

# EJB 3.0 and IIOP.NET

2007-10-10

## Table of contents

---

1. Introduction .....	2
2. Building the EJB Sessionbean.....	3
3. External Standard Java Client .....	4
4. Java Client with RMI-IIOP.....	6
5. C# Client with IIOP.NET.....	9
6. Conclusion.....	11
7. Appendix.....	12
7.1 Ant: iiop_builder.xml .....	12
7.2 Ant: client_stub_copy.xml .....	14

# 1. Introduction

---

I played a little bit around with accessing an EJB Sessionbean from a C# client and encountered a lot of problems to get a reference of the EJB Sessionbean in C#. To help other people, I want to share my experience in this short article. This article focuses on the creation of a connection between EJB 3.0 and C# and does not describe special ways of working with it (e.g. callback). The main goal of this article is to create a usual EJB 3.0 Sessionbean and access to it on three different ways.

1. Usual external Java Client
2. Java Client with RMI-IIOP  
(this example is used to build up the basics for the access from IIOP.NET)
3. C# Client with IIOP.NET

For this article, you need the basic knowledge

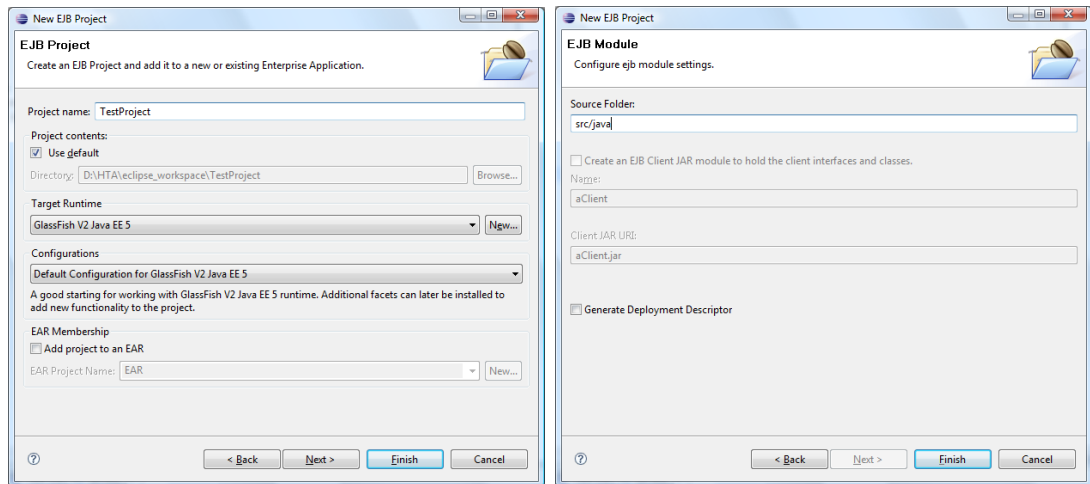
- about CORBA and IIOP (meaning of stubs and IDL should be known)
- about EJB beans and Java EE in general
- about working with Eclipse (Ant, how to manage a server) and Visual Studio

To follow up all examples, following tools are recommended

- Visual Studio 2005
- Eclipse 3.2
- Ant
- Glassfish v2 (Sun Java System Application Server 9.1)

## 2. Building the EJB Sessionbean

1. Register Glassfish in Eclipse server view
2. Create new EJB project in Eclipse and define as Target Runtime the Glassfish server. This automatically adds the Glassfish libraries to the build path. In the EJB Settings window, I recommend to use the Source Folder called src/java (to make them compatible for the Ant files, which are later used).



It's important to use a project name without spaces. Else it cannot be deployed correctly to the Glassfish server.

3. Now create the EJB 3.0 Sessionbean with its Interface.

```
package ch.stefanjaeger.ejb;

import javax.ejb.Remote;

@Remote
public interface EJBTest {
    public String hello();
    public String echo(String str);
}
```

```
package ch.stefanjaeger.ejb;

import javax.ejb.Stateless;

@Stateless
public class EJBTestBean implements EJBTest {
    public String hello() {
        return "hello, it works!";
    }
    public String echo(String str) {
        return str;
    }
}
```

4. Build the project and publish it on the Glassfish server.

### 3. External Standard Java Client

---

1. For the reason, the external Java Client isn't running in the context of the application server, @EJB isn't available. It is required to get the reference of the EJB Sessionbean with JNDI. First, create the Java Client in a new Eclipse project (e.g. TestProject2).
2. If the project is created, add the EJB project to the build path. This is required, because else the Client is unable to resolve the interface EJBTest.
3. Create the Client

```
package ch.stefanjaeger;

import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NameClassPair;
import javax.naming.NamingEnumeration;

import ch.stefanjaeger.ejb.EJBTest;

public class Client {
    public static void main(String[] args) {
        try {
            Properties p = new Properties();
            p.put("org.omg.CORBA.ORBInitialHost", "localhost");
            p.put("org.omg.CORBA.ORBInitialPort", "3700");
            InitialContext ctx = new InitialContext(p);

            // FQDN of the EJB is required
            String lookup = "ch.stefanjaeger.ejb.EJBTest";
            Object ref = ctx.lookup(lookup);
            EJBTest obj = (EJBTest)ref;

            System.out.println("hello: " + obj.hello());
            System.out.println("echo: " + obj.echo("test"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4. After running the Client, following output should appear.

```
hello: hello, it works!
echo: test
```

If the example doesn't work, check, if the EJB is correctly deployed. This can be done with `ctx.list` (described some lines beneath). Another possibility is the port configuration of the Glassfish server for remote objects. Just check the port at <http://localhost:4848> in Configuration – ORB – IIOP Listener – orb-listener-1.

5. Just a little hint: If you want to know all registered objects, just add

```
NamingEnumeration<NameClassPair> list = ctx.list("");  
while (list.hasMore()) System.out.println(list.next().getName());
```

At least the registred EJB "ch.stefanjaeger.ejb.EJBTest" should appear.

6. Just another hint: instead of defining the whole classname in the ctx.lookup(), this syntax can also be used:

```
Object ref = ctx.lookup(EJBTestHome.class.getName());
```

## 4. Java Client with RMI-IIOP

---

1. If the standard Java Client isn't working, please don't start with this example and try to get the standard Java Client working. With the standard Java Client, you can ensure a working Glassfish environment.
2. First of all, I have to explain some compatibility issues with IIOP and EJB 3.0. The goal with this Client (and also with the C# Client) is to access directly with IIOP. The properties are changed from the standard Client to

```
p.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.cosnaming.CNCtxFactory");
p.put(Context.PROVIDER_URL, "iiop://localhost:3700");
```

If the Client is started, the Exception

```
javax.naming.NameNotFoundException [Root exception is
org.omg.CosNaming.NamingContextPackage.NotFound:
IDL:omg.org/CosNaming/NamingContext/NotFound:1.0]
```

is thrown. The reason is, that access with cosnaming to EJB 3.0 Beans is only possible with EJB 2.x Home/Remote view. For this reason, the EJBTest Bean isn't registered with IIOP. To make the EJB Sessionbean accessible with IIOP, it needs to be rewritten as an EJB 2.x Bean. However, some EJB 3.0 syntax can be used ([http://blogs.sun.com/enterprisetechtips/entry/ejb\\_3\\_0\\_compatibility\\_and](http://blogs.sun.com/enterprisetechtips/entry/ejb_3_0_compatibility_and)).

3. Rewrite the EJB Sessionbean with Home and Remote interface and redeploy application.

Remote Interface extends EJBObject (which is a remote object) and adds throw RemoteException:

```
package ch.stefanjaeger.ejb;

import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface EJBTest extends EJBObject {
    public String hello() throws RemoteException;
    public String echo(String str) throws RemoteException;
}
```

Home Interface extends EJBHome and returns in the method create the EJBTest bean (which gets implemented by EJBTestBean)

```
package ch.stefanjaeger.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface EJBTestHome extends EJBHome {
    public EJBTest create() throws CreateException, RemoteException;
}
```

The EJBTestBean gets an annotation `@RemoteHome`, which defines the class of the EJBTestHome class (this class is needed for the creation of this bean). The method `create()` is declared with the annotation `@init`. This definition is required, because EJB 2.x always has a `create()` method to handle the bean creation. The `@Init` annotation tells the EJB container which method that is.

```
package ch.stefanjaeger.ejb;

import javax.ejb.Init;
import javax.ejb.RemoteHome;
import javax.ejb.Stateless;

@Stateless
@RemoteHome(EJBTestHome.class)
public class EJBTestBean {
    @Init
    public void create() {
    }
    public String hello() {
        return "hello, it works!";
    }
    public String echo(String str) {
        return str;
    }
}
```

4. If we compare the output of the `ctx.list("")` operation with the old bean and the new one (the EJB 2.x compatible bean), we can see, that the Sessionbean now get's registered correctly.

before

```
ejb
SerialContextProvider
ch.stefanjaeger\.ejb\.EJBTest__3_x_Internal_RemoteBusinessHome__
```

after

```
ejb
SerialContextProvider
ch.stefanjaeger\.ejb\.EJBTestHome
```

5. The next step is to adjust the Client. The handling of the EJB object is now a little bit different as with EJB 3.0. It uses the EJB 2.x methods `create()` for creating the object.

```
Object ref = ctx.lookup("ch.stefanjaeger.ejb.EJBTest");
EJBTestHome objHome = (EJBTestHome) PortableRemoteObject.narrow(ref,
EJBTestHome.class);
EJBTest obj = (EJBTest) objHome.create();
```

If we want to compile the client now, we are getting a syntax error. The `CreateException` of the function `objHome.create();` couldn't be resolved. The reason is that this information is stored in the JAR files from Java EE. Add to the build path the Jar files from Glassfish (in the build path window, add Variable, choose Server Runtime and add the Glassfish libraries).

Now, almost everything seems to be done. But if we run this Client, a `NullPointerException` is thrown from `objHome.create()`. The reference to `objHome` is null. That means, `PortableRemoteObject.narrow()` returns null. If you debug this situation, you will see that the class `ch.stefanjaeger.ejb._EJBTestHOMe_Stub` is missing...

If we remember what we are doing now, the reason for this missing file is clear. If you have already used RMI-IIOP, you know that you have to create the stub files manually, because they aren't created on runtime like RMI-JRMP.

6. To create the stub files, the batch file `asadmin.bat` from Glassfish is used with the parameter `generatertmstubs`. It creates a Client jar file with all stubs, based on the `ejb` jar file. The resulting Client jar file can be imported to the Client build path. The previously added EJB project to the build path can be removed.

To create the stub files and to copy the Client jar to the Client project, use the Ant files in the appendix. They are useful and save a lot of time. From the EJB project run "generate java client stub" (for the generation of the stubs, the Glassfish server need's to be running!). After that, in the folder `/build/stub` the stubs are generated. Then run "refresh stubs" from the Client Ant file, which copies the Client jar file to the lib folder of the Client (don't forget to add this jar file to the build path!).

After that, the Client should run without any problems.

## 5. C# Client with IIOP.NET

---

1. After the implementation of the Java Client over RMI-IIOP, we can be sure, the EJB is accessible with IIOP. The next part is easy. First of all, download the latest version of IIOP.NET and compile it with nmake (open Visual Studio command prompt and navigate to the IIOP.NET folder and run nmake without any parameters. It creates in every folder a bin folder with the executables).
2. The next step is to generate the IDL files from the Java classes. After that, we can generate a DLL file with the IDLToCLSCompiler.exe. To do this use the Ant file and run the target "generate iiop.net dll". Don't forget to define the IIOP.NET folder and all the EJBs, which are remotely accessible (define Property iiop\_net.path with the path and IDLToCLSCompiler.ejbs with "ch/stefanjaeger/ejb/EJBTest.idl ch/stefanjaeger/ejb/EJBTestHome.idl").  
After running the Ant file, the DLL is created (located in IIOP.NET folder\IDLToCLSCompiler\IDLCompiler\bin\tmp).
3. Create a new VS.NET project and copy the IIOPChannel.dll (inside IIOP.NET folder\IDLToCLSCompiler\IDLCompiler\bin) and EJBTestClient.dll (inside IIOP.NET folder\IDLToCLSCompiler\IDLCompiler\bin\tmp) to the project bin directory.

4. Add a reference to the DLL's to the project and enter or copy following code:

```
using System;
using System.Collections.Generic;
using System.Text;
using Ch.Elca.Iiop;
using System.Runtime.Remoting.Channels;
using Ch.Elca.Iiop.Services;
using omg.org.CosNaming;
using ch.stefanjaeger.ejb;

namespace TestApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                // register the channel
                IiopClientChannel channel = new IiopClientChannel();
                ChannelServices.RegisterChannel(channel, false);

                // access COS nameing service
                RmiIiopInit init = new RmiIiopInit("localhost", 3700);
                NamingContext nameService = init.GetNameService();

                // get the object
                NameComponent[] name = new NameComponent[] { new NameComponent("ch",
                "stefanjaeger.ejb.EJBTestHome") };

                // now, do the same as in Java
                EJBTestHome objHome = (EJBTestHome)nameService.resolve(name);
                EJBTest obj = objHome.create();
                Console.WriteLine("hello: " + obj.hello());
                Console.WriteLine("echo: " + obj.echo("test"));
                Console.ReadLine();
            }
            catch (Exception e)
            {
                Console.WriteLine("exception: " + e);
            }
        }
    }
}
```

5. You can see, that the name of the EJB isn't in one string. To get a list of all registered names (and how they are registered) use the list method:

```
omg.org.CosNaming.Binding[] bindings;
BindingIterator bindingIterator;
nameService.list(10, out bindings, out bindingIterator);
foreach (omg.org.CosNaming.Binding binding in bindings)
{
    NameComponent[] currentName = binding.binding_name;
    Console.WriteLine(currentName[0].id + " - " + currentName[0].kind);
}
Console.ReadLine();
```

6. Note: after generating the DLL, the Java exceptions were not implemented in C#. They "should" be implemented (the example works without. To get more information about this, check out the IOP.NET website).

## 6. Conclusion

---

Accessing to an EJB from C# is easy, if you know, how to do it ;-). I spend several days trying every possible solution to get Java and .NET working together. Most time I spend with was trying to get an EJB 3.0 object working with IIOP. After reading <http://forums.java.net/jive/thread.jspa?threadID=18064> and [http://blogs.sun.com/enterprisetechtips/entry/ejb\\_3\\_0\\_compatibility\\_and](http://blogs.sun.com/enterprisetechtips/entry/ejb_3_0_compatibility_and) most of the problems were solved.

I hope, this article helps you working with Glassfish and IIOP.NET. If you have any comments to this article, feel free to contact me.

## 7. Appendix

---

### 7.1 iiop\_builder.xml

It's important to change the yellow marked sections. Copy this file to the project dir. If it stored somewhere else, change the basedir in the first line.

This Ant file doesn't have a deploy function! This have to be done by Eclipse.

```
<project name="TestProject" default="generate all" basedir=".">

  <!-- common used (author and name of project for jar file) -->
  <property name="author" value="Stefan Jaeger" />
  <property name="project" value="TestProject" />

  <!-- IIOP.NET configuration -->
  <property name="IDLToCLSCompiler.target" value="EJBTestClient" />
  <property name="IDLToCLSCompiler.ejbs" value="ch/stefanjaeger/ejb/EJBTest.idl
ch/stefanjaeger/ejb/EJBTestHome.idl" />

  <!-- definition for app server -->
  <property name="sunappserver.serveraddress" value="localhost" />
  <property name="sunappserver.serverportnumber" value="8080" />
  <property name="sunappserver.adminname" value="admin" />
  <property name="sunappserver.adminpassword" value="adminadmin" />

  <!-- locations -->
  <property name="glassfish" location="D:/Java/glassfish" />
  <property name="build" location="build" />
  <property name="dist" location="dist" />
  <property name="src" location="src/java" />
  <property name="iiop_net.path" location="D:/IIOPNet.src.1.9.0.final" />

  <!-- generated locations -->
  <property name="stub" location="${build}/stub" />
  <property name="classes" location="${build}/classes" />
  <property name="IDLToCLSCompiler.path"
location="${iiop_net.path}/IDLToCLSCompiler/IDLCompiler/bin" />
  <property name="IDLToCLSCompiler.exec"
location="${IDLToCLSCompiler.path}/IDLToCLSCompiler.exe" />

  <!-- Init/Clean -->
  <target name="init" description="create directory structure">
    <mkdir dir="${build}" />
  </target>
  <target name="clean" description="clean up">
    <delete dir="${build}" />
    <delete dir="${dist}" />
  </target>

  <!-- Classpath -->
  <target name="classpath" description="generates the classpath for compiling">
    <path id="classpath">
      <fileset dir="${glassfish}/lib/">
        <!-- <include name="**/*.jar" />-->
        <!-- alle notwendig?? -->
        <include name="appserv-rt.jar" />
        <include name="javaee.jar" />
        <include name="mail.jar" />
        <include name="webservices-rt.jar" />
        <include name="webservices-tools.jar" />
        <include name="appserv-jstl.jar" />
        <include name="appserv-tags.jar" />
      </fileset>
    </path>
  </target>
</project>
```

```

        <include name="activation.jar" />
    </fileset>
</path>
</target>

<!-- compile and dist for generating jar-->
<target name="compile" depends="clean,init,classpath" description="compile the
source">
    <mkdir dir="${classes}" />
    <javac srcdir="${src}" destdir="${classes}">
        <classpath>
            <path refid="classpath" />
        </classpath>
    </javac>
</target>
<target name="dist" depends="compile" description="create jar">
    <mkdir dir="${dist}" />
    <jar jarfile="${dist}/${project}.jar" basedir="${classes}">
        <manifest>
            <attribute name="Built-By" value="${author}" />
            <attribute name="Sealed" value="false" />
        </manifest>
    </jar>
</target>

<!-- generate stubs -->
<target name="generate java client stub" depends="dist" description="generate
Client.jar with RMI-IIOP stubs">
    <mkdir dir="${stub}" />
    <exec executable="${glassfish}/bin/asadmin.bat" failonerror="true">
        <arg line=" deploy " />
        <arg line=" --generateterminstubs" />
        <arg line=" --retrieve ${stub}/" />
        <arg line=" ${dist}/${project}.jar" />
    </exec>
</target>
<!-- generate idl after dist, because idl files are generated in classes path -->
<target name="idl" depends="dist" description="generate IDL files">
    <rmic base="${classes}" idl="true">
        <classpath>
            <path refid="classpath" />
        </classpath>
    </rmic>
</target>

<target name="generate iiop.net dll" depends="idl" description="generate cls code
form idl">
    <delete dir="${IDLToCLSCompiler.path}/tmp" />

    <copy todir="${IDLToCLSCompiler.path}/tmp">
        <fileset dir="${classes}/">
            <include name="**/*.idl" />
        </fileset>
    </copy>
    <copy file="${iiop_net.path}/IDLToCLSCompiler/IDL/orb.idl"
todir="${IDLToCLSCompiler.path}/tmp" />

    <exec executable="${IDLToCLSCompiler.exec}" dir="${IDLToCLSCompiler.path}/tmp"
failonerror="true">
        <arg line=" -o . " />
        <arg line=" ${IDLToCLSCompiler.target} " />
        <arg line=" ${IDLToCLSCompiler.ejbs} " />
    </exec>

    <echo level="info" message="dll file generated in ${IDLToCLSCompiler.path}\tmp"
/>
    <echo level="info" message="on error, check property IDLToCLSCompiler.ejbs!" />
</target>

```

```
<!-- do everything -->
<target name="generate all" depends="generate iiop.net dll,generate java client
stub" description="generate all" />
</project>
```

## 7.2 client\_stub\_copy.xml

Change the paths of the project locations.

```
<project name="TestProjectClient" default="refresh_stubs" basedir=".">
  <target name="refresh_stubs" description="get stub library">
    <copy file="D:/TestProject/build/stub/RMI-IIOP.NET-GlassfishClient.jar"
todir="D:/TestProjectClient/lib" />
  </target>
</project>
```