



Using WebLogic RMI over IIOP

[Introduction](#)

[RMI over IIOP overview](#)

[Java-to-IDL mapping](#)

[Objects-by-Value](#)

[Client types](#)

[EJB-to-CORBA mapping](#)

[Implementing with WebLogic RMI over IIOP](#)

[System requirements](#)

[Developing an RMI over IIOP application](#)

[Develop the remote interface and implementation class](#)

[Generate the IDL file](#)

[Compile the IDL file](#)

[Develop the client](#)

[Configure WebLogic Server](#)

[Code examples](#)

[Additional considerations](#)

Other related documents:

[Using WebLogic RMI](#)

[Accessing WebLogic Server objects from a CORBA client through delegation](#)

[BEA WebLogic Server Enterprise JavaBeans](#)

Introduction

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++ and Java.

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.

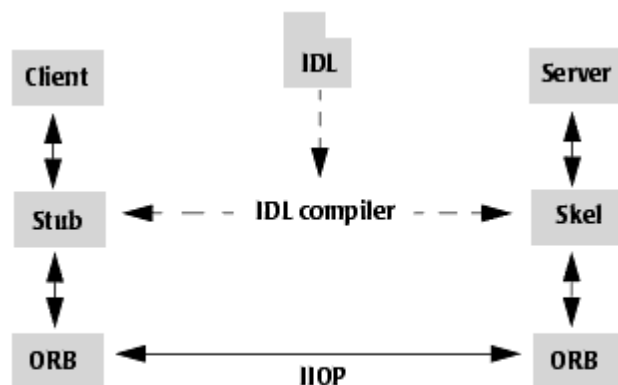
This document describes how to create RMI over IIOP applications for various clients types. For more general information on WebLogic RMI including discussions on Java RMI clients, please refer to [Using WebLogic RMI](#).

Note: Although it is possible to call an EJB using RMI-IIOP, this is **not** an officially supported feature of WebLogic Server 5.1.

RMI over IIOP overview

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 1-1 Corba object relationships



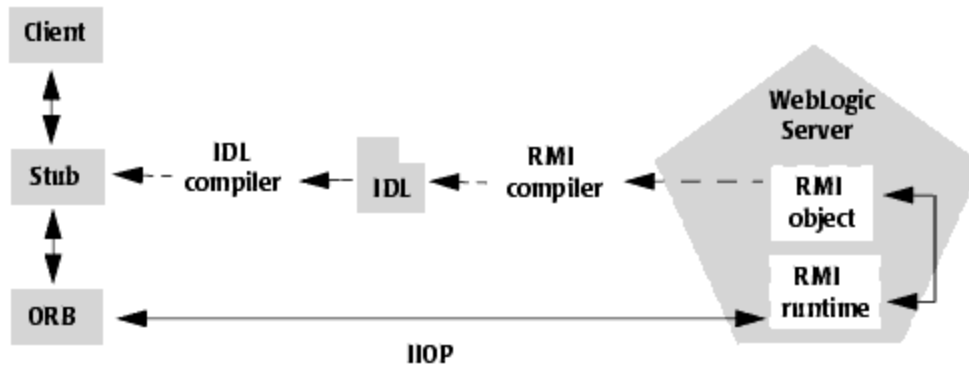
RMI over IIOP allows CORBA clients to access RMI objects and is based on two specifications of the OMG: Java-to-IDL mapping and Objects-by-value.

Java-to-IDL mapping

In WebLogic RMI over IIOP, 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 1-2 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 your 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 programming 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. **Note:** Although it is possible to call an EJB using RMI-IIOP, this is **not** an officially supported feature of WebLogic Server 5.1. While RMI over IIOP does allow CORBA clients to access EJBs, the following services will not be available:

- EJB transaction services
- EJB security services

Implementing with WebLogic RMI over IIOP

System requirements

WebLogic RMI over IIOP is supported under JDK 1.3 only. JDK 1.3 will be in production soon. Until JDK 1.3 is in final release and has been certified for WebLogic Server, WebLogic RMI over IIOP should be used for non-production purposes only. See [WebLogic platform support](#) for an up-to-date listing of supported platforms and JDKs.

Developing an RMI over IIOP application

To develop an RMI over IIOP application (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
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:

Vendor	Versions	Objects-by-Value
Inprise	VisiBroker 3.3, 3.4	not 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 [Introduction](#)), 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 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
<code>-idl</code>	Creates an IDL for the remote interface of the implementation class being compiled
<code>-idlDirectory</code>	Target directory where the IDL will be generated
<code>-idlNoFactories</code>	Do not generate factory methods for value types. This is useful if your client ORB does not support the <code>factory</code> valuetype.
<code>-idlOverwrite</code>	Causes the compiler to overwrite an existing idl file of the same name
<code>-idlAll</code>	Creates an IDL that adheres strictly to the Objects-By-Value specification
<code>-idlVerbose</code>	Display verbose information for IDL generation
<code>-iiop</code>	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.
<code>-iiopDirectory</code>	Target directory where the IIOP classes will be generated

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 `idlDirectory` 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).

Note: Although it is possible to call an EJB using RMI-IIOP, this is **not** an officially supported feature of WebLogic Server5.1.

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 EJB home, obtains a reference to an EJB, and calls methods on the EJB. To make this example work over IIOP, you must perform the following steps:

- Re-compile the EJB and EJB 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,
      "weblogic.jndi.WLInitialContextFactory");
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 EJB 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 property `weblogic.iiop.user` can be set in the `weblogic.properties` file to establish a single identity for all clients connecting over IIOP as shown in the example below:

```
weblogic.iiop.user=bob
```

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. EJB bean 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 `/weblogic/examples` directory and demonstrates connectivity with a given client ORB. 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.

BACK TO TOP ▲

[Copyright](#) © 2000 BEA Systems, Inc. All rights reserved.

Required browser: Netscape 4.0 or higher, or Microsoft Internet Explorer 4.0 or higher.