BEA WebLogic Server

Programming RMI over IIOP

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

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Programming RMI over IIOP

Part Number	Date	Software Version
	March 3, 2001	BEA Weblogic Server Version 6.0

Contents

About This Document

1. Using WebLogic RMI over IIOP

About This Document

This document explains Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP) and how to create RMI over IIOP applications for various clients types. It describes how to extend the RMI programming model by providing the ability for clients to access RMI remote objects using IIOP in the BEA WebLogic Server environment.

This document covers the following topic:

- Chapter 1, "Using WebLogic RMI over IIOP," provides an overview to RMI over IIOP, describes how to develop an RMI over IIOP application, and explains how to configure a WebLogic server. Code segments are provided to illustrate these tasks.

What You Need to Know

This document is intended mainly for application developers who are interested in providing the ability for clients to access Remote Method Invocation (RMI) remote objects using the Internet Inter-ORB Protocol (IIOP). This allows for RMI to Common Object Request Broker Architecture (CORBA) interoperability. It assumes a familiarity with the RMI over IIOP for WebLogic Server platform, CORBA, and Java programming.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

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

A PDF version of this document is available on the RMI over IIOP for 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 RMI over IIOP for 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/.

Related Information

The following BEA RMI over IIOP for WebLogic Server documents contain information that is relevant to using RMI over IIOP.

For more information in general about RMI over IIOP refer to the following sources.

- The OMG Web Site at http://www.omg.org/
- The Sun Microsystems, Inc. Java site at http://java.sun.com/

For more information about CORBA and distributed object computing, transaction processing, and Java, refer to the Bibliography at http://edocs.bea.com/.

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA RMI over IIOP for WebLogic Server 6.0 release.

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

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

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

`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.

Convention	Item

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

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

`...`	Indicates one of the following in a command line:
	■ 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 Using WebLogic RMI over IIOP

The following sections is this chapter describe the RMI over IIOP features.

- Overview
- RMI over IIOP overview
- RMI over IIOP with SSL
- Implementing with WebLogic RMI over IIOP
- Additional considerations

Overview

WebLogic RMI over IIOP extends the RMI programming model by providing the ability for clients to access RMI remote objects using the Internet Inter-ORB Protocol (IIOP). This exposes RMI remote objects to a new class of client -- the Common Object Request Broker Architecture (CORBA) client. CORBA clients can be written in a variety of languages including C++.

Within the developer community, there is a strong demand for the ability to access J2EE services -- specifically Enterprise JavaBeans (EJB) -- from CORBA clients. Since RMI is an enabling technology for EJB, providing RMI over IIOP enhances the ability to support various clients. However, Java and CORBA are based upon very different object models. Because of this, sharing data between objects created in the two programming models was, until recently, limited to Remote and CORBA

primitive data types. Neither CORBA structures nor Java objects could be readily passed between disparate objects. As a result, the Objects-by-Value specification was created by the Object Management Group (OMG). This specification defines the enabling technology for exporting the Java object model into the CORBA programming model allowing for the interchange of complex data types between the two models.

This document describes how to create RMI over IIOP applications for various clients types. For more general information on WebLogic RMI including discussions on Java RMI clients, please refer to Using WebLogic RMI.

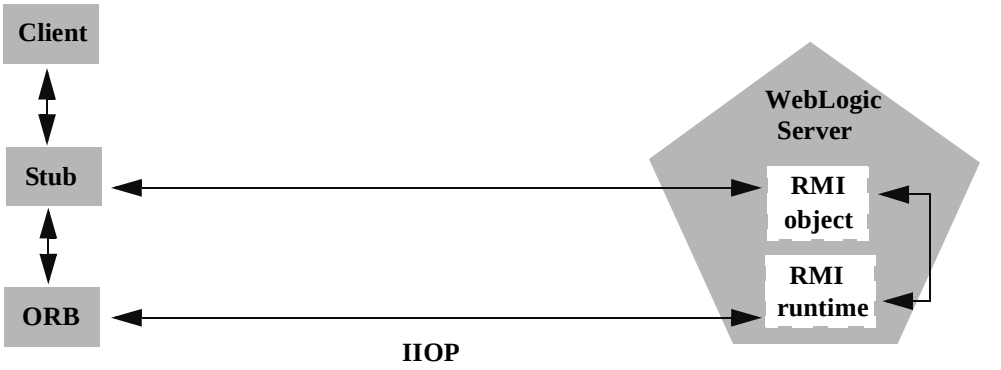
RMI over IIOP overview

This overview covers both the RMI with IIOP and the RMI with IDL programming models. RMI over IIOP give you access to a robust protocol that is supported by numerous vendors and is designed to facilitate interoperability of heterogeneous distributed objects.

RMI over IIOP

The RMI over IIOP is an application of the RMI programming model. In it programmers use JNDI and the RMI type system.

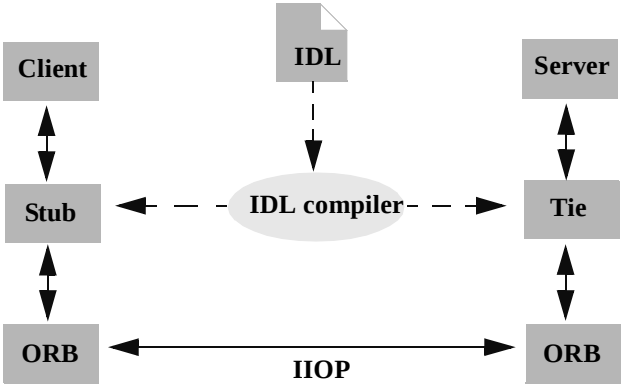
Figure 1-1 RMI object relationships



RMI over IIOP with IDL

In CORBA, interfaces to remote objects are described in a platform-neutral interface definition language (IDL). To map the IDL to a specific language, the IDL is compiled with an IDL compiler. The IDL compiler generates a number of classes such as stubs and skeletons which are used by the client and server for obtaining references to remote objects, forwarding requests, and marshalling incoming calls.

Figure 1-2 Corba object relationships



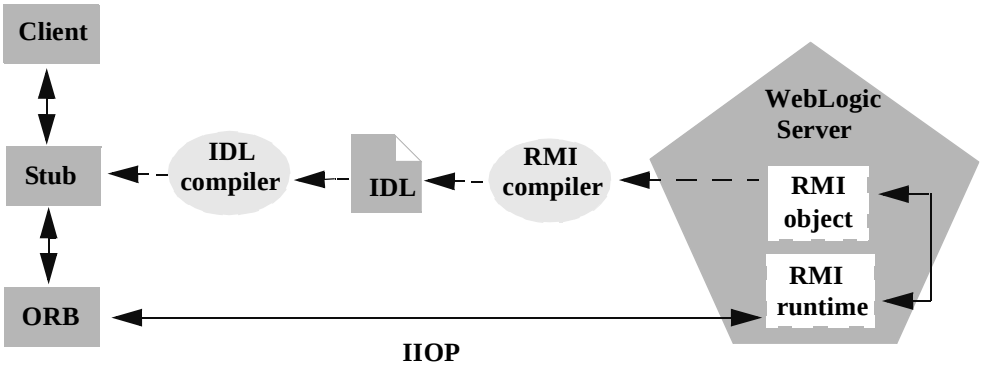
RMI over IIOP allows CORBA clients to access RMI objects and is based on two specifications of the OMG: Java-to-IDL mapping and Objects-by-value.

Java-to-IDL mapping

In WebLogic RMI, interfaces to remote objects are described in a Java remote interface that extends `java.rmi.Remote`. The Java-to-IDL mapping specification defines how an IDL is derived from a Java remote interface. In the WebLogic RMI over IIOP implementation, the implementation class is run through the WebLogic RMI compiler or WebLogic EJB compiler with the `-idl` option. This creates an IDL equivalent of the remote interface. This IDL is then compiled with an IDL compiler to generate the classes required by the CORBA client.

The client obtains a reference to the remote object and forwards method calls through the stub. WebLogic Server implements a CosNaming service that parses incoming IIOP requests and dispatches them directly into the RMI runtime.

Figure 1-3 WebLogic RMI over IIOP object relationships



Objects-by-Value

The Objects-by-Value specification allows complex data types to be passed between the two programming models. In order for a CORBA client to support Objects-by-Value, the client should be developed in conjunction with an Object Request Broker (ORB) that supports Objects-by-Value. To date, relatively few ORBs

support Objects-by-Value. When developing your RMI over IIOP application, you must consider whether your CORBA clients will support Objects-by-Value and design your RMI interface accordingly. In other words, you must limit you RMI interface to pass only primitive data types, if your application will support CORBA clients that do not support Objects-by-Value. This will be discussed further in Develop the remote interface and implementation class.

Client types

The CORBA 2.3 specification includes support for Objects-by-Value. While it is possible to support clients that utilize pre-2.3 ORBs, certain limitations will apply. There are three distinct kinds of CORBA clients you must consider when designing an RMI over IIOP application. The type of client is defined by the specification the client ORB supports and the programing model the client is developed against (RMI/JNDI or CORBA/CosNaming).

Client	Definition
RMI over IIOP client	RMI client that utilizes the CORBA 2.3 specification's support for Objects-by-Value. This Java client is developed using the standard RMI/JNDI model (with a few exceptions that are discussed in Develop the client).
IDL(OBV) client	C++ CORBA client that uses a CORBA 2.3 ORB. Note: Due to name-space conflicts, Java CORBA clients that use a CORBA 2.3 ORB are not supported by the RMI over IIOP specification.
IDL(non-OBV) client	CORBA client that uses a pre-CORBA 2.3 ORB

Implementing with WebLogic RMI over IIOP discusses how to create an RMI over IIOP application that supports these types of clients.

EJB-to-CORBA mapping

WebLogic RMI over IIOP is the framework for EJB-to-CORBA mapping support. Currently, however, a standard for passing user identity -- required to implement EJB-to-CORBA mapping -- does not exist and the requirement for transaction propagation from the client is in question. While RMI over IIOP does allow CORBA clients to access EJBeans, the following services will not be available:

- EJB transaction services
- EJB security services

RMI over IIOP with SSL

The SSL protocol can be used to protect IIOP connections to RMI remote objects. The SSL protocol secures connections through authentication and encrypts the data exchanged between objects. To use the SSL protocol to protect IIOP over RMI connections, do the following:

1. Configure WebLogic Server to use the SSL protocol. For more information, see Configuring the SSL Protocol
2. Configure the client Object Request Broker (ORB) to use the SSL protocol. Refer to the product documentation for your client ORB for information about configuring the SSL protocol.
3. Use the `host2ior` utility to print the WebLogic Server IOR to the console. The `host2ior` utility prints two versions of the IOR, one for SSL connections and one for non-SSL connections. The header of the IOR specifies whether or not the IOR can be used for SSL connections.
4. Use the SSL IOR when obtaining the initial reference to the CosNaming service that accesses the WebLogic Server JNDI tree.

Using Client Certificates

Once you have set the client ORB to support SSL, you can enforce an additional level of security by using client certificates with RMI over IIOP and SSL. The behavior will differ depending on whether you choose to enforce client certificates.

The client ORB must be aware of WebLogic Server's trusted certificate authenticator and WebLogic Server must be aware of the ORB's trusted certificate authenticator. To make WebLogic Server aware of the ORB's certificate authenticator copy the client ORB's trusted certificate authenticator to WebLogic Server.

1. Use java `utils.der2pem` to convert the certificate authenticator.
2. Copy the `ca.pem` file to a new `ca_new.pem` file.
3. Add the client ORB's trusted `ca.pem` file to the end of the new `ca_new.pem` file.
4. In the Console, change the Trusted CAFile Name to `ca_new.pem`.

   The certificate chain file will still be `ca.pem`.

Note: Refer to the ORB's product documentation to see how to make the client ORB aware of WebLogic Server's trusted certificate authenticator.

Implement the `weblogic.security.acl.CertAuthenticator` interface and register the class in the Console. See `examples.security.cert`, in your WebLogic Server distribution for a sample of how this is handled.

Using the Administration Console, set the client certificate enforced option:

1. Click the Server tab and in the right pane, choose the desired server.
2. Click the SSL tab and select the Client Certificate Enforced box.
3. Click Apply.

With RMI over IIOP and SSL, you can expect the following behavior:

If client certificates are enforced:

- The client ORB invokes on the IOR using SSL.
- On the server side, the Certificate Authenticator class is called to determine if the user is authorized.

To configure the certificate authenticator, choose on the SSL tab and specify the certificate authenticator to be used to determine the validity of the certificate. The certificate authenticator class accesses the certificate and uses the fields on the certificate to determine the user.

The samples/examples/security/cert/example contains a very simple certificate authenticator class that maps the email address from a certificate to a user in the realm.

- If the certificate authenticator is not configured or the certificate authenticator returns `null`, then a no permission exception is returned to the client and the method is not executed.

If client certificates are not enforced:

- The client ORB invokes on the IOR using SSL.
- On the server side, the invoke occurs on the default IIOP user.

  To set this user option on the SSL tab, click the Protocols tab and check the Default IIOP Users box.

Implementing with WebLogic RMI over IIOP

System requirements

WebLogic RMI over IIOP is supported under JDK 1.3 only. See WebLogic platform support for an up-to-date listing of supported platforms and JDKs.

Developing an RMI over IIOP application

This section covers developing RMI applications over IIOP with or without IDL.

RMI over IIOP

To develop an RMI over IIOP application, the following steps must be performed:

1. Develop the remote interface and implementation class and compile with a Java compiler.
2. Generate the IIOP stub classes using the `-iiop` option.
3. Compile the implementation class using the WebLogic RMI compiler or WebLogic EJB compiler.
4. Develop the client and compile with a language-specific compiler

See Using WebLogic RMI for more instructions on developing an RMI application.

RMI over IIOP using IDL

To develop an RMI over IIOP application using IDL (RMI and/or EJB), the following steps must be performed:

1. Develop the remote interface and implementation class and compile with a Java compiler
2. Generate the IDL file using the WebLogic RMI compiler or WebLogic EJB compiler.
3. Compile the IDL file with an IDL compiler and compile the resulting classes with a language-specific compiler, such as C++.
4. Develop the client and compile with a language-specific compiler

Develop the remote interface and implementation class

To develop an RMI object, you must define the object's public methods in an interface that extends `java.rmi.Remote`.

With RMI objects, you can implement the interface in a class named `interfaceNameImpl`. The implementation class it can be bound to the JNDI tree to be made available to clients. Typically, your implementation class will be configured as a WebLogic startup class and will include a `main` method that binds the object into the JNDI tree. For more information on developing RMI objects, see Using WebLogic RMI.

Special considerations for supporting non-OBV clients

If your client ORB does not support Objects-by-Value, you must limit your RMI interface to pass only other interfaces or CORBA primitive data types. The following table lists ORBs that we have tested with respect to Objects-by-Value support:

Table 1-1

Vendor	Versions	Objects-by-Value
Inprise	VisiBroker 3.3, 3.4	not supported
Inprise	VisiBroker 4.x	supported
JavaSoft	JDK 1.2	not supported
JavaSoft	RMI over IIOP Reference Implementation	supported

Generate the IDL file

After developing and compiling the implementation class, you must generate an IDL file by running the WebLogic RMI compiler or WebLogic EJB compiler with the `-idl` option. If the client is an RMI over IIOP client (as defined in Overview), you must also generate the IIOP stub classes required by the client using the `-iiop` option. If the client is an IDL client, the required stub classes will be generated when you compile the IDL file as described in the following section. For general information on the these compilers, refer to Using WebLogic RMI and BEA WebLogic Server Enterprise JavaBeans. The following compiler options are specific to RMI over IIOP:

Option	Function
`-idl`	Creates an IDL for the remote interface of the implementation class being compiled

Option	Function
`-idlDirectory`	Target directory where the IDL will be generated
`-idlNoFactories`	Do not generate factory methods for value types. This is useful if your client ORB does not support the `factory` valuetype.
`-idlOverwrite`	Causes the compiler to overwrite an existing idl file of the same name
`-idlAll`	Creates an IDL that adheres strictly to the Objects-By-Value specification
`-idlVerbose`	Display verbose information for IDL generation
`-iiop`	Creates stub classes required for RMI over IIOP clients that utilize the JDK 1.3 ORB. Note: Tie classes are also created, however these are not used by the server or client.
`-iiopDirectory`	Target directory where the IIOP classes will be generated
`-idlNoValueTypes`	Suppresses generation of idl for value types.

The options are applied as shown in this example of running the RMI compiler:

```
$ java weblogic.rmic -idl -idlDirectory /IDL
    examples.rmi_iiop.HelloImpl
```

The compiler will generate the IDL file within sub-directories of the `idlDirectoy` according to the package of the implementation class. For example, the above command will result in a `Hello.idl` file generated in the `/IDL/examples/rmi_iiop` directory. If the `idlDirectory` option is not used, the IDL file will be generated relative to the location of the generated stub and skeleton classes.

Compile the IDL file

Now that you have an IDL file, it can be used to create the stub classes required by your IDL client (as defined in Client types) to communicate with the remote class. Your ORB vendor will provide an IDL compiler.

This step is unnecessary for RMI over IIOP clients since the stub class should have already been generated using the `-iiop` option with the RMI or EJB compiler in the previous step. Note that the IIOP stubs created by the WebLogic RMI compiler are intended to be used with the JDK 1.3 ORB. If you are using another ORB, consult the ORB vendor's documentation to determine whether these stubs are appropriate.

The IDL file generated by the WebLogic compilers contains the directives: `#include orb.idl`. This IDL file should be provided by your ORB vendor. The directory containing this file should be included in the IDL compiler's include path at compile time. An `orb.idl` file is shipped in the `/lib` directory of the WebLogic distribution. This file is only intended for use with the ORB included in the JDK.

If you are developing a Java IDL(non-OBV) client, you should be careful to compile your server-side and client-side classes into separate directories and to keep the two CLASSPATHs (server- and client-side CLASSPATHs) separate. Package and class names can be repeated on the server- and client- side, particularly with the class that defines the remote interface. Since 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. To avoid conflicts, it is essential that the client CLASSPATH does not include the RMI object classes, and that the server CLASSPATH does not include any client classes.

Develop the client

With RMI over IIOP, clients may be developed using the RMI/JNDI programming model (RMI over IIOP clients) or the CORBA/`CosNaming` model (IDL clients).

RMI over IIOP clients

RMI clients access remote objects by creating an initial context and performing a lookup on the object. The object is then cast to the appropriate type. RMI over IIOP clients differ from regular RMI clients in that IIOP is defined as the protocol when obtaining an initial context. Because of this, lookups and casts must be performed in conjunction with the `javax.rmi.PortableRemoteObject.narrow()` method.

For example, in the stateless session EJB example (the `examples.ejb.basic.statelessSession` package included in your distribution), an RMI client creates an initial context, performs a lookup on the EJBean home, obtains a reference to an EJBean, and calls methods on the EJBean. To make this example work over IIOP, you must perform the following steps:

- Re-compile the EJBean and EJBean home implementation classes using the WebLogic EJB compiler with the `-iiop` option. This generates the appropriate stubs for exporting over IIOP.
- Obtain an initial context by specifying IIOP as the protocol.
- Modify the client code to perform the lookup in conjunction with the `javax.rmi.PortableRemoteObject.narrow()` method.

In the statelessSession example, the client obtains an initial context using the code below:

Listing 1-1 Obtaining an InitialContext

```
h.put(Context.INITIAL_CONTEXT_FACTORY,
  “com.sun.jndi, cosnaming.CNCtxFactory”);
h.put(Context.PROVIDER_URL, url);
InitialContext ic = new InitialContext(h);
```

where `url` defines the protocol, hostname, and listen port for the WebLogic Server and is passed in as a command-line argument. To make this client connect over IIOP, you would run client with a command like:

```
$ java examples.ejb.basic.statelessSession.Client
    iiop://localhost:7001
```

Additionally, `javax.rmi.PortableRemoteObject.narrow()` must be used in any situation where you would normally cast an object to a specific class type. For example, the client code responsible for looking up the EJBean home and casting the result to a `TraderHome` object must be modified to use the `javax.rmi.PortableRemoteObject.narrow()` as shown below:

Listing 1-2 Performing a lookup

```
TraderHome brokerage = (TraderHome)
    javax.rmi.PortableRemoteObject.narrow(
    ctx.lookup(“statelessSession.TraderHome”),
    TraderHome.class);
```

IDL clients

IDL clients are pure CORBA clients and do not require any WebLogic classes. Depending on your ORB vendor, additional classes may be generated to help resolve, narrow, and obtain a reference to the remote class. In the following example of a client developed against a VisiBroker 3.4 ORB, the client initializes a naming context, obtains a reference to the remote object, and calls a method on the remote object.

Listing 1-3 Code segment from C++ client of the RMI-IIOP hello example

```
// obtain WebLogic Server IOR from command line argument
const char* ior = argv[1];

// string to object
CORBA::Object_ptr o = orb->string_to_object(ior);

// obtain a naming context
CosNaming::NamingContext_var context =
  CosNaming::NamingContext::_narrow(o);
CosNaming::Name name;
name.length(1);
name[0].id = "HelloServer";
name[0].kind = "";

// resolve and narrow to RMI object
CORBA::Object_var object = context->resolve(name);


examples::rmi_iiop::hello::Hello_var hi =
  examples::rmi_iiop::hello::Hello::_narrow(object);
```

The naming context is obtained by narrowing a CORBA object to the WebLogic IOR. In a future version, RMI over IIOP will have “plug-and-play” capability with select ORBs and will not require obtaining the IOR of the server.

The `host2ior` utility included with WebLogic Server can be used to print the WebLogic Server IOR to the console by running the following command:

```
$ java utils.host2ior hostName port
```

where `hostName` is the name of the machine running WebLogic Server and `port` is the port where WebLogic Server is listening for connections.

Configure WebLogic Server

Because of a lack of standards for propagating client identity from a CORBA client, the identity of any client connecting over IIOP will default to "guest". The user, as well as a password, can be set in the `config.xml` file to establish a single identity for all clients connecting over IIOP, as shown in the example below:

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

There is also a `IIOPEnabled` attribute which can be set in the `config.xml`. The default value `"true"` and set this to `"false"` only if you wish 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.

Code examples

The `examples.rmi_iiop` package is included within the `/samples/examples` directory in your WebLogic installation directory. Refer to the example documentation for more details.

Additional considerations

WebLogic RMI over IIOP is intended to be a complete implementation of RMI. Please refer to the release notes for any additional considerations that might apply to your version.