files
    Java Simpapp sample application 2-5
    JDBC Bankapp sample application 3-13
    XA Bankapp sample application 4-7
  starting
    ATM client application 3-24
  support
    documentation xiii
    technical xiv

**T**

Teller interface
  activation policy 3-4
  description 3-4
  OMG IDL 3-5
  transaction policy 3-4
  use in JDBC Bankapp
    sample application 3-3
  TLOGDEVICE parameter 4-6
  tmboot command
    use in the Java Simpapp
      sample application 2-14
    use in XA Bankapp
      sample application 4-15
  TMFFNAME application process
    Java Simpapp sample application 2-10
    JDBC Bankapp sample application 3-21
    XA Bankapp sample application 4-15
  tmloadcf command
    JDBC Bankapp sample application 3-18
    XA Bankapp sample application 4-13
  TMS_ORA
    use in XA Bankapp
      sample application 4-16
  TMSYSEVT application process
    Java Simpapp sample application 2-10
    JDBC Bankapp sample application 3-21
    XA Bankapp sample application 4-15
  transaction log
    creating 4-13
  transaction manager
    TMS_ORA 4-16
    use in XA Bankapp
      sample application 4-2
  transaction policies
    DBAccess interface 3-8
    defining in Server Description file 1-5
    for Teller interface
      JDBC Bankapp
        sample application 3-8
      XA Bankapp
        sample application 4-5
    for TellerFactory interface
      JDBC Bankapp
        sample application 3-8
      XA Bankapp

I-4  Guide to the Java Sample Applications
sample application 4-5
Implementation Configuration file 4-5
XA Bankapp sample application 4-5

TUXCONFIG file

description 1-5

TUXDIR parameter

Java Simpapp sample application 2-7
JDBC Bankapp sample application 3-16
XA Bankapp sample application 4-11

UBBCONFIG file

Java Simpapp sample application 2-8
JDBC Bankapp sample application 3-9
XA Bankapp sample application 4-5

XA Bankapp sample application

building 4-6
changing protection on files 4-10
compiling client applications 4-14
compiling server applications 4-14
creating a transaction log 4-13
description 4-2
development process 4-3
generated files 4-16
illustrated 4-2
initializing the database 4-15
loading the UBBCONFIG file 4-13
OMG IDL 4-4
required environment variables 4-11
setting up a work directory 4-7
setupX command 4-12
software requirements 4-3
starting the ATM
   client application 4-20, 4-21
starting the Java
   server application 4-15
stopping 4-21