BEA WebLogic Server

Using WebLogic Connectivity

BEA WebLogic Server Version 6.0
Document Date: March 3, 2001

Using WebLogic Enterprise Connectivity

Document Edition	Date	Software Version
N/A	March 3, 2001	BEA WebLogic Server Version 6.0

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 document covers the following topics:

- Chapter 1, “Introducing WebLogic Enterprise Connectivity,” presents an overview of the WebLogic Enterprise (WLE) system and the WLEC component. A description of how the WLE system and the WLEC component interact is also included.
- Chapter 2, “Writing WebLogic Server Clients That Invoke WLE Objects,” describes the steps required to access WLE objects from a WebLogic Server client.

What You Need to Know

This document is intended for programmers who want to invoke WLE objects (CORBA objects, EJBs, and RMI 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 WLE 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 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 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, please indicate that you are using the documentation for the BEA WebLogic Server 6.0 release.

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 file names 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

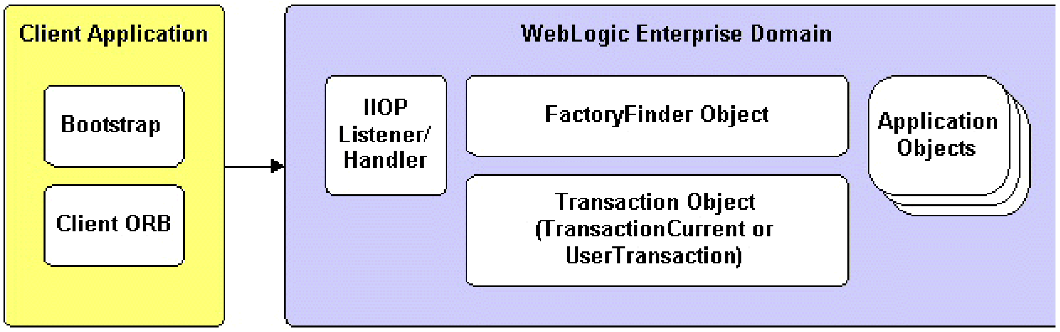
This section includes the following topics:

What Is the WebLogic Enterprise System?

The BEA WebLogic Enterprise™ (WLE) product enables you to build, deploy, and manage component-based solutions for your enterprise. The WLE product combines Object Request Broker (ORB) and online transaction processing (OLTP) functions with industry programming models such as Enterprise JavaBeans (EJBs), Common Object Request Broker Architecture (CORBA), and 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.

Figure 1-1 presents an overview of the WLE system.

Figure 1-1 The WLE System



The following topics introduce some of the WLE system terminology.

CORBA and J2EE Development

The WLE product supports two types of objects that you can access from WebLogic Server clients:

- CORBA objects
- J2EE objects—EJBs and remote method invocation (RMI) objects

You can access both types of objects from one WebLogic Server client. The steps for accessing each type of object are different.

CORBA Objects, IDL, Factories, and ORBs

Common Object Request Broker Architecture (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 WLE system supports C++ 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.

The Object Management Group (OMG) interface definition language (IDL) is a language that you use to describe CORBA objects. You write an IDL description of the objects that will be visible remotely, then run this 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.

J2EE, RMI, EJBs, and JNDI

J2EE is the Java 2 (previously called JDK 1.2) platform that provides a complete range of enterprise-class functionality for server-side computing. It is designed to provide an integrated application environment in Java for building enterprise-level n-tier applications.

Remote method invocation (RMI) is a Java-only version of CORBA. Instead of writing IDL to describe an object (like you do with CORBA), you run a program called `rmic` on your Java `.class` files. The `rmic` program generates stub and skeleton classes directly from the `.class` files.

Enterprise JavaBean (EJB) is an API specification for building scalable, distributed, component-based, multitiered applications. EJBs leverage and extend the JavaBeans component model to provide a rich, object-oriented transactional environment for developers creating enterprise applications.

The Java Naming and Directory Interface (JNDI) provides access to an EJB's home interface. The home interface provides access to the EJB.

Domains, Transactions, and Transaction Contexts

A WLE 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 WLE system provides the following environmental objects that enable client applications to use WLE system services:

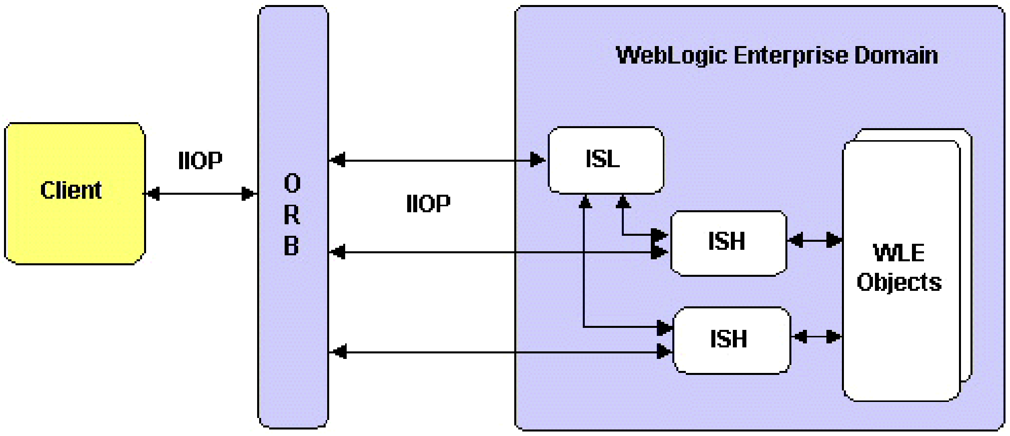
- The Bootstrap object establishes communication between a client application and a WLE domain. It also obtains object references to the other environmental objects in the WLE domain.
- The FactoryFinder object enables a client application to find CORBA factories. A CORBA factory can create object references 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 WLE transaction. The TransactionCurrent object is the WLE 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 TransactionCurrent object is available only for CORBA objects, not for EJBs or RMI objects.
- 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 objects, RMI objects, and bean-managed EJBs.

IIOP, ISLs, ISHs, and RMI on IIOP

Internet Inter-ORB Protocol (IIOP) is the standard protocol defined by the CORBA specification for interoperation among ORBs. A WLE 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 WLE object. Each WLE domain that supports remote clients has at least one ISL.

Figure 1-2 presents an overview of IIOP.

Figure 1-2 How IIOP Worked in the WLE System



RMI is a Java API for distributed object computing and Web connectivity. A client uses RMI to get a reference to an object that exists elsewhere on the network and invoke methods on the object as though it existed locally. RMI on IIOP is the WebLogic version of RMI implemented to use the CORBA IIOP protocol. RMI on IIOP enables clients of WLE domains to use JNDI to access EJBs and RMI objects.

For More Information About WLE

For more information about the WLE system, see the WLE documentation. If you are new to the WLE system, we recommend that you read Getting Started.

What Is WebLogic Enterprise Connectivity?

WebLogic Enterprise Connectivity (WLEC) is a component of WebLogic Server™ that enables you to use WLEC connection pools to call WLE objects (CORBA objects, EJBs, and RMI objects) from WebLogic Server clients (servlets, EJBs, JSPs, and RMI objects).

The rest of this section covers the following topics:

- "Key Features of WebLogic Enterprise Connectivity"
- "System Architecture"
- "Clients and Servers in the WLEC Component"
- "WLEC Connection Pools"
- "WLE Security Features Supported by the WLEC Component"
- "Security Context Propagation"
- "Connection Failure Handling"

Key Features of WebLogic Enterprise Connectivity

The key features of the WLEC component are:

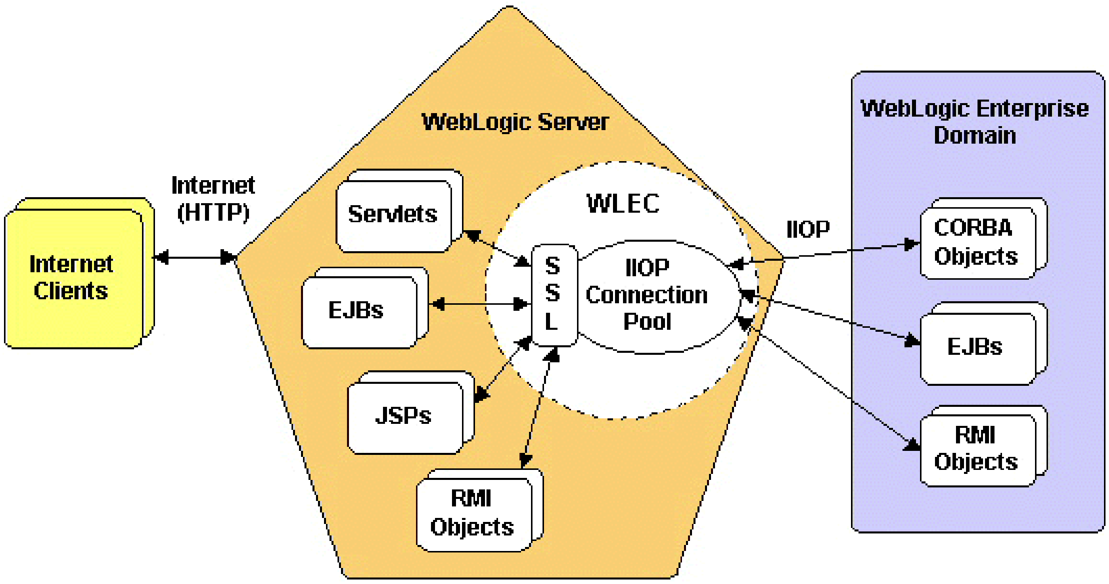
- Pooled WLEC connections to a WLE system
- Multiple active WLE client transactions from a single WebLogic Server process
- Configuration of WLEC connection pools through the Administration Console
- Monitoring of WLEC connection pools through the Administration Console
- Support for the Secure Sockets Layer (SSL) protocol

- Security context propagation from WebLogic Server to a WLE domain
- Pool re-initialization at run time

System Architecture

Figure 1-3 illustrates the WLEC system architecture and its relationship to WebLogic Server and WLE systems.

Figure 1-3 The WLEC System Architecture



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

Clients and Servers in the WLEC Component

Internet clients are served by applications on WebLogic Server. These applications act as clients to WLE domains by way of the WLEC component. WLE domains provide the requested WLE 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 WLEC component uses WLEC connection pools to enable WebLogic Server clients to connect to WLE domains. A WLEC connection pool is a set of IIOP connections to a WLE 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 WLE objects and operations quickly.

WLEC connection pooling has the following features:

- Uses IIOP.
- Supports one WLEC connection pool for each WLE domain.
- Allows WebLogic Server to have multiple simultaneous active WLE transaction contexts. However, a thread in WebLogic Server can have only one active WLE client transaction context at a time. The WLEC component does not support nested transactions.
- Supports multiple concurrent requests on the same physical connection, each with its own security context.
- Allows you to re-initialize WLEC connection pools at run time.

WLE Security Features Supported by the WLEC Component

The WLEC component provides the following WLE security features:

- Authentication

  Authentication ensures that two communicating parties are authorized to communicate with each other. The WLE system supports username/password authentication 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 and for RMI on IIOP requests to EJBs or RMI objects.

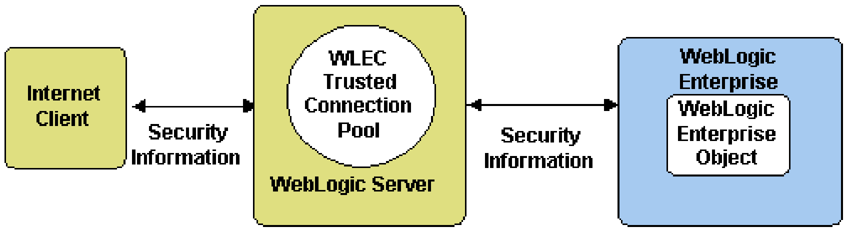
The WLEC component does not support the WLE security API. For an explanation of why this is so, see the "Security,"section in Chapter 2, "Writing WebLogic Server Clients That Invoke WLE Objects."

Security Context Propagation

Security context propagation requires a WLEC connection pool. A WLEC connection pool is a pool of IIOP connections in which each network connection has been authenticated using a particular principal identity that is pre-determined by the system administrator. You can use either password authentication or certificate authentication to establish a WLEC connection pool.

As Figure 1-4 shows, a WLEC connection pool enables you to propagate security information from a client through WebLogic Server to the WLE system:

Figure 1-4 Security Context Propagation



Connection Failure Handling

The WLEC 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 WLEC 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 WLEC 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, it 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 WLE Objects

This section contains the following topics:

Before You Begin

Before you implement WebLogic Server clients that invoke WebLogic Enterprise (WLE) objects, you need to:

- Configure and run WebLogic Server

- Configure and run a WLE server application
- Be familiar with the WebLogic Enterprise Connectivity (WLEC) 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.

Implementation Overview

To implement a WebLogic Server client that invokes a WLE object, write a WebLogic Server client (servlet, EJB, JSP, or RMI object) and include code for accessing existing WLE objects and operations. For more information, see the following:

- The WebLogic Programming Guides. These guides provide information about creating servlets, EJBs, JSPs, and RMI objects in the WebLogic Server environment.
- Configuring Security Context Propagation for information about configuring a WLEC connection pool.
- “Accessing WLE CORBA Objects” or “Accessing a WLE EJB or RMI Object” for information about the code you need to include in the WebLogic Server client in order to access the WLE object.

Configuring a WLEC Connection Pool

Configure a WLEC connection pool for each WLE domain that contains a WLE object you want to access from a WebLogic Server client. Because the security context established for the WebLogic Server client is propagated to the WLE domain through the WLEC connection pool, the WLEC connection pools are the basis for the security context propagation feature. For information about configuring WLEC connection pools, see Configuring Security Context Propagation.

Accessing WLE CORBA Objects

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

Step 1. Create Client Stubs

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

For more information, see *Using the idltojava Command* in the WLE documentation.

Make sure the WebLogic Server `CLASSPATH` environment variable includes the directory that contains the client stubs for the WLE 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 WLE Domain

Each WLEC connection pool has a `Tobj_Bootstrap` object that lets you access the associated WLE domain. The WLEC component provides an object called BootstrapFactory which provides access to the `Tobj_Bootstrap` object for a particular WLE domain. Include the following code in your WebLogic Server client to connect to a WLE 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 WLE domain. This WLEC connection pool needs to be defined in the Administration Console.

Step 4. Get an Object Reference for the WLE CORBA Object

Use the FactoryFinder object in your WebLogic Server client to get a reference to the WLE 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 WLE 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 WLE 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 WLE 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 WLE documentation.

Step 5. Start a Transaction (optional)

You can access WLE CORBA objects within the scope of a transaction. The following sample code uses the TransactionCurrent object to access a WLE CORBA object within a scope of a transaction. You can also use the UserTransaction object when accessing a WLE 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 WLE 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 WLE CORBA Object and Its Operations

Call methods on the WLE CORBA object in the WLE 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 WLE CORBA object within a transaction context, use one of the following TransactionCurrent methods to end the transaction:

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

Accessing a WLE EJB or RMI Object

This section describes the steps for accessing a WLE EJB or RMI object from a WebLogic Server client.

Step 1. Create Client Stubs

Client stubs provide the programming interface for WLE EJB and RMI object operations. To create client stubs, use the `weblogic.rmic` command to compile the Java `.class` files for the WLE EJB or RMI object.

For more information, see the RMI compiler information in the WLE documentation:

Make sure the WebLogic Server `CLASSPATH` environment variable includes the directory that contains the client stubs for the WLE EJB or RMI object.

Step 2. Import Java Packages

Import the `javax.naming.*` Java package into your WebLogic Server client.

You also need to import the package for the stubs and skeletons of the WLE EJB or RMI object you access. For an example of how to import a package for stubs and skeletons, see the WebLogic Enterprise Connectivity JSP Stateless Session Bean Example in the `/samples/examples/wlec` directory in the WebLogic Server installation.

Step 3. Connect the WebLogic Server Client to a WLE Domain

To access a WLE domain from a WebLogic Server client, use a hash table to set environment properties for the JNDI `InitialContext` class. The `InitialContext` class implements the `Context` interface and provides the starting context for naming operations. The environment properties determine how the `InitialContext` attaches to a WLEC connection pool. Include the following code in your WebLogic Server client to connect to a WLE domain:

```
public Context getInitialContext() throws Exception {
    Hashtable myEnvironment = new Hashtable();
    myEnvironment.put(Context.PROVIDER_URL, "wlepool://myPool");
    myEnvironment.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.beasys.jndi.WLEInitialContextFactory");
    myEnvironment.put(Context.SECURITY_AUTHENTICATION,
        "securityStyle");
    myEnvironment.put(Context.SECURITY_PRINCIPAL, "username");
    myEnvironment.put(Context.SECURITY_CREDENTIALS, "password");
    return new InitialContext(myEnvironment);
}
.
.
.
myContext = getInitialContext();
```

where

- *`myEnvironment`* is the name of the JNDI hash table that sets the environment properties.

- `PROVIDER_URL`, `INITIAL_CONTEXT_FACTORY`, `SECURITY_AUTHENTICATION`, `SECURITY_PRINCIPAL`, and `SECURITY_CREDENTIALS` are the JNDI environment properties. For more information, see the Javadoc for the `weblogic.jndi` package.
- `wlepool` is the constant that specifies access to a WLE domain.
- *`myPool`* is the name of the WLEC connection pool for the desired WLE domain. This WLEC connection pool needs to be defined in the Administration Console.
- `com.beasys.jndi.WLEInitialContextFactory` is the class that provides the implementation for delegating JNDI methods to the WLE service provider.
- *`myContext`* is the result of the call to the `getInitialContext()` method.

Step 4. Get an Object Reference for the WLE EJB or RMI Object

To get a reference to a WLE EJB or RMI object, include perform a JNDI lookup and cast the object:

```
myObjectRef = (myObjectType)myContext.lookup("objectName");
```

where

- *`myObjectRef`* is a reference to an object of type *`myObjectType`*.
- *`myObjectType`* is the object type to which the object is being cast.
- *`myContext`* is the JNDI context that was defined in Step 4. Get an Object Reference for the WLE EJB or RMI Object.
- For an EJB, *`objectName`* is defined in the deployment descriptor. For an RMI object, an instance of the object was created and bound to *`objectName`*.

Step 5. Access the WLE EJB or RMI Object and Its Operations

Call methods the EJBs or RMI objects in the WLE domain associated with the WLEC connection pool. If desired, you can access objects and operations within the scope of a transaction. To access EJBs and RMI objects in a transaction context, you must use the UserTransaction interface.

Working with Transactions

This section includes additional information about transactions in the WLE system.

Multithreading

The WLE 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 WLE 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 WLE 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 WLEC 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 WLE 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 WLE domains. If you try to make a call for an object in a different domain, the WLE 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 WLEC component does not use this connection to send requests that are not in the transaction context. The WLEC 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 WLEC 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.

Security

The security context established in WebLogic Server is propagated to the WLE domain. Therefore, WebLogic Server clients do not have access to the WLE security API. If you try to access the security API by calling `resolve_initial_reference("SecurityCurrent")` on the `Tobj_Bootstrap` object, the WLE system throws an `InvalidName` exception.

For more information about propagating a WebLogic Server security context to the WLE domain, see Using WebLogic Server as a Client to WebLogic Enterprise and Configuring Security Context Propagation.