BEA Tuxedo

BEA Tuxedo Interoperability

BEA Tuxedo Release 8.0
Document Edition 8.0
June 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA Tuxedo Interoperability

Document Edition	Date	Software Version
8.0	June 2001	BEA Tuxedo 8.0

Contents

2. Connectivity Between a BEA Tuxedo CORBA Client and an EJB in WebLogic Server

3. Connectivity from WebLogic Server to a CORBA Object

Index

About This Document

This document presents an overview of BEA Tuxedo® interoperability with other BEA software components and third-party software providers. This document identifies interoperability solutions, outlines some solutions, and provides references to other documentation and sample applications.

This document includes the following topics:

- Chapter 1, “Introduction,” provides a high-level overview of the interoperability and coexistence capabilitities in the BEA Tuxedo system between the WebLogic Server J2EE and BEA Tuxedo programming models. Additionally, this chapter provides information on interoperability solutions between specific components.
- Chapter 2, “Connectivity Between a BEA Tuxedo CORBA Client and an EJB in WebLogic Server,” provides detailed information about the WLStrader Value Type sample program, stepping you through the build and run processes for this sample application.
- Chapter 3, “Connectivity from WebLogic Server to a CORBA Object,” provides supplementary information about the WebLogic Enterprise Connectivity (WLEC) EJB simpapp sample application.

What You Need to Know

This document is intended for application developers who are interested in creating secure, scalable, transaction-based server applications that interoperate between components in the BEA product suite. It assumes you are knowledgeable about CORBA, Enterprise JavaBeans, and the C++ and Java programming languages. This document is also a resource for locating information about other BEA interoperability solutions.

e-docs Web Site

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

How to Print the Document

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

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

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

Related Information

For more information about CORBA, BEA Tuxedo, distributed object computing, transaction processing, C++ programming, and Java programming, see the *CORBA Bibliography* in the BEA Tuxedo online documentation.

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 8.0 release.

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

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

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

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o` *`name`* `] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

Convention	Item
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

. . .	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Introduction

This topic includes the following sections:

- Interoperability Between the WLS J2EE and BEA Tuxedo Programming Models
- Interoperability Solutions
- Interoperability Sample Applications

Interoperability Between the WLS J2EE and BEA Tuxedo Programming Models

This section describes the interoperability and coexistence capabilities in the BEA Tuxedo system between the WebLogic Server J2EE and Tuxedo programming models. The key interoperability features are presented in the following categories:

- BEA Tuxedo Server Interoperability
- BEA Tuxedo Client and Server Interoperability
- BEA WebLogic Server and BEA Tuxedo Interoperability
- BEA Tuxedo Interoperability with Third-Party ORBs
- BEA Tuxedo Interdomain Interoperability

A summary description of BEA clients and servers follows.

Note the following definitions:

- **BEA client**

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

  - Jolt client application (through the Jolt listener/handler)
  - BEA Tuxedo /WS client application (through the Tuxedo /WS listener/handler)
  - BEA Tuxedo CORBA client application (through the IIOP listener/handler)
  - ActiveX client application (through the IIOP listener/handler)
  - RMI client application (through the IIOP listener/handler)
  - EJB on BEA WebLogic Server using WebLogic Enterprise Connectivity (WLEC)

  Note that a BEA Tuxedo client invoking another BEA Tuxedo client is not supported.

- **BEA Tuxedo server**

  A BEA Tuxedo server includes Tuxedo services and CORBA objects that run on the BEA Tuxedo system. These servers run within the administrative unit of a BEA Tuxedo domain and are configured through a `UBBCONFIG` file.

BEA Tuxedo Server Interoperability

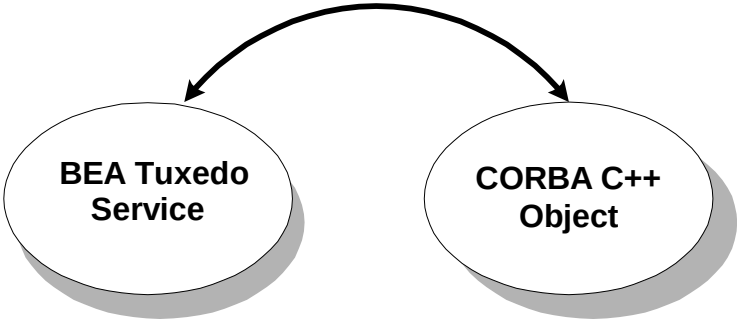
This section describes the interoperability between the following BEA Tuxedo server components:

- BEA Tuxedo ATMI service
- CORBA C++ object

Figure 1-1 shows the direct interoperability support between BEA Tuxedo server applications.

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

Figure 1-1 BEA Tuxedo Server Interoperability



A C++ object can include ATMI calls to BEA Tuxedo services. See the Wrapper University sample application, available in the *Guide to the CORBA University Sample Applications*, for a sample application that demonstrates this feature.

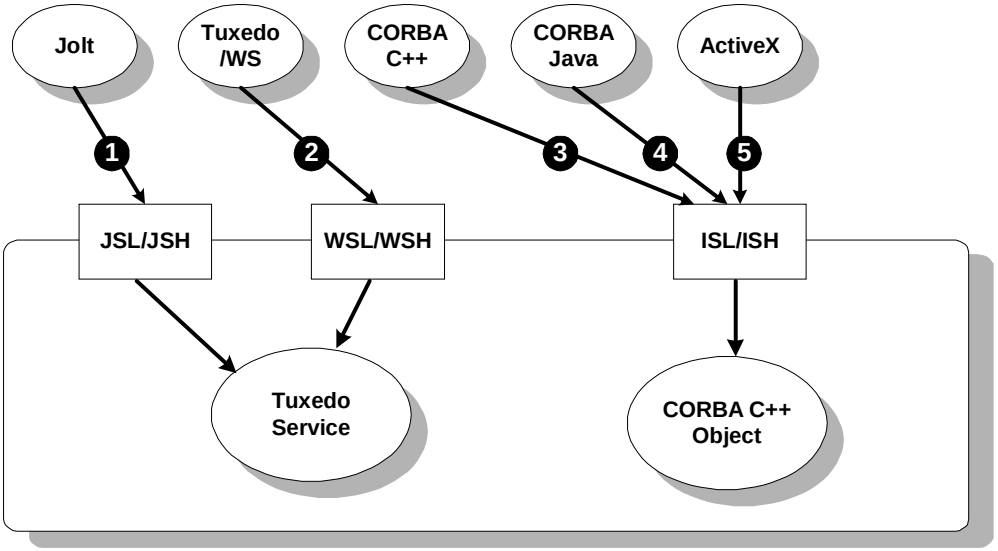
Transactions and Security

Transaction and security context propagation between server applications running in a BEA Tuxedo domain is fully supported.

BEA Tuxedo Client and Server Interoperability

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

Figure 1-2 Remote Client and Server Interoperability



Note the following information illustrated in the preceding figure:

1. Jolt client application invoking a BEA Tuxedo service

   A Jolt client can invoke a BEA Tuxedo service running in the BEA Tuxedo domain through a Jolt listener/handler. For more information about Jolt, see *Using BEA Jolt* and the *BEA Jolt API Reference* in the online BEA Tuxedo documentation.

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

   A BEA Tuxedo /WS client application can invoke a BEA Tuxedo service running in the BEA Tuxedo domain using the workstation listener/handler.

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

   A BEA CORBA C++ client application can invoke CORBA C++ objects. For more information, see *Creating CORBA Client Applications*.

4. BEA CORBA Java client application invoking a CORBA object

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

5. BEA ActiveX client application invoking a CORBA object

   A BEA ActiveX client application can invoke CORBA C++ objects running in a BEA Tuxedo domain using the IIOP listener/handler. For more information, see *Creating CORBA Client Applications*.

The following additional invocation paths are also supported in the BEA Tuxedo environment using proxy objects or servers:

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

  You can create a C++ client with a set of operations that maps one-to-one with calls to BEA Tuxedo services using an intermediary C++ server-side object. For an example application illustrating this feature, see the Wrapper University sample application in the *Guide to the CORBA University Sample Applications*.

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

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

Transactions and Security

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

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

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

About BEA Jolt

BEA Jolt provides a mechanism for allowing Java clients to make ATMI calls on BEA Tuxedo services that exist in a BEA Tuxedo domain. Jolt also provides a mechanism for allowing BEA WebLogic Server to invoke BEA Tuxedo services. This latter capability is performed through Jolt connection pools and is described in the next section.

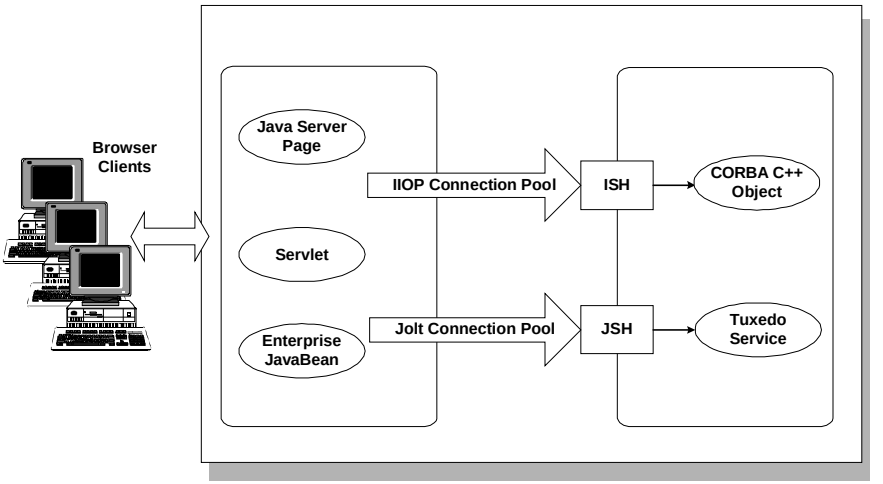
For more information, see the following documentation:

- For information about Jolt, see *Using BEA Jolt*.
- For information about setting up Jolt connection pools to connect WebLogic Server to BEA Tuxedo, see *Using BEA Jolt with BEA WebLogic Server*.

BEA WebLogic Server and BEA Tuxedo Interoperability

WebLogic Server applications can invoke CORBA objects and BEA Tuxedo services in a BEA Tuxedo domain. Figure 1-3 illustrates the use of connection pools for these implementations.

Figure 1-3 WebLogic Server and BEA Tuxedo Interoperability



Note the following about these connection pools:

- IIOP connection pools allow BEA WebLogic Server applications to invoke CORBA objects in a BEA Tuxedo domain. For information about setting up and using IIOP connection pools, see WebLogic Enterprise Connectivity (WLEC) topics in WebLogic Server documentation.
- Jolt connection pools allow BEA WebLogic Server applications to invoke BEA Tuxedo services in a BEA Tuxedo domain. For information about setting up and using Jolt connection pools, see *Using BEA Jolt with BEA WebLogic Server*.

BEA Tuxedo Interoperability with Third-Party ORBs

This release of BEA Tuxedo includes support for the CORBA Services Interoperable Naming Service (INS) to enhance interoperability with third-party ORB applications. The addition of INS enables third-party ORBs that use INS to interoperate with the BEA Tuxedo CORBA server ORB. Using INS, third-party ORBs can execute the following operations on BEA Tuxedo CORBA servers without requiring the use of the BEA Bootstrap, SecurityCurrent, or TransactionCurrent environmental objects:

- Bootstrapping
- Authentication
- Starting transactions

Note: BEA Tuxedo CORBA client environmental objects continue to be supported in BEA Tuxedo 8.0, as they were in BEA WebLogic Enterprise Version 5.1.

BEA Tuxedo Interdomain Interoperability

A server application running in one BEA Tuxedo domain can interoperate with a server application running in another BEA Tuxedo domain through the domain gateway. For more information about interdomain BEA Tuxedo interoperability, see the *BEA Tuxedo Product Overview*. and *Administering a BEA Tuxedo Application at Run Time* in the BEA Tuxedo online documentation.

Interoperability Solutions

This section provides information on interoperability solutions between these components:

- Connectivity from BEA WebLogic Server to BEA Tuxedo ATMI
- Connectivity from BEA Tuxedo CORBA to BEA WebLogic Server
- Connectivity from BEA WebLogic Server to BEA Tuxedo CORBA

Connectivity from BEA WebLogic Server to BEA Tuxedo ATMI

This section provides information on options for connectivity between WebLogic Server and BEA Tuxedo ATMI. It includes the following sections:

- Using BEA Jolt
- Using WebLogic Tuxedo Connector (WTC) Version 1.0

Using BEA Jolt

With BEA Jolt for BEA WebLogic Server, you can enable BEA Tuxedo services for the Web, using WebLogic Server as the front-end HTTP and applications server.

BEA Jolt is a Java-based client API that manages requests to BEA Tuxedo services using a Jolt Service Listener (JSL) running on the Tuxedo server. The Jolt API is embedded in the WebLogic Server API, and is accessible from a servlet, JHTML, or other BEA WebLogic application.

The product consists of two main components: the Jolt Class Library and the Jolt Repository. With BEA Jolt, you can create secure, scalable transactions, over the Internet, between clients and severs.

For complete information on using BEA Jolt with WebLogic Server, see *Using BEA Jolt with BEA WebLogic Server* in the BEA Tuxedo online documentation. This document explains the operation of BEA Jolt for WebLogic Server, and describes how to use, configure, and integrate BEA Jolt, BEA Tuxedo ATMI, and WebLogic Server. The document includes these sample programs:

- Appendix B, Simple Servlet Example

  This example demonstrates how to use BEA Jolt to connect to BEA Tuxedo ATMI from a WebLogic servlet.

- Appendix C, Servlet with Enterprise JavaBean Example

  This example demonstrates an EJB interface for accessing BEA Tuxedo ATMI.

Note: The current release of BEA Jolt does not support bidirectional connectivity or transaction context propagation.

The current release of BEA Jolt has been enhanced to support:

- XML buffer types.

  You can send an eXtensible Markup Language (XML) buffer from a Jolt client to a BEA Tuxedo ATMI Service using Data Dependent Routing (DDR.) A Jolt client can receive an XML document from a BEA Tuxedo ATMI Service.

- Transparent end-to-end user authentication for BEA Tuxedo ATMI service requests initiated by WebLogic Server.

  WebLogic Server-authenticated user credentials are mapped to the appropriate security interfaces/protocols, and an incoming request does not require re-authentication before invoking BEA Tuxedo ATMI services.

- Jolt Connection Pooling, which supports connection pool reset in the event of connection pool failure.

  In a WebLogic Server/BEA Tuxedo environment, this eliminates the requirement to restart BEA WebLogic Server if the connection pool requires a restart.

Using WebLogic Tuxedo Connector (WTC) Version 1.0

WebLogic Tuxedo Connector version 1.0 delivers bidirectional interoperability between WebLogic Server and BEA Tuxedo ATMI. This allows BEA Tuxedo applications using ATMI APIs to take advantage of the advanced messaging facilities provided by JMS.

BEA Tuxedo services makes calls through WebLogic Tuxedo Connector using ATMI. Communication with Java programs using the connector is transparent to BEA Tuxedo ATMI clients. WebLogic Tuxedo Connector extracts the contents of the BEA Tuxedo ATMI buffer and presents the contents to a Java application.

WebLogic Tuxedo Connector also provides a Java API that enables invocation from WLS servlets, JSPs, and EJBs on BEA Tuxedo ATMI services without changes to the ATMI services.

For information on the features and operation of WebLogic Tuxedo Connector 1.0, see `http://e-docs.bea.com/wtc/wtc10/index.html` in the online documentation. These pages provide WebLogic Tuxedo Connector 1.0 documentation on the following specific topics:

- Administration Guide: `http://e-docs.bea.com/wtc/wtc10/admin`
- ATMI Guide: `http://e-docs.bea.com/wtc/wtc10/atmi`

Connectivity from BEA Tuxedo CORBA to BEA WebLogic Server

This section provides information on options for connectivity between BEA Tuxedo CORBA Server and BEA WebLogic Server. It includes the following sections:

- Using RMI/IIOP and IDL Interfaces
- Using WebLogic Tuxedo Connector (WTC) Version 1.1

Using RMI/IIOP and IDL Interfaces

BEA Tuxedo CORBA C++ clients, and CORBA C++ servers acting as clients, support the CORBA standard of passing objects by value. This enables CORBA client support for RMI over IIOP invocations on IDL interfaces for WebLogic Server EJBs.

The BEA Tuxedo CORBA WLStrader Value Type client application employs RMI over IIOP and IDL interfaces to connect to the Trader EJB in WebLogic Server. See Chapter 2, “Connectivity Between a BEA Tuxedo CORBA Client and an EJB in WebLogic Server,” for complete information on how to build and run this sample application.

Using WebLogic Tuxedo Connector (WTC) Version 1.1

WebLogic Tuxedo Connector version 1.1 enhances the bidirectional interoperability between BEA WebLogic Server and BEA Tuxedo components. This gateway provides interoperability between BEA WebLogic Server and both BEA Tuxedo CORBA and BEA Tuxedo ATMI. This product provides functional equivalence to BEA Jolt and WebLogic Enterprise Connectivity. Unlike Jolt, this product does not provide transparent migration.

Using this version of WebLogic Tuxedo Connector, a WebLogic Server programmer can invoke CORBA C++ objects using the standard CORBA API. WebLogic Tuxedo Connector supports routing of CORBA invocations through the BEA Tuxedo domain gateway to the appropriate CORBA C++ object. WebLogic Tuxedo Connector supports security and transaction propagation from BEA WebLogic Server to BEA Tuxedo.

WebLogic Tuxedo Connector version 1.1 supports all of the functionality included in version 1.0.

BEA Systems encourages the use of WebLogic Tuxedo Connector version 1.1 for WebLogic Server/BEA Tuxedo connectivity.

Connectivity from BEA WebLogic Server to BEA Tuxedo CORBA

This topic provides information on options for connectivity between BEA WebLogic Server and BEA Tuxedo CORBA. It includes the following sections:

- Using WebLogic Enterprise Connectivity (WLEC)
- Using WebLogic Tuxedo Connector (WTC) Version 1.1

Using WebLogic Enterprise Connectivity (WLEC)

WebLogic Enterprise Connectivity (WLEC) is a component of WebLogic Server. This component enables the use of IIOP connection pooling from WebLogic Server clients, including servlets, EJBs, JSPs, and RMI objects, to invoke CORBA, EJB, and other RMI objects.

The key features of WebLogic Enterprise Connectivity are:

- Pooled IIOP connections to the BEA Tuxedo system.
- Multiple active BEA Tuxedo CORBA client transactions from a single WebLogic Server process.
- Configuration of IIOP connection pools through the `weblogic.properties` file.
- Monitoring of IIOP connection pools through the WebLogic Console.
- Secure Sockets Layer (SSL) support.
- Security context propagation from WebLogic Server to BEA Tuxedo CORBA.
- Pool reinitialization at run time.

Note: WebLogic Enterprise Connector is CORBA 2.2 compliant. WebLogic Enterprise Connectivity does not support passing of objects by value, in value types.

For complete information on WebLogic Enterprise Connectivity, see WebLogic Enterprise Connectivity (WLEC) topics in the BEA WebLogic Server online documentation.

Documentation provided at the BEA WebLogic Server installation location includes examples showing how to use WebLogic Enterprise Connectivity to access BEA WebLogic Enterprise or BEA Tuxedo CORBA objects from servlets, JSP and Enterprise JavaBeans running on a BEA WebLogic Server. The files are in this location: *`wlserver6`*`\samples\examples\wlec\package-summary.html`.

For supplementary information on building and running the WebLogic Server WLEC EJB simpapp application to connect to a BEA Tuxedo CORBA object, see "Connectivity from WebLogic Server to a CORBA Object."

Using WebLogic Tuxedo Connector (WTC) Version 1.1

See "Using WebLogic Tuxedo Connector (WTC) Version 1.1" on page 1-11 for information on WebLogic Tuxedo Connector Version 1.1.

Interoperability Sample Applications

This section lists the interoperability sample applications provided with BEA Tuxedo and BEA WebLogic Server software. The sample applications provide client and server programmers with information about the basic concepts of combining Enterprise JavaBeans (EJBs) and CORBA objects in an application.

The BEA Tuxedo and WebLogic Server software include the sample applications outlined in Table 1-1.

Table 1-1 Interoperability Sample Applications

Application	Description
WLStrader Value Type sample application	Demonstrates connectivity between a CORBA C++ client application using RMI over IIOP, and passing objects by value to an EJB in WebLogic Server.
WLEC EJB simpapp sample application	Demonstrates how to use WebLogic Enterprise Connectivity (WLEC) to access a BEA Tuxedo CORBA object from a stateless EJB on WebLogic Server.

2 Connectivity Between a BEA Tuxedo CORBA Client and an EJB in WebLogic Server

This topic includes the following sections:

- Overview of the WLStrader Value Type Sample Application
- Components of the WLStrader Value Type Sample Application
- Building and Running the WLStrader Value Type Sample Application
- Additional Sources of Information

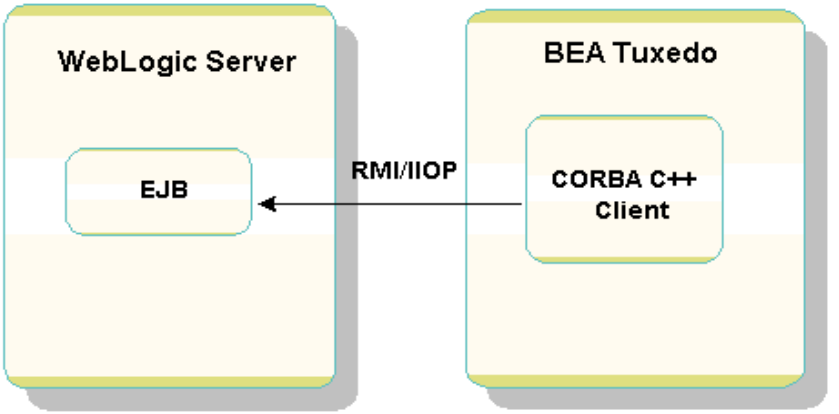
Overview of the WLStrader Value Type Sample Application

The WLStrader Value Type sample application demonstrates connectivity between an RMI over IIOP client and an EJB. More specifically, this sample application illustrates how a CORBA C++ client application developed in BEA Tuxedo can interact with an

EJB in WebLogic Server. The application makes use of the Trader EJB included in the WebLogic Server examples. The WLStrader Value Type sample builds on components of the Trader sample application, while providing its own build and run procedures.

Figure 2-1 illustrates the connectivity provided in the WLStrader Value Type sample application.

Figure 2-1 CORBA C++ Client to WebLogic Server EJB



In the WLStrader Value Type example, a BEA Tuxedo CORBA C++ client employs a value type, passing an object-by-value. This supports an RMI over IIOP invocation of an IDL interface, and provides connectivity to the WebLogic Server TraderBean Enterprise JavaBean (EJB).

A value type represents a class whose values can be moved between systems. Value types can be passed as arguments or results of remote methods, or as fields within other objects that are passed remotely. For information on support for value types in BEA Tuxedo CORBA, see Chapter 13, "Mapping of OMG IDL Statements to C++," in the *CORBA Programming Reference* in the BEA Tuxedo online documentation.

The CORBA client obtains the WebLogic Server Inter-operable Object Reference (IOR) by invoking the COSNaming Service layer provided in the WebLogic Server JNDI implementation. For information on the CORBA Name Service and the COSNaming data structures, see *Using the CORBA Name Service* in the BEA Tuxedo online documentation.

Components of the WLStrader Value Type Sample Application

This topic describes the components and operations in the WLStrader Value Type sample application. It includes the following sections:

- WebLogic Server Trader Sample Application
- Mapping from a WebLogic Server to a CORBA Client
- BEA Tuxedo CORBA C++ Client

WebLogic Server Trader Sample Application

The WLStrader Value Type sample builds on components of the WebLogic Server Trader sample application, employing its XML deployment files and Trader EJB. To derive the optimum benefit from this example, read through the source code files to understand the design and the steps involved. Review the deployment files to understand the general structure of the Trader EJB.

The WebLogic Server code sample files for the Trader sample are in the *wlserver6*`\samples\examples\rmi_iiop\ejb\rmi_iiop` directory.

Table 2-1 provides a summary of the Java interfaces in this example.

Table 2-1 Java Interface Summary

Java Interface	Description
`Trader`	The methods in this interface are the public face of `TraderBean`.
`TraderHome`	This interface is the home interface for `TraderBean.java`, which in WebLogic is implemented by the code-generated container class `TraderBeanC`.

Table 2-2 provides a summary of the Java classes in this example.

Table 2-2 Java Class Summary

Java Class	Description
`Tuxclient`	This C++ client program logic invokes a stateless Session Bean and performs the following exercises: creates a Trader, buys some shares using the Trader, retrieves the `NumberTraded` returned in a value type, and removes the Trader.
`TraderBean`	TraderBean is a stateless Session Bean.
`TradeResult`	This class implements the serializable interface, and does not have any classes generated on the value type. The state of `TradeResult` is maintained in private instance members, and there is no default public access to the state of the value type. Because `TradeResult` implements the serializable interface, the `ejbc` utility generates a value type for it in the IDL file.

Mapping from a WebLogic Server to a CORBA Client

When deriving the mapping from a WebLogic Server application to a CORBA client, the sources of the mapping information are the EJB classes as defined in the Java source files. WebLogic Server provides the `weblogic.ejbc` utility for generating required CORBA IDL files. These files represent the CORBA view into the state and behavior of the target EJB.

The `weblogic.ejbc` utility performs the following:

- Places the EJB classes, interfaces, and deployment descriptor files in a JAR file.
- Generates WebLogic Server container classes for the EJBs.
- Runs each EJB container class through the RMI compiler to create stubs and skeletons.
- Generates a directory tree of CORBA IDL files describing the CORBA interface to these classes.

The `weblogic.ejbc` utility supports a number of command qualifiers. In the WLStrader Value Type example, the `ejbc` step invokes these qualifiers on the command line:

- `-idl`

  Directs the utility to create CORBA Interface Definition Language (IDL) files for all appropriate classes.

- `-idlDirectory idlSources`

  Directs the utility to create a directory tree named `idlSources`, and to store IDL files in this location. The directory tree structure corresponds to the Java package hierarchy.

- `-idlOverwrite`

  Directs the utility to overwrite any existing IDL files in the `idlSources` output directory.

Resulting files are processed using the BEA Tuxedo IDL compiler, reading source files from the `idlSources` directory and generating CORBA C++ stub and skeleton files. These generated files are sufficient for all CORBA data types with the exception of value types. Value types need to be implemented on each platform on which they are defined or referenced. Specifying the `-i` qualifier directs the IDL compiler to create implementation files named *`FileName`*`_i.h` and *`FileName`*`_i.cpp`. For example, this syntax creates the `TradeResult_i.h` and `TradeResult_i.cpp` implementation files:

```
idl -IidlSources -i
idlSources\examples\rmi_iiop\ejb\rmi_iiop\TradeResult.idl
```

The resulting source files provide implementations for application-defined operation on a value type. Implementation files are included in a CORBA client application.

BEA Tuxedo CORBA C++ Client

The CORBA C++ client program `tuxclient` performs the exercises against the Trader. The results from a `Trader` transaction are returned to the client through the `TradeResult` value type. The `tuxclient` application performs these operations:

- Registers value factories for value types from which it expects to receive output. Listing 2-1 is a code extract from `tuxclient.cpp`, illustrating the `register_value_factory` operation.

Listing 2-1 tuxclient.cpp—Register Value Factories

```
 . . .
// Need to register value factories for all value types we
// might receive.
//
examples_rmi_iiop_ejb_rmi_iiop_TradeResult_factory* TRf = new
    examples_rmi_iiop_ejb_rmi_iiop_TradeResult_factory();
orb->register_value_factory((char*const)
   examples::rmi_iiop::ejb::rmi_iiop::_tc_TradeResult->id(), TRf);
//
 . . .
```

- Reads the `ior.txt` file containing the stringified IOR for the WebLogic Server.
- Converts the IOR to an object.
- Obtains a COSNaming context.
- Creates an instance of a `Trader`.
- Buys some shares by issuing a trade request.
- Employs `WStringValue` value types for specifying the stock symbol and number of shares requested.
- Obtains the actual number of shares traded from the `TradeResult` value type.
- Reports the actual number of shares bought.

The code extract from `tuxclient.cpp` in Listing 2-2 contains the steps for creating a `Trader` instance and buying some shares of BEAS.

Listing 2-2 tuxclient.cpp—Create a Trader; Buy Some Shares

```
 . . .
// Create a Trader instance
::examples::rmi_iiop::ejb::rmi_iiop::Trader_ptr trader =
     home->create();

// Buy some shares
CORBA::WStringValue_ptr BEASsym = new
    CORBA::WStringValue(CORBA::wstring_dup((wchar_t *)L"BEAS"));
```

```
::examples::rmi_iiop::ejb::rmi_iiop::TradeResult_ptr result;
cout << "Buying 3000 shares of BEAS" << endl;
result = trader->buy(BEASsym, 3000);
 . . .
```

The “buy” request from the CORBA client is for 3000 shares of BEA stock with symbol BEAS. The actual number of shares traded is 500, imposed by the `tradeLimit` threshold defined in the `ejb-jar.xml` deployment file for the WebLogic Server examples.

Listing 2-3 tradeLimit Defined in ejb-jar.xml

```
 . . .
<env-entry-name>tradeLimit</env-entry-name>
<env-entry-type>java.lang.Integer</env-entry-type>
<env-entry-value>500</env-entry-value>
 . . .
```

To achieve different results when you run the WLStrader Value Type sample application, you can vary definitions in the `ejb-jar.xml` file, or constants in the `tuxclient.cpp` source file.

Building and Running the WLStrader Value Type Sample Application

This section leads you through the process of building and running the WLStrader Value Type sample application.

Before running the example, described later in the sequence of steps, you must obtain the WebLogic Server Inter-operable Object Reference (IOR) by running the `host2ior` utility. Complete this step once for any WebLogic Server installation. You can obtain this file any time after installing WebLogic Server, and before running an application requiring the file.

To build and run the wlstrader sample application, complete the following steps. Each step is described in detail in subsequent sections. The steps detailed here supplement the information in the WebLogic Server installation location: *wlserver6*\samples\examples\rmi_iiop\ejb\rmi_iiop\package-summary.html.

- Set the Development Environment
- Copy the BEA Tuxedo WLStrader Value Type Files
- Build the Example
- Configure the Server
- Run the Example

Set the Development Environment

When developing in Java, make sure you have a controlled development environment.

1. Set your development environment as described in the *WebLogic Server Examples Guide* (which is included with the product). See *Setting Up Your Environment for Building and Running the Examples*.
2. From your user interface tool or from a command prompt, set the `TUXDIR` environment variable to the BEA Tuxedo installation location.

   For example:

   Windows

   ```
   > set TUXDIR=D:\TUXDIR
   ```

   UNIX

   ```
   ksh prompt> export TUXDIR=/usr/local/TUXDIR
   ```

Copy the BEA Tuxedo WLStrader Value Type Files

Rename the `build` command script, and copy the BEA Tuxedo WLStrader Value Type files from the BEA Tuxedo CORBA samples directory to your WebLogic Server examples build area.

1. Use a command prompt or user interface tool to rename the `build` command script to a unique, recognizable name. The remainder of this text uses the script name `tuxbuild`. You will be copying this file into a WebLogic Server examples directory that already contains a `build` command script.

   For example:

   Windows

   ```
   rename %TUXDIR%\samples\corba\wlstrader\build.cmd tuxbuild.cmd
   ```

   UNIX

   ```
   mv $TUXDIR/samples/corba/wlstrader/build.sh tuxbuild.sh
   ```

2. From a command prompt or user interface tool, copy files from the BEA Tuxedo CORBA samples `wlstrader` directory to your WebLogic Server examples build area.

   For example:

   Windows

   ```
   copy %TUXDIR%\samples\corba\wlstrader\*.*
   D:\bea\wlserver6\samples\examples\rmi_iiop\ejb\rmi_iiop\*.*
   ```

   UNIX

   ```
   cp $TUXDIR/samples/corba/wlstrader/*.*
   /usr/local/wlserver6/samples/examples/rmi_iiop/ejb/rmi_iiop/*.*
   ```

The sample files listed in Table 2-3 should now be in your WebLogic Server examples build area.

Table 2-3 WLStrader Value Type Sample Files

File	Description
`TradeResult_i.cpp`	C++ source defining the constructor for the TradeResult value type. `TradeResult_i.cpp` and `TradeResult_i.h` provide the actual implementation of the concrete class for the TradeResult value type.
`TradeResult_i.h`	The C++ header file that defines the implementation class for the TradeResult value type.
`tuxbuild.cmd`	The renamed build command file for Windows systems.

Table 2-3 WLStrader Value Type Sample Files (Continued)

File	Description
`tuxbuild.sh`	The renamed build command script for UNIX systems.
`tuxclient.cpp`	C++ source code for the CORBA client application.

Build the Example

Execute the renamed build script for this example. This is the file you renamed before copying from the BEA Tuxedo `samples\corba\wlstrader` directory into the `samples\examples\rmi_iiop\ejb\rmi_iiop` directory of your WebLogic Server installation. The script performs the following steps presented here for a Windows environment:

1. Sets the `TUXDIR` environment variable. You can modify this step in the `tuxclient.cmd` script, or comment out the step if `TUXDIR` is already defined.
2. Sets the `IDL2CPP` environment variable, defining the `idl` executable program in the `%TUXDIR%\bin` directory and setting the required command-line parameters.
3. Creates a `build` directory structure, copies in the deployment descriptors, and copies in `*.gif` images.

   ```
   > mkdir build build\META-INF build\images
   > copy *.xml build\META-INF
   > copy *.gif build\images
   ```

4. Compiles EJB classes into the build directory (JAR preparation).

   ```
   > javac -d build Trader.java TraderHome.java
           TraderResult.java TraderBean.java
   ```

5. Makes an EJB JAR file, including the XML deployment descriptors.

   ```
   > cd build
   > jar cv0f std_ejb_over_iiop.jar META-INF examples images
   > cd ..
   ```

6. Runs the `weblogic.ejbc` utility against the JAR file, directing generated IDL files to the `idlSources` directory.

   ```
   > java weblogic.ejbc -compiler javac -keepgenerated
     -idl -idlDirectory idlSources
     -iiop build\std_ejb_over_iiop.jar
     %APPLICATIONS%\ejb_over_iiop.jar
   ```

7. Compiles the EJB interfaces and client application into the directory defined by the `CLIENT_CLASSES` target variable.

   ```
   > javac -d %CLIENT_CLASSES% Trader.java TraderHome.java
     TradeResult.java Client.java
   ```

8. Runs the `IDL` compiler against the `IDL` files built in the `weblogic.ejbc` step, creating C++ source files.

   ```
   >%IDL2CPP% idlSources\examples\rmi_iiop\ejb\rmi_iiop\Trader.idl
     ...
   >%IDL2CPP% idlSources\javax\ejb\RemoveException.idl
   ```

9. Builds the CORBA C++ client executable program `tuxclient.exe` by invoking the `buildobjclient` command.

Configure the Server

Start the WebLogic Server by reviewing and implementing the steps defined in the WebLogic Server Examples documentation. See *Starting WebLogic Server with the Examples Configuration.* Review the configuration file containing the configuration attributes for all of the examples, located in `config\examples\config.xml` in the WebLogic Server installation location. To run a start server script:

Windows

```
prompt> startExampleServer
```

Alternatively, on Windows platforms you can start the WebLogic Server with the examples configuration from the Windows Start Menu.

UNIX

```
$ sh startExamplesServer.sh
```

Run the Example

Run the WLStrader Value Type sample program by setting up the environment and starting the `tuxclient.exe` executable program. The following steps are for a Windows environment:

1. Add the `ejb_over_iiop.jar` file to your WebLogic Server `CLASSPATH`. For example:

   ```
   prompt> set CLASSPATH=%CLASSPATH%;%WL_HOME%\config\
           examples\applications\ejb_over_iiop.jar
   ```

2. Ensure that you have obtained the WebLogic Server Inter-operable Object Reference (IOR) by running the `host2ior` utility. The `ior.txt` file must be accessible to the `tuxclient.exe` CORBA C++ client program. You create this file once for a specific WebLogic Server installation. For example:

   ```
   prompt> java utils.host2ior hostname port
   ```

3. Start the `tuxclient.exe` CORBA client program by invoking the executable client application you created when you built the example.

   ```
   prompt> tuxclient.exe
   ```

4. Observe CORBA client output messages similar to the following:

   ```
   Creating a trader

   Buying 3000 shares of BEAS
   500 shares bought
   Buying 100 shares of PSFT
   100 shares bought

   End statelessSession.Client
   ```

Additional Sources of Information

The process of developing a BEA Tuxedo CORBA client application to interact with a WebLogic Server application begins with an EJB. If you are creating a new WebLogic Server application, the first step is to design and implement the server.

The following sections list sources of information about the steps required to implement connectivity between an RMI over IIOP CORBA Client and an EJB in WebLogic Server.

WebLogic Server

These topics are included in the WebLogic Server online documentation:

- WebLogic RMI over IIOP in the Developer Guide, *Using WebLogic RMI over IIOP.*
- WebLogic EJB in the Developer Guide, *BEA WebLogic Server Enterprise JavaBeans.*
- WebLogic RMI in the Developer Guide, *Using WebLogic RMI.*

Object Management Group (OMG)

Information about CORBA objects and value types, which are objects passable by value, is available from the OMG Web site: `http://www.omg.org.`

BEA Tuxedo CORBA

The BEA Tuxedo online documentation provides complete documentation on CORBA topics in the BEA Tuxedo product. The references listed here contain information on creating CORBA clients, the COSNaming Service, the IDL command, and BEA Tuxedo support for value types:

- *Creating CORBA Client Applications*
- *Using the CORBA Name Service*
- *BEA Tuxedo Command Reference*
- *CORBA Programming Reference*

CHAPTER

3 Connectivity from WebLogic Server to a CORBA Object

This topic includes the following sections:

- WLEC EJB simpapp Sample Application
- Building and Running the WLEC EJB simpapp Sample Application

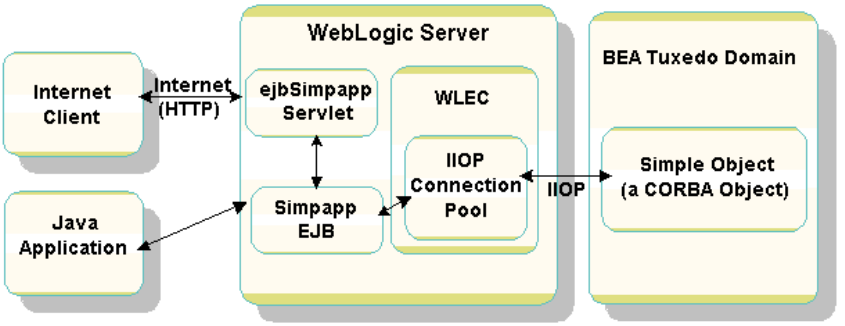
WLEC EJB simpapp Sample Application

The WebLogic Server WebLogic Enterprise Connectivity (WLEC) EJB simpapp sample application demonstrates how to use WebLogic Enterprise Connectivity (WLEC) to access a BEA Tuxedo CORBA object from a stateless EJB on WebLogic Server.

Note: Information for building and running this sample application is in WebLogic Server online documentation. This section provides supplementary information, and uses BEA Tuxedo 8.0 terminology. The information in WebLogic Server 6.0 documentation uses WebLogic Enterprise 5.1 terminology.

Figure 3-1 illustrates the connectivity provided in the WLEC EJB simpapp sample application.

Figure 3-1 WebLogic Server EJB to BEA Tuxedo CORBA Object



This example combines a WebLogic Server Enterprise JavaBean and the simple object from the BEA Tuxedo simpapp sample application. There are two ways to run this example:

- From a Java application
- From an Internet client and servlet

Building and Running the WLEC EJB simpapp Sample Application

The WebLogic Server code sample files and instructions for building and running the the WLEC EJB simpapp sample application are in the WebLogic Server installation location, in the *wlserver6*\samples\examples\wlec\ejb\simpapp directory. The information in the following sections supplement the WebLogic Server documentation for this sample.

Description

This WebLogic Server file explains how to build and run the sample application: *wlserver6*\samples\examples\wlec\ejb\simpapp\package-summary.html

At startup, WLS creates a WLEC connection pool for simpapp Domain, a BEA Tuxedo domain. At run time, the WebLogic Server simpapp EJB acts as a remote BEA Tuxedo CORBA client.

Prerequisites

The software products listed are prerequisites for running this sample application. For version information, see *Installing the BEA Tuxedo System*. Install and set up the following:

- WebLogic Server (WLS)
- Java Software Development Kit (SDK)
- BEA Tuxedo CORBA

Build the Example

To build and run the BEA Tuxedo CORBA simpapp sample, go to the Samples page in the BEA Tuxedo online documentation, and select the simpapp sample application.

Note: There is no longer a requirement to add the `RemoteObjectReference` class to the `CLASSPATH` in your WebLogic Server start script. The BEA Tuxedo 8.0 release does not include a `wlej2eecl.jar` file.

Index