BEA WebLogic Server™

Using WebLogic Enterprise Connectivity

Release 7.0
Document Revised: June 28, 2002

Using WebLogic Enterprise Connectivity

Part Number	Document Revised	Software Version
N/A	June 28, 2002	BEA WebLogic Server Version 7.0

Contents

About This Document

This document provides details about how to use the WebLogic Enterprise Connectivity (WLEC) component of the BEA WebLogic Server™ product.This product is deprecated and is nolonger supported.

This document is organized as follows:

- Chapter 1, "Introducing WebLogic Enterprise Connectivity," presents an overview of the CORBA environment in the BEA Tuxedo™ product and the WLEC component. A description of how the CORBA environment and the WLEC component interact is also included.
- Chapter 2, "Writing WebLogic Server Clients That Invoke CORBA Objects," describes the steps required to access BEA Tuxedo CORBA objects from WebLogic Server.

What You Need to Know

This document is intended for programmers who want to invoke BEA Tuxedo CORBA objects from a WebLogic Server client (servlet, EJB, JSP, or RMI object). The WLEC component provides a mechanism for propagating the security context established in WebLogic Server to a BEA Tuxedo domain.

e-docs Web Site

The BEA WebLogic Server product documentation is available on the BEA corporate Web site. From the BEA Home page, click the Product Documentation.

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 Server 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 Server 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/.

Contact Us!

Your feedback on the BEA WebLogic Server 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 Server documentation.

In your e-mail message, the software name and version you are using, as well as the title and documentation date of your documentation.

If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, 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.

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

Convention	Item

`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`*`]...`

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

Convention	Item
`...`	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.

1 Introducing WebLogic Enterprise Connectivity

The following sections introduce WebLogic Enterprise Connectivity including:

Deprecation Of WLEC

WebLogic Enterprise Connectivity (WLEC) is deprecated in this release of WebLogic Server 7.0. Tuxedo Corba applications using WLEC should migrate to WebLogic Tuxedo Connector. For more information, see WebLogic Tuxedo Connector at http://e-docs.bea.com/wls/docs70/wtc.html.

The BEA Tuxedo CORBA Environment

The BEA Tuxedo product enables you to build, deploy, and manage component-based solutions for your enterprise. The BEA Tuxedo product combines the Object Request Broker (ORB) in the Common Object Request Broker Architecture (CORBA) and online transaction processing (OLTP) functions available in Application-to-Transaction Monitor Interface (ATMI). The result is a platform that enables you to deliver scalable, secure, and transactional e-commerce applications in a well-managed environment.

The following sections introduce some of the BEA Tuxedo CORBA terminology.

CORBA Objects in the BEA Tuxedo Product

CORBA is a language-independent specification that promotes an object-oriented approach to building and integrating distributed software applications. You can implement your business logic as CORBA objects in different languages. The CORBA environment in the BEA Tuxedo supports C++, Automation, and Java language bindings. For example, a banking application can have objects for customer accounts. These customer account objects can have operations for depositing, withdrawing, and viewing the account balances.

You use the Object Management Group (OMG) interface definition language (IDL) to describe CORBA objects. You write an OMG IDL description of your CORBA objects and run the OMG IDL through a compiler. The compiler generates stubs and skeletons.

A CORBA factory is an object that provides an operation for creating object references to another CORBA object. A server application uses CORBA factories to let client applications access CORBA objects that are implemented in the server application. When a client application needs to get a reference to a CORBA object that is managed by a server application, it typically gets that object reference from a CORBA factory.

An ORB is a communications bus that enables client applications to communicate with distributed objects that are managed by server applications. In a CORBA environment, applications do not need to include network and operating system information to communicate. Instead, heterogeneous client and server applications communicate with the ORB. The ORB delivers client requests to the appropriate server applications and returns the server responses to the requesting client application.

Domains, Transactions, and Transaction Contexts

A BEA Tuxedo domain is a group of objects, services, machines, and resources that you administer as a unit. You can set up domains based on characteristics such as application functions, security needs, or geographical locations. For example, one domain might consist of the objects, services, machines, and resources for a bank's customer accounts. The bank might have a separate domain for its employee and payroll resources.

A transaction is a set of operations based on business rules. The operations act as one logical unit, even if they are distributed geographically. By acting as one unit, either all the transaction's operations complete successfully (in which case the transaction completes successfully) or all the operations roll back (the transaction fails). For example, a transaction withdraws money from one customer's account and deposits it into another customer's account. This transaction consists of two operations. Both operations must succeed in order for the transaction to succeed.

A transaction context defines the scope of a transaction, and is shared by the objects that are participating in the transaction. A transaction context can consist of various types of data, such as local variables, locks, cursor positions, and file control blocks.

Environmental Objects

The CORBA environment in the BEA Tuxedo product provides the following environmental objects that enable client applications to use CORBA services:

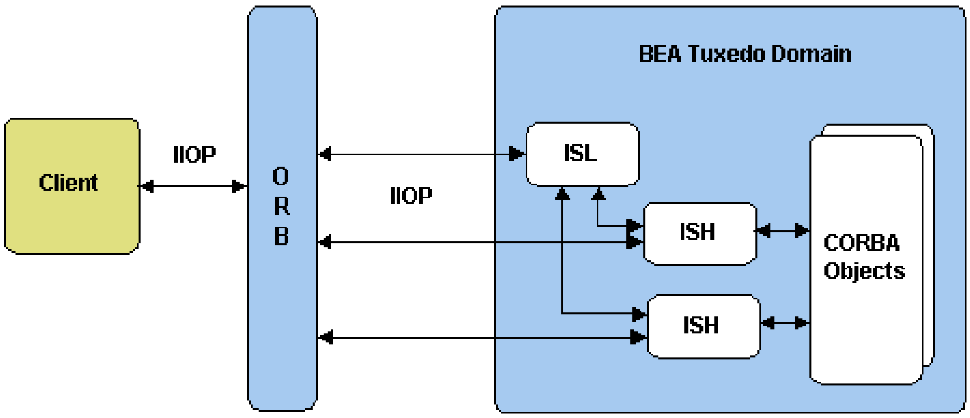
- The Bootstrap object establishes communication between a client application and a BEA Tuxedo domain. It also obtains object references to the other environmental objects in the BEA Tuxedo domain.
- The FactoryFinder object enables a client application to find CORBA factories. A CORBA factory can create object references for CORBA objects. The factories available to client applications are those that register with the FactoryFinder object at startup.
- The TransactionCurrent object enables a client application to manage a transaction. The TransactionCurrent object is the BEA Tuxedo implementation of the CORBA Object Transaction Service (OTS), which supports multiple transaction models. Some operations that the TransactionCurrent object provides are `begin()`, `commit()`, `rollback()`, `suspend()`, `resume()`, and `get_status()`.
- The UserTransaction object enables a client application to participate in a transaction. This environmental object is an implementation of the Sun Microsystems, Inc. Java Transaction Application (JTA) Programming Interface. The UserTransaction object is available for CORBA Java objects.

IIOP and the IIOP Listener/Handler

Internet Inter-ORB Protocol (IIOP) is the standard protocol defined by the CORBA specification for interoperation among ORBs. In the BEA Tuxedo CORBA environment, the IIOP Listener/Handler handles communication between the ORB and CORBA client applications. The IIOP Listener (ISL) manages incoming communications for remote CORBA clients. Each ISL has one or more IIOP Handlers (ISHs) associated with it. An ISL assigns client applications to ISHs and balances the incoming client loads across the ISHs. An ISH is a communications link between a client application and a CORBA object. Each BEA Tuxedo domain that supports remote clients has at least one ISL.

Figure 1-1 presents an overview of IIOP.

Figure 1-1 How IIOP Works in the BEA Tuxedo System



More Information About the BEA Tuxedo CORBA Environment

For more information about the CORBA environment in the BEA Tuxedo product, see the BEA Tuxedo documentation. If you are new to the BEA Tuxedo product, we recommend that you read the BEA Tuxedo Product Overview.

What Is WebLogic Enterprise Connectivity?

WebLogic Enterprise Connectivity is a component of WebLogic Server that enables you to use WebLogic Enterprise Connectivity connection pools (referred to as WLEC connection pools in the Administration Console) to call BEA Tuxedo CORBA objects from WebLogic Server clients (servlets, EJBs, JSPs, and RMI objects).

Key Features of WebLogic Enterprise Connectivity

The key features of the WebLogic Enterprise Connectivity component are:

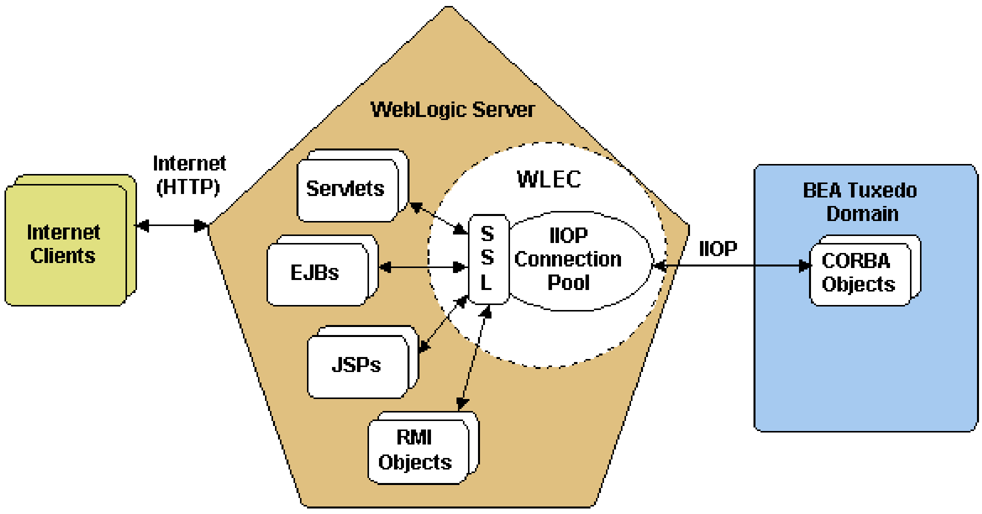
- Pooled WebLogic Enterprise Connectivity connections to a BEA Tuxedo CORBA application
- Multiple active CORBA client transactions from a single WebLogic Server process
- Configuration of WLEC connection pools through the WebLogic Server Administration Console

- Monitoring of WLEC connection pools through the WebLogic Server Administration Console
- Support for the Secure Sockets Layer (SSL) protocol
- Security context propagation from WebLogic Server to a BEA Tuxedo domain
- Pool reinitialization at run time

WebLogic Enterprise Connectivity System Architecture

Figure 1-2 illustrates the WebLogic Enterprise Connectivity system architecture and its relationship to WebLogic Server and BEA Tuxedo.

Figure 1-2 The WebLogic Enterprise Connectivity System Architecture



For each BEA Tuxedo domain, WebLogic Server creates a WLEC connection pool which is configured in through the Administration Console in WebLogic Server. WebLogic Server clients use the WLEC connection pool to access CORBA objects in the BEA Tuxedo domain which can be on a remote machine and/or behind a firewall.

Clients and Servers in the WebLogic Enterprise Connectivity Component

Internet clients are served by applications on WebLogic Server. These applications act as clients to BEA Tuxedo domains by way of the WebLogic Enterprise Connectivity component. BEA Tuxedo domains provide the requested CORBA objects and send results back to WebLogic Server clients. WebLogic Server clients can then process the results, do some other work, and send the results to the Internet clients.

WLEC Connection Pools

The WebLogic Enterprise Connectivity component uses WLEC connection pools to enable WebLogic Server clients to connect to BEA Tuxedo domains. A WLEC connection pool is a set of IIOP connections to a BEA Tuxedo domain. WebLogic Server creates the WLEC connection pools at startup and assigns connections to WebLogic Server clients as needed. WLEC connection pools are efficient because they let a limited number of connections serve many users. Because the overhead for creating connections is performed at startup, WebLogic Servers can access CORBA objects and operations quickly.

WebLogic Enterprise Connectivity connection pooling has the following features:

- Uses IIOP.
- Supports one WLEC connection pool for each BEA Tuxedo domain.
- Allows WebLogic Server to have multiple simultaneous active BEA Tuxedo transaction contexts. However, a thread in WebLogic Server can have only one

active client transaction context at a time. The WebLogic Enterprise Connectivity component does not support nested transactions.

- Supports multiple concurrent requests on the same physical connection, each with its own security context.
- Allows you to reinitialize WLEC connection pools at run time.

BEA Tuxedo Security Features Supported by the WebLogic Enterprise Connectivity Component

The WebLogic Enterprise Connectivity component provides the following BEA Tuxedo security features:

- Authentication

  Authentication ensures that two communicating parties are authorized to communicate with each other. The BEA Tuxedo system supports password and certificate authentication.

- Confidentiality

  Confidentiality is the ability to keep communications secret from parties other than the intended recipient. It is achieved by encrypting all data.

- Integrity

  Integrity is a guarantee that the data being transferred has not been modified in transit.

- The SSL protocol

  The SSL protocol establishes secure communications between client and server applications. SSL is provided for IIOP requests to CORBA objects.

Note: The WebLogic Enterprise Connectivity component does not support the CORBA security application programming interface (API) in BEA Tuxedo.

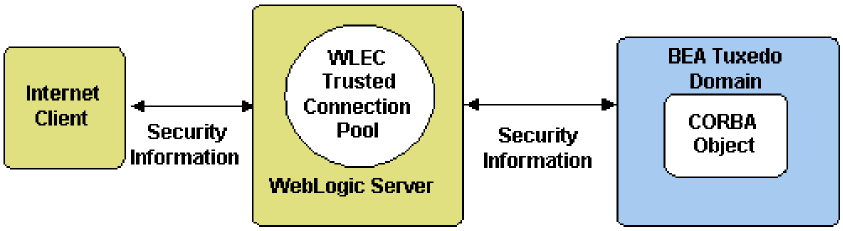
Security Context Propagation

Using WebLogic Enterprise Connectivity, a security context established in WebLogic Server can be used to establish a security identify in a BEA Tuxedo domain. This passing of security credentials is referred to as security context propagation.

Security context propagation requires a WLEC connection pool. In a WLEC connection pool, each network connection has been authenticated through a User identity that is defined by the system administrator of WebLogic Server. You can use either password or certificate authentication to establish a WLEC connection pool.

As Figure 1-3 shows, a WLEC connection pool enables you to propagate security information from a client through WebLogic Server to the CORBA environment in the BEA Tuxedo product.

Figure 1-3 Security Context Propagation



Connection Failure Handling

The WebLogic Enterprise Connectivity component provides connection failure handling by using two lists of ISL addresses for each WLEC connection pool: a primary list and a failover list. The WebLogic Enterprise Connectivity component provides connection failure handling in the following cases:

- When WebLogic Server is booted.

If no ISL defined in the primary address list is accessible at server startup, the WebLogic Enterprise Connectivity component uses ISL addresses from the failover list.

- When an WLEC connection pool loses an active connection.

  When the WLEC connection pool loses a connection, WebLogic Server tries to reconnect by using other addresses from the primary address list. If all addresses in the primary list fail, WebLogic Server tries to reconnect using addresses from the failover list. Lost connections are restarted only when they are needed. If the current load on the WLEC connection pool does not require a lost connection to be reopened, it stays disconnected and other active connections are used instead.

CHAPTER

2 Writing WebLogic Server Clients That Invoke CORBA Objects

The following sections describe the requirements to access BEA Tuxedo CORBA objects from WebLogic Server including:

- Before You Begin
- Configuring a WLEC Connection Pool
- Accessing BEA Tuxedo CORBA Objects
- More Information About Transactions

Before You Begin

Before you implement WebLogic Server clients that invoke BEA Tuxedo CORBA objects, you need to:

- Configure and run WebLogic Server
- Configure and run a BEA Tuxedo CORBA server application
- Be familiar with the WebLogic Enterprise Connectivity architecture as described in Chapter 1, "Introducing WebLogic Enterprise Connectivity."

- Run the WebLogic Enterprise Connectivity examples in the `/samples/examples/wlec` directory in the WebLogic Server installation.

Configuring a WLEC Connection Pool

Configure a WLEC connection pool for each BEA Tuxedo domain that contains a BEA Tuxedo CORBA object you want to access from a WebLogic Server client. Because the security context established for the WebLogic Server client is propagated to the BEA Tuxedo domain through the WLEC connection pool, the WLEC connection pools are the basis for the security context propagation feature.

Before using WLEC connection pools, add *`WL_HOME`*`/lib/wleorb.jar`, *`WL_HOME`*`/lib/wlepool.jar`, and *`TUXDIR/udataobj/java/jdk/wleclient.jar`* to the `CLASSPATH` variable in the `startAdminWebLogic.sh` or `startAdminWebLogic.cmd` file. If you are using a JSP that requests an EJB served by WLE 5.1, add the location of the WLE 5.1 *`wlej2eecl.jar`* file to the `CLASSPATH` variable in the `startAdminWebLogic.sh` or `startAdminWebLogic.cmd` file.

Create a new WLEC connection pool for the purpose of security context propagation. To create a WLEC connection pool:

1. Go to the Services→WLEC node in the left pane of the Administration Console. In the right pane of the Administration Console, click the Create a new WLEC Connection Pool link. Define the attributes listed in the following table.

Table 2-1 WLEC Connection Pool Attributes on the General Tab

Attribute	Description
Name	The name of the WLEC connection pool. The name must be unique for each WLEC connection pool.

Table 2-1 WLEC Connection Pool Attributes on the General Tab (Continued)

Attribute	Description
Primary Addresses	A list of addresses for IIOP Listener/Handlers that can be used to establish a connection between the WLEC connection pool and the BEA Tuxedo domain. The format of each address is `//`*`hostname`*`:`*`port`*. The addresses must match the ISL addresses defined in the `UBBCONFIG` file. Multiple addresses are separated by commas. For example: `//main1.com:1024, //main2.com:1044`. To configure the WLEC connection pool to use the SSL protocol, use the `corbalocs` prefix with the address of the IIOP Listener/Handler. For example: `corbalocs://`*`hostname`*`:`*`port`*.
Failover Addresses	A list of addresses for IIOP Listener/Handlers that are used if connections cannot be established with the addresses defined in the Primary Addresses attribute. Multiple addresses are separated by commas. This attribute is optional.
Domain	The name of the BEA Tuxedo domain to which this WLEC connection pool connects. You can have only one WLEC connection pool per BEA Tuxedo domain. The domain name must match the `domainid` parameter in the `RESOURCES` section of the `UBBCONFIG` file for the BEA Tuxedo domain.
Minimum Pool Size	The number of IIOP connections to be added to the WLEC connection pool when WebLogic Server starts. The default is 1.
Maximum Pool Size	The maximum number of IIOP connections that can be made from the WLEC connection pool. The default is 1.

2. Click the Create button.

3. Propagate the security context for a User in a WebLogic Server security realm to a BEA Tuxedo domain. To do so, define the attributes on the Security tab in the Connection Pool Configuration window. The following table describes these attributes.

Table 2-2 WLEC Connection Pool Attributes on the Security Tab

Attribute	Description
User Name	A BEA Tuxedo user name. This attribute is required only when the security level in the BEA Tuxedo domain is `USER_AUTH`, `ACL` or `MANDATORY_ACL`.
User Password	The password for the User defined in the `User Name` attribute. This attribute is required only when you define the `User Name` attribute.
User Role	The BEA Tuxedo user role. This attribute is required when the security level in the BEA Tuxedo is `APP_PW`, `USER_AUTH`, `ACL`, or `MANDATORY_ACL.`
Application Password	The password for the BEA Tuxedo CORBA application. This attribute is required when the security level in the BEA Tuxedo domain is `APP_PW`, `USER_AUTH`, `ACL`, or `MANDATORY_ACL`.
Minimum Encryption Level	The minimum SSL encryption level used between the BEA Tuxedo domain and WebLogic Server. The possible values are 0, 40, 56, and 128. The default is 40. Zero (0) indicates that the data is signed but not sealed. 40, 56, and 128 specify the length, in bits, of the encryption key. If this minimum level of encryption is not met, the SSL connection between the BEA Tuxedo domain and WebLogic Server fails.
Maximum Encryption Level	The maximum SSL encryption level used between the BEA Tuxedo domain and WebLogic Server. The possible values are 0, 40, 56, and 128. The default is the maximum level allowed by the Encryption Package kit license. Zero (0) indicates that the data is signed but not sealed. 40, 56, and 128 specify the length, in bits, of the encryption key. If this minimum level of encryption is not met, the SSL connection between the BEA Tuxedo domain and WebLogic Server fails.

Table 2-2 WLEC Connection Pool Attributes on the Security Tab (Continued)

Attribute	Description
Enable Certificate Authentication	Checkbox that enables the use of certificate authentication. By default, certificate authentication is disabled.
Enable Security Context	Check this check box to pass the security context of the WebLogic Server User passed to the BEA Tuxedo domain. By default, security context is disabled.

4. To save your changes, click the Apply button.
5. Click the Targets tab and select a server. Click Apply.
6. Reboot your WebLogic Server.
7. Run the `tpusradd` command to define the WebLogic Server User as an authorized User in the BEA Tuxedo domain.
8. Set the `-E` option of the `ISL` command to configure the IIOP Listener/Handler to detect and utilize the propagated security context from the WebLogic Server realm. The `-E` option of the `ISL` command requires you to specify a principal name. The principal name defines the principal used by the WLEC connection pool to log in to the BEA Tuxedo domain. The principal name should match the name defined in the User Name attribute when creating a WLEC connection pool.

Using certificate authentication between the WebLogic Server environment and the BEA Tuxedo CORBA environment implies performing a new SSL handshake when establishing a connection from the WebLogic Server environment to a CORBA object. To support multiple client requests over the same SSL network connection, you must set up certificate authentication so that it operates as follows:

1. Obtain a digital certificate for the principal and put the private key in the `TUXDIR/udataobj/security/keys` directory of BEA Tuxedo.
2. Use the `tpusradd` command to define the principal as a BEA Tuxedo user.
3. Define the IIOP Listener/Handler in the `UBBCONFIG` file with the `-E` option to indicate the principal is to be used for authentication.

4. Define the principal name in the User Name attribute when creating a WLEC Connection pool in the Administration Console of WebLogic Server.
5. Obtain a digital certificate for the IIOP Listener/Handler.
6. Specify the digital certificate in the `SEC_PRINCIPAL_NAME` option of the ISL command and use the `-S` option to indicate that a secure port should be used for communication between the BEA Tuxedo domain and the WebLogic Server security realm.

For more information about configuring security in BEA Tuxedo CORBA applications, see *Using Security in CORBA Applications*.

Accessing BEA Tuxedo CORBA Objects

This section describes accessing a BEA Tuxedo CORBA object from a WebLogic Server client.

Step 1. Create Client Stubs

Client stubs provide the programming interface for operations of a BEA Tuxedo CORBA object. To create client stubs, compile the Object Management Group (OMG) Interface Definition Language (IDL) file for the BEA Tuxedo CORBA object you want to access from the WebLogic Server client. To create client stubs for a BEA Tuxedo CORBA object, use the `idl` compiler that is included in the BEA Tuxedo software.

Make sure the WebLogic Server `CLASSPATH` environment variable includes the directory that contains the client stubs for the BEA Tuxedo CORBA object.

Step 2. Import Java Packages

Import the following Java packages into your WebLogic Server client:

- `org.omg.CORBA.*`

- `com.beasys.Tobj.*`
- `com.beasys.*`

Step 3. Connect the WebLogic Server Client to a BEA Tuxedo Domain

Each WLEC connection pool has a `Tobj_Bootstrap` object that lets you access the associated BEA Tuxedo domain. The WebLogic Enterprise Connectivity component provides an object called BootstrapFactory which provides access to the `Tobj_Bootstrap` object for a particular BEA Tuxedo domain. Include the following code in your WebLogic Server client to connect to a BEA Tuxedo domain:

```
Tobj_Bootstrap myBootstrap =
Tobj_BootstrapFactory.getClientContext("myPool");
```

where

- The `getClientContext()` method returns the `Tobj_Bootstrap` object that is associated with *`myPool`*. If `getClientContext()` cannot find a WLEC connection pool with this name, it returns `null`.
- *`myPool`* is the name of a WLEC connection pool for the desired BEA Tuxedo domain. This WLEC connection pool needs to be defined in the Administration Console.

Step 4. Get an Object Reference for the BEA Tuxedo CORBA Object

Use the FactoryFinder object in your WebLogic Server client to get a reference to the BEA Tuxedo CORBA object.

1. Get the FactoryFinder object as follows:

```
org.omg.CORBA.Object myFFObject =
    myBootstrap.resolve_initial_references("FactoryFinder");
FactoryFinder myFactFinder =
    FactoryFinderHelper.narrow(myFFObject);
```

where

- `myBootstrap` is the `Tobj_Bootstrap` object for the desired BEA Tuxedo domain.
- The `resolve_initial_references()` method returns the object reference for the FactoryFinder object.
- The `FactoryFinderHelper` interface provides auxiliary functionality for the `FactoryFinder` interface, notably the `narrow()` method.
- The `narrow()` method casts the object reference to point to a FactoryFinder object.

2. Get the factory for the desired BEA Tuxedo CORBA object as follows:

```
org.omg.CORBA.Object myFactoryRef =
    myFactFinder.find_one_factory_by_id(myFactoryHelper.id());
myFactory =
    myFactoryHelper.narrow(myFactoryRef);
```

where

- *`myFactFinder`* is the FactoryFinder object.
- The `find_one_factory_by_id()` method finds and returns a factory object reference based on an ID number.
- The *`myFactoryHelper`* interface provides auxiliary functionality for the *`myFactory`* interface, notably the `narrow()` method.
- The `narrow()` method casts the object reference to point to the object factory.

3. Get the BEA Tuxedo CORBA object using the object's `find()` method. For example, if you are accessing an object named `Simple`, use the following code:

```
Simple mySimple = mySimpleFactory.find_simple();
```

where the factory provides the `find_simple()` method for finding the `Simple` object.

For information about the FactoryFinder object, see the *CORBA C++ Programming Reference* in the BEA Tuxedo documentation.

Step 5. Start a Transaction (Optional)

You can access BEA Tuxedo CORBA objects within the scope of a transaction. The following sample code uses the TransactionCurrent object to access a BEA Tuxedo CORBA object within a scope of a transaction. You can also use the UserTransaction object when accessing a BEA Tuxedo CORBA object.

To start a transaction, include the following code in your WebLogic Server client:

1. Get the TransactionCurrent object as follows:

   ```
   org.omg.CORBA.Object myTCObject =
     myBootstrap.resolve_initial_references("TransactionCurrent");
   CosTransactions.Current myTransaction =
       CosTransactions.CurrentHelper.narrow(myTCObject);
   ```

   where

   - *`myBootstrap`* is the Bootstrap object for the BEA Tuxedo domain associated with the WLEC connection pool.
   - The `resolve_initial_references()` method returns the object reference for the TransactionCurrent object.
   - The `CosTransactions.Current` interface defines the interface for the TransactionCurrent object. It lets you explicitly manage the associations between threads and transactions.
   - The `CurrentHelper` interface provides auxiliary functionality for the `Current` interface, notably the `narrow()` method.
   - The `narrow()` method casts the object reference to point to a `CosTransactions` object.

2. Begin a transaction as follows:

   ```
   myTransaction.begin();
   ```

   where the `begin()` method creates a transaction context and associates it with *`myTCObject`* in step 1. Because *`myTCObject`* is associated with *`myBootstrap`* and *`myBootstrap`* is associated with a specific WLEC connection pool, *`myTransaction`* is associated with a specific WLEC connection pool.

Step 6. Access the BEA Tuxedo CORBA Object and Its Operations

Call methods on the BEA Tuxedo CORBA object in the BEA Tuxedo domain associated with the WLEC connection pool.

If you are accessing objects within a transaction context, you can use the following TransactionCurrent methods to manipulate and query the transaction context:

- `suspend()`
- `resume()`
- `rollback_only()`
- `get_status()`
- `get_transaction_name()`
- `set_timeout()`
- `get_control()`

Step 7. End the Transaction (Optional)

If you accessed the BEA Tuxedo CORBA object within a transaction context, use one of the following TransactionCurrent methods to end the transaction:

- `commit()`
- `rollback()`

More Information About Transactions

This section includes additional information about transactions in the BEA Tuxedo environment.

Multithreading

The BEA Tuxedo CORBA Transaction service enables multiple threads of a single application process to start separate transactions simultaneously. For example, if two threads simultaneously call `CosTransactions.Current.begin()` or `UserTransaction.begin()` both threads have separate transaction contexts that correspond to separate transactions.

The CORBA Transaction Service does not let multiple threads of a single application work with the same transaction at the same time. If you use CosTransactions, you can suspend and resume a transaction in order to use the transaction in multiple threads:

1. In the first thread, call `Current.suspend()` to suspend the transaction and to obtain a Control object.
2. In the second thread, call `Current.resume()` for the Control object in order to resume the transaction.

If a thread tries to resume a transaction that has not been suspended, the BEA Tuxedo system throws an `INVALID_CONTROL` exception.

The `UserTransaction` interface does not support the `suspend()` and `resume()` methods. Therefore, you cannot use a transaction in multiple threads when you use UserTransaction.

Multiple Active WLEC Connection Pools

The WebLogic Enterprise Connectivity component supports multiple simultaneously active WLEC connection pools in a single WebLogic Server client. When you call `CosTransactions.Current.begin()` or `UserTransaction.begin()` to create a transaction context, the BEA Tuxedo system associates the transaction with a WLEC connection pool. All calls made inside the scope of the transaction must be for objects that reside in the domain associated with the transaction's WLEC connection pool.

A transaction cannot span multiple BEA Tuxedo domains. If you try to make a call for an object in a different domain, the BEA Tuxedo system throws an `INVALID_TRANSACTION` exception.

Relationship Between Active Transactions and Connections

When a WebLogic Server client starts or resumes a transaction, the WLEC connection pool infrastructure reserves a connection for requests that are sent in the context of the transaction. The WebLogic Enterprise Connectivity component does not use this connection to send requests that are not in the transaction context. The WebLogic Enterprise Connectivity component reserves the connection until the transaction is committed, rolled back, or suspended.

Note: The `suspend()` and `resume()` are available only for CosTransactions.

The number of concurrently active transactions is bound by the number of available connections in the pool. If a connection is not available when a thread begins or resumes a transaction, the WebLogic Enterprise Connectivity component throws a `NO_RESOURCES` exception.

Transaction Management

Each thread has its own transaction context. When a thread starts or resumes a transaction, the transaction is active until it is committed, rolled back, or suspended. There is no guarantee that subsequent invocations to a WebLogic Server client get executed in the same thread. Therefore, it is important for a thread to commit, roll back, or suspend a transaction before ending a WebLogic Server client invocation.

Note: The `suspend()` and `resume()` are available only for CosTransactions.

If necessary, you can use a transaction in multiple WebLogic Server client invocations as follows:

1. At the end of each invocation, suspend the transaction.
2. In the next invocation, resume the transaction.

Be careful when using this solution, because a transaction can time out before you make the next WebLogic Server client invocation.