BEA WebLogic Server®

Programming WebLogic RMI over IIOP

Version 9.0 BETA
Revised: December 15, 2004

Contents

Introduction and Roadmap

This section describes the contents and organization of this guide—*Programming WebLogic RMI over IIOP*.

Document Scope and Audience

This document is written for application developers who want to build e-commerce applications using the Remote Method Invocation (RMI) over Internet Interop-Orb-Protocol (IIOP) features. It is assumed that readers know Web technologies, object-oriented programming techniques, CORBA, and the Java programming language. This document emphasizes the value-added features provided by WebLogic Server® and key information about how to use WebLogic Server features and facilities when developing RMI over IIOP applications.

Guide to this Document

This document explains Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP) and describes how RMI-IIOP extends the RMI programming model by enabling Java clients to access both Java and CORBA remote objects in the BEA WebLogic Server environment.

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide.
- Chapter 2, "Overview of RMI over IIOP," defines RMI and RMI over IIOP, and provides general information about the WebLogic Server RMI-IIOP implementation.
- Chapter 3, "Configuring WebLogic Server for RMI-IIOP," describes concepts, issues, and procedures related to using WebLogic Server to support RMI-IIOP applications.

Related Documentation

For information on topics related to WebLogic RMI, see the following documents:

- *Java^{TM} Remote Method Invocation (RMI)* is a link to basic Sun MicroSystems tutorials on Remote Method Invocation.
- *Developing WebLogic Server Applications* is a guide to developing WebLogic Server applications.
- *Programming WebLogic JNDI* is a guide using the WebLogic Java Naming and Directory Interface toprimary source of information about deploying WebLogic Server applications.
- CORBA Technology and the Java Platform provides an overview of CORBA and Java platform.
- Java^{TM} IDL Technology contains information using standard IDL (Object Management Group Interface Definition Language) and IIOP.
- omg.org is the Object Management Group homepage.
- CORBA Language Mapping Specifications at http://www.omg.org/technology/documents/index.htm
- Objects-by-Value Specification at ftp://ftp.omg.org/pub/docs/orbos/98-01-18.pdf

Samples and Tutorials

In addition to this document, BEA Systems provides a variety of code samples and tutorials for developers. The examples and tutorials illustrate WebLogic Server in action, and provide practical instructions on how to perform key development tasks.

The `examples.iiop` package is available for download at http://dev2dev.bea.com/code/certwls90.jsp (not for beta). The examples in this package demonstrate connectivity between numerous clients and applications, including using EJB's with RMI-IIOP, connecting to C++ clients, and setting up interoperability with a Tuxedo Server. Refer to the example documentation for more details. For examples pertaining specifically to WebLogic Tuxedo Connector, download the `examples.wtc` package.

BEA recommends that you run some or all of the IIOP examples before developing your own applicationss.

Avitek Medical Records Application (MedRec) and Tutorials

MedRec is an end-to-end sample J2EE application shipped with WebLogic Server that simulates an independent, centralized medical record management system. The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients.

MedRec demonstrates WebLogic Server and J2EE features, and highlights BEA-recommended best practices. MedRec is included in the WebLogic Server distribution, and can be accessed from the Start menu on Windows machines. For Linux and other platforms, you can start MedRec from the `WL_HOME\samples\domains\medrec` directory, where `WL_HOME` is the top-level installation directory for WebLogic Platform.

MedRec includes a service tier comprised primarily of Enterprise Java Beans (EJBs) that work together to process requests from web applications, web services, and workflow applications, and future client applications. The application includes message-driven, stateless session, stateful session, and entity EJBs.

Examples in the WebLogic Server Distribution

WebLogic Server 9.0 optionally installs API code examples in `WL_HOME\samples\server\examples\src\examples`, where `WL_HOME` is the top-level directory of your WebLogic Server installation. You can start the examples server, and obtain information about the samples and how to run them from the WebLogic Server 9.0 Start menu.

New and Changed Features in This Release

The following sections provides information on new and changed features for this release of WebLogic RMI over IIOP:

- Compliance with the Java^{TM} 2 Platform Standard Edition 1.4 API Specification

- CallRouter support
- Weight-based RH support
- POA Support
- IDL failover Support
- CSIv2 Support

For more release-specific information on new and changed features, see these sections in *WebLogic Server 9.0 Release Notes*:

- "WebLogic Server 9.0 Features and Changes" lists new, changed, and deprecate features.
- "WebLogic Server 9.0 Known Issues" lists known problems by service pack, for all WebLogic Server APIs.

For more release-specific information about the hardware and software configurations supported by BEA for this release of WebLogic Server, see WebLogic Platform Supported Configurations.

Overview of RMI over IIOP

The following sections provide a high-level view of RMI over IIOP:

- What Are RMI and RMI over IIOP?
- Overview of WebLogic RMI-IIOP
- Protocol Compatibility

What Are RMI and RMI over IIOP?

To understand RMI-IIOP, you should first have a working knowledge of RMI. Remote Method Invocation (RMI) is the standard for distributed object computing in Java. RMI enables an application to obtain a reference to an object that exists elsewhere in the network, and then invoke methods on that object as though it existed locally in the client's virtual machine. RMI specifies how distributed Java applications should operate over multiple Java virtual machines. RMI is written in Java and is designed exclusively for Java programs.

RMI over IIOP extends RMI to work across the IIOP protocol. This has two benefits that you can leverage. In a Java to Java paradigm this allows you to program against the standardized Internet Interop-Orb-Protocol (IIOP). If you are not working in a Java-only environment, it allows your Java programs to interact with Common Object Request Broker Architecture (CORBA) clients and execute CORBA objects.CORBA clients can be written in a variety of languages (including C++) and use the Interface-Definition-Language (IDL) to interact with a remote object.

Overview of WebLogic RMI-IIOP

RMI over IIOP is based on the RMI programming model and, to a lesser extent, the Java Naming and Directory Interface (JNDI). For detailed information on WebLogic RMI and JNDI, refer to *Using WebLogic RMI* at `http://e-docs.bea.com/wls/docs90/rmi/rmi_api.html` and *Programming with WebLogic JNDI* at `http://e-docs.bea.com/wls/docs90/jndi`. Both technologies are crucial to RMI-IIOP and it is highly recommended that you become familiar with their general concepts before starting to build an RMI-IIOP application.

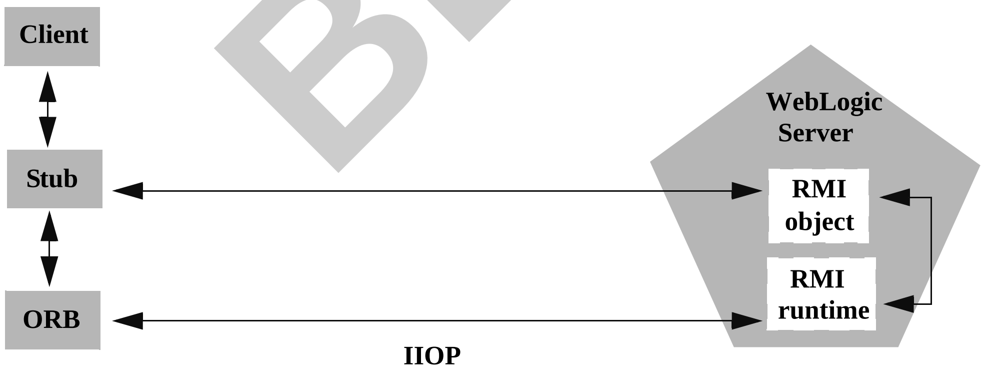
The WebLogic Server implementation of RMI-IIOP allows you to:

- Connect Java RMI clients to WebLogic Server using the standardized IIOP protocol
- Connect CORBA/IDL clients, including those written in C++, to WebLogic Server
- Interoperate between WebLogic Server and Tuxedo clients
- Connect a variety of clients to EJBs hosted on WebLogic Server

How you develop your RMI-IIOP applications depends on what services and clients you are trying to integrate. See Programming Stand Alone Clients for more information on how to create applications for various clients types that use RMI and RMI-IIOP.

Figure 2-1 shows RMI Object Relationships for objects that use IIOP.

Figure 2-1 RMI Object Relationships



Support for RMI-IIOP with RMI (Java) Clients

You can use RMI-IIOP with Java/RMI clients, taking advantage of the standard IIOP protocol. WebLogic Server 9.0 provides multiple options for using RMI-IIOP in a Java-to-Java environment, including the new J2EE Application Client (thin client), which is based on the new small footprint client jar. To use the new thin client, you need to have the `wlclient.jar` (located in *`WL_HOME`*`/server/lib`) on the client side's CLASSPATH. For more information on RMI-IIOP client options, see Overview of RMI-IIOP Programming Models in *Programming Stand Alone Clients*.

Support for RMI-IIOP with Tuxedo Client

WebLogic Server 9.0 contains an implementation of the WebLogic Tuxedo Connector, an underlying technology that enables you to interoperate with Tuxedo servers. Using WebLogic Tuxedo Connector, you can leverage Tuxedo as an ORB, or integrate legacy Tuxedo systems with applications you have developed on WebLogic Server. For more information, see the *WebLogic Tuxedo Connector Guide* at http://e-docs.bea.com/wls/docs90/wtc.html.

Support for RMI-IIOP with CORBA/IDL Clients

The developer community requires the ability to access J2EE services from CORBA/IDL clients. However, Java and CORBA are based on very different object models. Because of this, sharing data between objects created in the two programming paradigms was, until recently, limited to Remote and CORBA primitive data types. Neither CORBA structures nor Java objects could be readily passed between disparate objects. To address this limitation, the Object Management Group (OMG) created the Objects-by-Value specification . This specification defines the enabling technology for exporting the Java object model into the CORBA/IDL programming model--allowing for the interchange of complex data types between the two models. WebLogic Server can support Objects-by-Value with any CORBA ORB that correctly implements the specification.

Protocol Compatibility

Interoperability between WebLogic Server 9.0 and WebLogic Server 6.1, 7.0, and 8.1 is supported in the following scenarios:

- Server-to-Server Interoperability
- Client-to-Server Interoperability

Server-to-Server Interoperability

The following table identifies supported options for achieving interoperability between two WebLogic Server instances.

Table 2-1 WebLogic Server-to-Server Interoperability

To Server / From Server	WebLogic Server 6.1 SP2 and any service pack higher than SP2	WebLogic Server 7.0	WebLogic Server 8.1	WebLogic Server 9.0
WebLogic Server 6.1 SP2 and any service pack higher than SP2	RMI/T3 RMI/IIOP[1] HTTP Web Services	RMI/T3 RMI/IIOP[2] HTTP Web Services	RMI/T3[3] RMI/IIOP[4] HTTP Web Services[5]	RMI/T3[6] RMI/IIOP[7] HTTP Web Services[8]
WebLogic Server 7.0	RMI/T3 RMI/IIOP[9] HTTP	RMI/T3 RMI/IIOP[10] HTTP Web Services	RMI/T3 RMI/IIOP[11] HTTP Web Services[12]	RMI/T3 RMI/IIOP[13] HTTP Web Services[14]
WebLogic Server 8.1	RMI/T3 RMI/IIOP[15] HTTP	RMI/T3 RMI/IIOP[16] HTTP Web Services[17]	RMI/T3 RMI/IIOP HTTP Web Services	RMI/T3 RMI/IIOP HTTP Web Services
WebLogic Server 9.0	RMI/T3 RMI/IIOP[18] HTTP	RMI/T3 RMI/IIOP[19] HTTP Web Services[20]	RMI/T3 RMI/IIOP HTTP Web Services	RMI/T3 RMI/IIOP HTTP Web Services
Sun JDK ORB client[21]	RMI/IIOP[22]	RMI/IIOP[23]	RMI/IIOP[24]	RMI/IIOP[25]

1. No support for clustered URLs and no transaction propagation
2. No support for clustered URLs and no transaction propagation
3. Known problems with exception marshalling with releases prior to 6.1 SP4
4. No support for clustered URLs and no transaction propagation. Known problems with exception marshalling.
5. Must use portable client stubs generated from the "To Server" version
6. Known problems with exception marshalling with releases prior to 6.1 SP4
7. No support for clustered URLs and no transaction propagation. Known problems with exception marshalling.

8. Must use portable client stubs generated from the “To Server” version
9. No support for clustered URLs and no transaction propagation
10. No support for clustered URLs
11. No support for clustered URLs
12. Must use portable client stubs generated from the “To Server” version
13. No support for clustered URLs
14. Must use portable client stubs generated from the “To Server” version
15. No support for clustered URLs and no transaction propagation. Known problems with exception marshalling
16. No support for clustered URLs and no transaction propagation
17. Must use portable client stubs generated from the “To Server” version
18. No support for clustered URLs and no transaction propagation. Known problems with exception marshalling
19. No support for clustered URLs and no transaction propagation
20. Must use portable client stubs generated from the “To Server” version
21. This option involves calling directly into the JDK ORB from within application hosted on WebLogic Server.
22. JDK 1.3.x only. No clustering. No transaction propagation
23. JDK 1.3.x or 1.4.1. No clustering. No transaction propagation
24. JDK 1.3.x or 1.4.1. No clustering. No transaction propagation
25. JDK 5.0. No clustering. No transaction propagation

Client-to-Server Interoperability

The following table identifies supported options for achieving interoperability between a stand-alone Java client application and a WebLogic Server instance.

Table 2-2 Client-to-Server Interoperability

To Server From Client (stand-alone)	WebLogic Server 6.1	WebLogic Server 7.0	WebLogic Server 8.1	WebLogic Server 9.0
WebLogic Server 6.1	RMI/T3 HTTP Web Services	RMI/T3 HTTP Web Services[1]	RMI/T3[2] HTTP Web Services[3]	RMI/T3[4] HTTP Web Services[5]
WebLogic Server 7.0	RMI/T3 RMI/IIOP[6] HTTP	RMI/T3 RMI/IIOP[7] HTTP Web Services	RMI/T3 RMI/IIOP[8] HTTP Web Services[9]	RMI/T3 RMI/IIOP[10] HTTP Web Services[11]
WebLogic Server 8.1	RMI/T3 RMI/IIOP[12] HTTP	RMI/T3 RMI/IIOP[13] HTTP Web Services[14]	RMI/T3 RMI/IIOP HTTP Web Services	RMI/T3 RMI/IIOP HTTP Web Services
WebLogic Server 9.0	RMI/T3 RMI/IIOP[15] HTTP	RMI/T3 RMI/IIOP[16] HTTP Web Services[17]	RMI/T3 RMI/IIOP HTTP Web Services	RMI/T3 RMI/IIOP HTTP Web Services
Sun JDK ORB client[18]	RMI/IIOP[19]	RMI/IIOP[20]	RMI/IIOP[21]	RMI/IIOP[22]

1. Must use portable client stubs generated from the “To Server” version
2. Known problems with exception marshalling with releases prior to 6.1 SP4
3. Must use portable client stubs generated from the “To Server” version
4. Known problems with exception marshalling with releases prior to 6.1 SP4
5. Must use portable client stubs generated from the “To Server” version
6. No Cluster or Failover support. No transaction propagation
7. No Cluster or Failover support
8. No Cluster or Failover support
9. Must use portable client stubs generated from the “To Server” version
10. No Cluster or Failover support

11. Must use portable client stubs generated from the “To Server” version
12. No Cluster or Failover support and no transaction propogation. Known problems with exception marshalling
13. No Cluster or Failover support and no transaction propogation. Known problems with exception marshalling
14. Must use portable client stubs generated from the “To Server” version
15. No Cluster or Failover support and no transaction propogation. Known problems with exception marshalling
16. No Cluster or Failover support and no transaction propogation. Known problems with exception marshalling
17. Must use portable client stubs generated from the “To Server” version
18. This option involved calling directly into the JDK ORB from within a client application.
19. JDK 1.3.x only. No clustering. No transaction propagation
20. JDK 1.3.x or 1.4.1. No clustering. No transaction propagation
21. JDK 1.3.x or 1.4.1. No clustering. No transaction propagation
22. JDK 5.0. No clustering. No transaction propagation

Configuring WebLogic Server for RMI-IIOP

The following sections describe concepts and procedures relating to configuring WebLogic Server for RMI-IIOP:

Set the Listening Address

To facilitate the use of IIOP, always specify a valid IP address or DNS name for the Listen Address attribute in the configuration file (`config.xml`) to listen for connections.

The Listen Address default value of `null` allows it to "listen on all configured network interfaces". However, this feature only works with the T3 protocol. If you need to configure multiple listen addresses for use with the IIOP protocol, then use the Network Channel feature, as described in "Configuring Network Resources."

Setting Network Channel Addresses

The following sections provide information to consider when implementing IIOP network channel addresses for thin clients.

Considerations for Proxys and Firewalls

Many typical environments use firewalls, proxys, or other devices that hide the application server's true IP address. Because IIOP relies on a per-object addressing scheme where every object contains a host and port, anything that masks the true IP address of the server will prevent the external client from maintaining a connection. To prevent this situation, set the PublicAddress on the server IIOP network channel to the virtual IP that the client sees.

Considerations for Clients with Multiple Connections

IIOP clients publish addressing information that is used by the application server to establish a connection. In some situations, such as running a VPN where clients have more than one connection, the server cannot see the IP address published by the client. In this situation, you have two options:

- Use a bi-directional form of IIOP. Use the following WebLogic flag:

  ```
  -Dweblogic.corba.client.bidir=true
  ```

  In this instance, the server does not need the IP address published by the client because the server uses the inbound connection for outbound requests.

- Use the following JDK property to set the address the server uses for outbound connectons:

  `-Dcom.sun.CORBA.ORBServerHost=client_`*`ipaddress`*

  where `client_`*`ipaddress`* is an address published by the client.

Using a IIOPS Thin Client Proxy

The IIOPs Thin Client Proxy provides a WebLogic thin client the ability to proxy outbound requests to a server. In this situation, each user routes all outbound requests through their proxy. The user's proxy then directs the request to the WebLogic Server. You should use this method when it is not practical to implement a Network Channel. To enable a proxy, set the following properties:

```
-Diiops.proxyHost=<host>

-Diiops.proxyPort=<port>
```

where:

- *hostname* is the network address of the user’s proxy server.
- *port* is the port number. If not explicitly set, the value of the port number is set to 80.
- *hostname* and *port* support symbolic names, such as:

```
-Diiops.proxyHost=https.proxyHost
-Diiops.proxyPort=https.proxyPort
```

You should consider the following security implications:

- This feature does not change the behavior of WebLogic Server. However, using this feature does expose IP addresses though the client’s firewall. As both ends of the connection are trusted and the linking information is encrypted, this is an acceptable security level for many environments.
- Some production environments do not allow enabling the `CONNECT` attribute on the proxy server. These environments should use HTTPS tunneling. For more information, see Setting Up WebLogic Server for HTTP Tunneling in *Configuring and Managing WebLogic Server.*

Using RMI-IIOP with SSL and a Java Client

The Java clients that support SSL are the thin client and the WLS-IIOP client. To use SSL with these clients, simply specify an ssl url.

Accessing WebLogic Server Objects from a CORBA Client through Delegation

WebLogic Server provides services that allow CORBA clients to access RMI remote objects. As an alternative method, you can also host a CORBA ORB (Object Request Broker) in WebLogic Server and delegate incoming and outgoing messages to allow CORBA clients to indirectly invoke any object that can be bound in the server.

Overview of Delegation

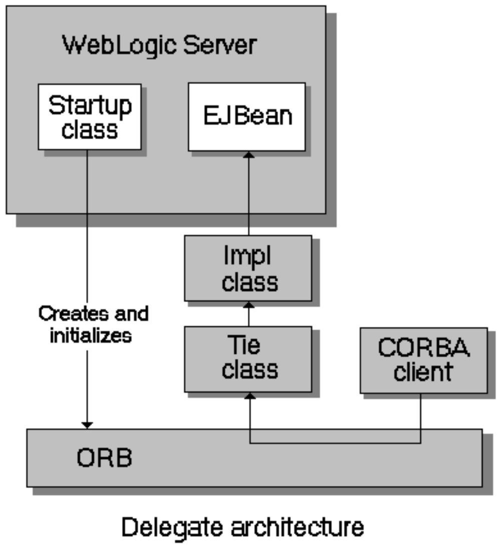
Here are the main steps to create the objects that work together to delegate CORBA calls to an object hosted by WebLogic Server.

1. Create a startup class that creates and initializes an ORB so that the ORB is co-located with the JVM that is running WebLogic Server.

2. Create an IDL (Interface Definition Language) that will create an object to accept incoming messages from the ORB.
3. Compile the IDL. This will generate a number of classes, one of which will be the Tie class. Tie classes are used on the server side to process incoming calls, and dispatch the calls to the proper implementation class. The implementation class is responsible for connecting to the server, looking up the appropriate object, and invoking methods on the object on behalf of the CORBA client.

Figure 3-1 is a diagram of a CORBA client invoking an EJB by delegating the call to an implementation class that connects to the server and operates upon the EJB. Using a similar architecture, the reverse situation will also work. You can have a startup class that brings up an ORB and obtains a reference to the CORBA implementation object of interest. This class can make itself available to other WebLogic objects throughout the JNDI tree and delegate the appropriate calls to the CORBA object.

Figure 3-1 CORBA Client Invoking an EJB with a Delegated Call



Example of Delegation

The following code example creates an implementation class that connects to the server, looks up the `Foo` object in the JNDI tree, and calls the `bar` method. This object is also a startup class that is responsible for initializing the CORBA environment by:

- Creating the ORB
- Creating the Tie object
- Associating the implementation class with the Tie object
- Registering the Tie object with the ORB
- Binding the Tie object within the ORB's naming service

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.rmi.*;
import javax.naming.*;
import weblogic.jndi.Environment;

public class FooImpl implements Foo

{
  public FooImpl() throws RemoteException {
    super();

  }

  public void bar() throws RemoteException, NamingException {
    // look up and call the instance to delegate the call to...
    weblogic.jndi.Environment env = new Environment();
    Context ctx = env.getInitialContext();
    Foo delegate = (Foo)ctx.lookup("Foo");
    delegate.bar();
    System.out.println("delegate Foo.bar called!");

  }

  public static void main(String args[]) {
    try {
      FooImpl foo = new FooImpl();
```

```
        // Create and initialize the ORB
        ORB orb = ORB.init(args, null);

        // Create and register the tie with the ORB
        _FooImpl_Tie fooTie = new _FooImpl_Tie();
        fooTie.setTarget(foo);
        orb.connect(fooTie);

        // Get the naming context
        org.omg.CORBA.Object o = \
        orb.resolve_initial_references("NameService");
        NamingContext ncRef = NamingContextHelper.narrow(o);

        // Bind the object reference in naming

        NameComponent nc = new NameComponent("Foo", "");
        NameComponent path[] = {nc};
        ncRef.rebind(path, fooTie);

        System.out.println("FooImpl created and bound in the ORB
        registry.");

      }
      catch (Exception e) {
        System.out.println("FooImpl.main: an exception occurred:");
        e.printStackTrace();

      }

    }

  }
```

For more information on how to implement a startup class, see *Starting and Stopping WebLogic Servers*.

Using RMI over IIOP with a Hardware LoadBalancer

Note: This feature works correctly only when the bootstrap is through a hardware load-balancer.

An optional enhancement for WebLogic Server 9.0 BEA ORB and higher, supports hardware loadbalancing by forcing reconnection when bootstrapping. This allows hardware load-balancers to balance connection attempts

In most situations, once a connection has been established, the next NameService lookup is performed using the original connection. However, since this feature forces re-negotiation of the end point to the hardware load balancer, all in-flight requests on any existing connection are lost.

Use the `-Dweblogic.system.iiop.reconnectOnBootstrap` system property to set the connection behavior of the BEA ORB. Valid values are:

- true —Forces re-negotiation of the end point.
- false—Default value.

Environments requiring a hardware loadbalancer should set this property to true.

Limitations of WebLogic RMI-IIOP

The following sections outline various issues relating to WebLogic RMI-IIOP.

Limitations Using RMI-IIOP on the Client

Use WebLogic Server with JDK 1.3.1_01 or higher. Earlier versions are not RMI-IIOP compliant. Note the following about these earlier JDKs:

- Send GIOP 1.0 messages and GIOP 1.1 profiles in IORs.
- Do not support the necessary pieces for EJB 2.0 interoperation (GIOP 1.2, codeset negotiation, UTF-16).
- Have bugs in its treatment of mangled method names.
- Do not correctly unmarshal unchecked exceptions.
- Have subtle bugs relating to the encoding of valuetypes.

Many of these items are impossible to support both ways. Where there was a choice, WebLogic supports the spec-compliant option.

Limitations Developing Java IDL Clients

BEA Systems strongly recommends developing Java clients with the RMI client model if you are going to use RMI-IIOP. Developing a Java IDL client can cause naming conflicts and classpath problems, and you are required to keep the server-side and client-side classes separate. Because the RMI object and the IDL client have different type systems, the class that defines the interface for the server-side will be very different from the class that defines the interface on the client-side.

Limitations of Passing Objects by Value

To pass objects by value, you need to use value types (see Chapter 5 of the CORBA/IIOP 2.4.2 Specification for further information) You implement value types on each platform on which they are defined or referenced. This section describes the difficulties of passing complex value types, referencing the particular case of a C++ client accessing an Entity bean on WebLogic Server.

One problem encountered by Java programmers is the use of derived datatypes that are not usually visible. For example, when accessing an EJB finder the Java programmer will see a Collection or Enumeration, but does not pay attention to the underlying implementation because the JDK run-time will classload it over the network. However, the C++, CORBA programmer must know the type that comes across the wire so that he can register a value type factory for it and the ORB can unmarshal it.

Simply running `ejbc` on the defined EJB interfaces will **not** generate these definitions because they do not appear in the interface. For this reason `ejbc` will also accept Java classes that are not remote interfaces--specifically for the purpose of generating IDL for these interfaces. Review the `/iiop/ejb/entity/cppclient` example to see how to register a value type factory.

Java types that are serializable but that define `writeObject()` are mapped to custom value types in IDL. You must write C++ code to unmarshal the value type manually. See *`SAMPLES_HOME`*`/server/src/examples/iiop/ejb/entity/tuxclient/ArrayList_i.cpp` for an example of how to do so.

Note: When using Tuxedo, you can specify the `-i` qualifier to direct 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\iiop\ejb\iiop\TradeResult.idl
```

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

Propagating Client Identity

Until recently insufficient standards existed for propagating client identity from a CORBA client. If you have problems with client identity from foreign ORBs, you may need to implement one of the following methods:

- The identity of any client connecting over IIOP to WebLogic Server will default to `<anonymous>`. You can set the user and password in the `config.xml` file to establish a single identity for all clients connecting over IIOP to a particular instance of WebLogic Server, as shown in the example below:

```
<Server
Name="myserver"
NativeIOEnabled="true"
DefaultIIOPUser="Bob"
DefaultIIOPPassword="Gumby1234"
ListenPort="7001">
```

- You can also set the `IIOPEnabled` attribute in the `config.xml`. The default value is "`true`"; set this to "`false`" only if you want to disable IIOP support. No additional server configuration is required to use RMI over IIOP beyond ensuring that all remote objects are bound to the JNDI tree to be made available to clients. RMI objects are typically bound to the JNDI tree by a startup class. EJBean homes are bound to the JNDI tree at the time of deployment. WebLogic Server implements a `CosNaming Service` by delegating all lookup calls to the JNDI tree.
- This release supports RMI-IIOP `corbaname` and `corbaloc` JNDI references. Please refer to the CORBA/IIOP 2.4.2 Specification. One feature of these references is that you can make an EJB or other object hosted on one WebLogic Server available over IIOP to other Application Servers. So, for instance, you could add the following to your `ejb-jar.xml`:

```
<ejb-reference-description>
<ejb-ref-name>WLS</ejb-ref-name>
<jndi-name>corbaname:iiop:1.2@localhost:7001#ejb/j2ee/interop/foo</jndi
-name>
</ejb-reference-description>
```

The reference-description stanza maps a resource reference defined in `ejb-jar.xml` to the JNDI name of an actual resource available in WebLogic Server. The `ejb-ref-name` specifies a resource reference name. This is the reference that the EJB provider places within the `ejb-jar.xml` deployment file. The `jndi-name` specifies the JNDI name of an actual resource factory available in WebLogic Server.

Note: The `iiop:1.2` contained in the `<jndi-name>` section. This release contains an implementation of GIOP (General-Inter-Orb-Protocol) 1.2. The GIOP specifies formats for messages that are exchanged between inter-operating ORBs. This allows interoperability with many other ORBs and application servers. The GIOP version can be controlled by the version number in a `corbaname` or `corbaloc` reference.

These methods are not required when using `WLInitialContextFactory` in RMI clients or can be avoided by using the WebLogic C++ client as demonstrated in the `sectuxclient` example located at

`SAMPLES_HOME``/server/examples/src/examples/iiop/ejb/stateless/sectuxclient`.