A Sample OFBiz application implementing remote access via RMI and SOAP

Table of contents

1. About this document

document version: 1.1

released in date : 1 September 2006

2. Introduction

This tutorial describes how to build a complete application using OFBiz and how to access this application directly via RMI and SOAP over HTTP. It is entirely based on Hello3 tutorial from Open Source Strategies Inc. you may found at this url http://www.opensourcestrategies.com/ofbiz/hello_world3.php, it reproduces some parts of it, modifies some other ones or extends them when necessary.

Before starting you should have downloaded:

- The extended hello3 application (deploy it under <ofbiz_base>/hot-deploy after having unzipped it)
- The script bshcontainer.bsh (deploy under <ofbiz_base>)
- The test client RMI (deploy where you want)
- The test client SOAP (deploy where you want)
- The new OFBiz class SOAPClientEngine modified in order to make the invocation of a remote Web Service working (you should replace the old one and compile OFBiz)

It could be necessary to adjust the scripts used to compile in order to set the proper classpaths.

Warning:

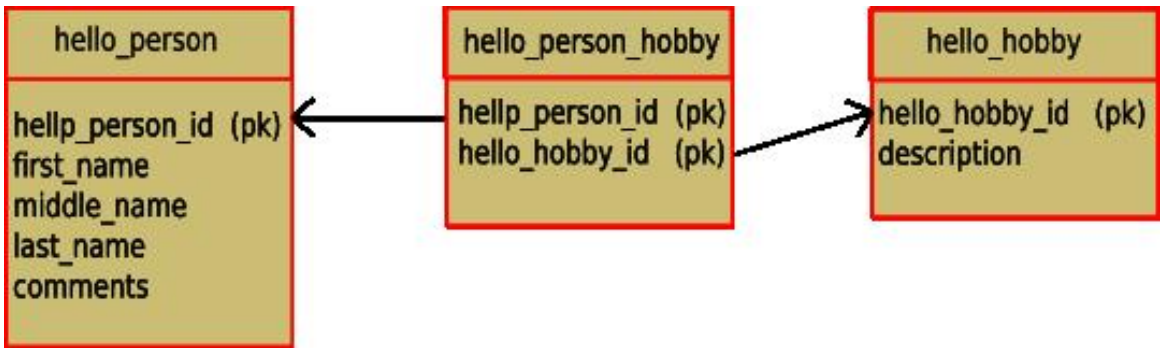
You are supposed to know at least the basics of OFBiz before reading this tutorial. Some of them could be found as well on the Open Source Strategies web site. In addition some information about Web Services and RMI could be useful.

3. Defining the data model

The first step is to define the data model. We want to track persons and their hobbies and

lookup all the hobbies of a person (or, alternatively, all the people who share a hobby.) The data model thus calls for person, hobby, and person-hobby linkage. With a relational database, you would define two tables, one for a person and one for a hobby, and link them together with a third table. The third table would allow you to associate as many hobbies as you would want with a person, and vice versa. You would define foreign-keys to constrain the third table so only actual persons and hobbies are present.

The following picture shows the ER diagram for such tables.



ER diagram

Note that you do not need to create those tables in your database manually, OFBiz will take care of that automatically, this will be clarified later.

OFBiz works similarly. You would define two entities, which we will call HelloPerson and HelloHobby, and a linking entity, HelloPersonHobby, and establish a relationship between them. The relationship serve as foreign-key constraints but also allow you to go from one entity to another without having to remember yourself what their respective keys are (or change your code when keys change.)

To define data models, go to the entitydef/ directory inside your application (hello3/ in this case) and locate the files **entitymodel.xml** and **entitygroup.xml** inside your entitydef/ directory. Let's have a look at entitymodel.xml first:

```
<entity entity-name="HelloPerson"
        package-name="org.ofbiz.hello3"
        title="Entity for storing data about persons">
  <field name="helloPersonId" type="id-ne"></field>
  <field name="firstName" type="id"></field>
  <field name="middleName" type="id"></field>
  <field name="lastName" type="id"></field>
  <field name="comments" type="comment"></field>
  <prim-key field="helloPersonId"/>
</entity>
<entity entity-name="HelloHobby"
        package-name="org.ofbiz.hello3"
        title="Hobbies available">
  <field name="helloHobbyId" type="id-ne"></field>
  <field name="description" type="description"></field>
  <prim-key field="helloHobbyId"/>
</entity>
<entity entity-name="HelloPersonHobby"
        package-name="org.ofbiz.hello3"
        title="Entity and ttribute Entity">
  <field name="helloPersonId" type="id-ne"></field>
  <field name="helloHobbyId" type="id-ne"></field>
  <prim-key field="helloPersonId"/>
  <prim-key field="helloHobbyId"/>
  <relation type="one" fk-name="HPRSN_PRSN" rel-entity-name="HelloPerson">
    <key-map field-name="helloPersonId"/>
  </relation>
  <relation type="one" fk-name="HPRSN_HBBY" rel-entity-name="HelloHobby">
    <key-map field-name="helloHobbyId"/>
  </relation>
</entity>
```

snippet from entitymodel.xml

Just a few words about the entity definition.

HelloPerson and HelloHobby each has a primary key, and HelloPersonHobby has two primary key fields, which it uses to link HelloPerson and HelloHobby. It is considered good practice to give your relations a foreign key name to make debugging easier and avoid accidental collision of foreign key names OFBiz generates for you.

There is an implicit rule linking database objects (such as table names and column names) with entity definition (entity names and field names). In particular:

entity element	rule	example
entity name	similar to Java class names: all words starting with capital letters including the first one. The corresponding database table name is lowercase and uses the separator "_"	the entity name "HelloPerson" is mapped to a database table with name "hello_person".

	(underscore).	
entity fields	similar to Java methods: all words starting with capital letters except for the first one. The corresponding database field name is lowercase and uses the separator "_" (underscore).	the field name "helloPersonId" is mapped to a database field with name "hello_person_id".

The data type is defined by the attribute "type" in the tag "field". This data type is abstract and should be translated into a real data type that is database specific. This translation is done in particular in the files fieldtype<database_name>.xml contained in the directory <ofbiz_base>/framework/entity/fieldtype.

Now the file entitygroup.xml

```
<entity-group group="org.ofbiz" entity="HelloPerson" />
<entity-group group="org.ofbiz" entity="HelloHobby" />
<entity-group group="org.ofbiz" entity="HelloPersonHobby" />
```

snippet from entitigroup.xml

The main issue with this file id the "group" attribute. It is important in order to identify the database to be used. Have a look at the file entityengine.xml you may find under <ofbiz_base>/framework/entity/config, you will see that each dispatcher (OFBiz object used to access a data source) has a corresponding group-name attribute that should match the one defined in entitygroup.xml. This group-name points to a real data source configured in the same file entityengine.xml. Once the data source is identified per each entity by using this mechanism, it is also possible to select the right data types through the entity definition and the translation rules for the fieldtypes. (described above).

Now start OFBiz. You will see the following lines in your console.log (Linux) or roll past you on your console (Windows), telling you that your entities were loaded:

```
....
2391 [            UtilXml.java:263:DEBUG] XML Read 0.0s:
    D:/ofbiz_work/hot-deploy/hello3/entitydef/entitymodel.xml
....
2719 [            UtilXml.java:263:DEBUG] XML Read 0.0s:
    D:/ofbiz_work/hot-deploy/hello3/entitydef/entitygroup.xml
....
```

When you go into Web Tools application, you will see the entities:

GoodIdentification	Crt Fnd All	ProductPriceCond
GoodIdentificationAndProduct	View Entity	ProductPriceRule
GoodIdentificationType	Crt Fnd All	ProductPriceType
HelloHobby	Crt Fnd All	ProductPromo
HelloPerson	Crt Fnd All	ProductPromoAction
HelloPersonHobby	Crt Fnd All	ProductPromoCategory
ImageDataResource	Crt Fnd All	ProductPromoCode
InventoryEventPlanned	Crt Fnd All	ProductPromoCodeEmail
InventoryEventPlannedType	Crt Fnd All	ProductPromoCodeEmailParty
InventoryItem	Crt Fnd All	ProductPromoCodeParty

entities in Web Tools

The tables have been automatically created on your database by OFBiz, including the foreign keys.

4. Populating the database tables with Seed Data

Now let's populate the person_hobby table with some seed data. In most OFBiz applications, we would create a data/ directory inside our application and create an XML file for our seed data. Let's see our HobbiesData.xml:

```
<entity-engine-xml>
    <!-- Each tag is an entity name.  Data for fields is in attribure-value pairs -->
    <HelloHobby helloHobbyId="READING" description="Reading"/>
    <HelloHobby helloHobbyId="MOVIES" description="Movies"/>
    <HelloHobby helloHobbyId="THEATER" description="The theater"/>
    <HelloHobby helloHobbyId="OPERA" description="The opera"/>
    <HelloHobby helloHobbyId="SKIING" description="Skiing"/>
    <HelloHobby helloHobbyId="SURFING" description="Surfing"/>
    <HelloHobby helloHobbyId="WINDSURFING" description="Windsurfing"/>
    <HelloHobby helloHobbyId="BASKETBALL" description="Basketball"/>
    <HelloHobby helloHobbyId="FOOTBALL_US" description="American Football"/>
    <HelloHobby helloHobbyId="FOOTBALL_OTHER" description="Football (Soccer)"/>
    <HelloHobby helloHobbyId="COOKING" description="Cooking"/>
    <!-- But data for fields can also be tags inside the entity name tags, so you can use CDATA
        for longer fields -->
    <HelloHobby helloHobbyId="WINE">
        <description>Wine</description>
    </HelloHobby>
</entity-engine-xml>
```

snippet from HobbiesData.xml seed file

The content of the file is auto-explaining.

Now you are ready to load your seed data. Go to the Web Tools application's "Main" screen, and you will see links for "XML Import". Click on "XML Import" and on the next screen, it will prompt you for the name of your file, relative to your ofbiz/ directory. I usually don't click on any of the optional check boxes and just "Import". If you are

successful, the same screen will come back and tell you at the bottom how many values were added.

Note:

More details about this phase on the tutorial by Open Source Strategies.

5. Creating Business Logic

Now that we have the data model defined, we can of course write a simple application with a delegator to access the entities directly. Standard practice for OFBiz applications, however, calls for creating a separate layer for the business logic and for creating, updating, and removing entries. The delegator is used directly for looking up values, although more complex lookups are also coded as a service.

Creating services is a two step process:

1. define the service generically in an XML file, which tells the OFBiz service engine what parameters your service takes, where to find it (class and method or location of a script) and if the external access should be enabled
2. implement the service in Java, the OFBiz minilang, or another scripting language

Service definitions are usually inside a servicedef/ directory in your application and consists of one or more services.xml files. Here is our services.xml file:

```
<!-- this will be implemented in Java -->
<service name="createHelloPerson" engine="java" export="true"
    location="org.ofbiz.hello3.Hello3Services" invoke="createHelloPerson">
    <description>Create a HelloPerson</description>
    <auto-attributes mode="IN" entity-name="HelloPerson" include="nonpk" optional="true"/>
    <attribute name="helloPersonId" mode="OUT" type="String" optional="false"/>
</service>


<!-- this will be implemented in Java -->
<service name="searchHelloPerson" engine="java" export="true"
    location="org.ofbiz.hello3.Hello3Services" invoke="searchHelloPerson">
    <description>Search a HelloPerson</description>
    <attribute name="helloPersonId" mode="IN" type="String" optional="true"/>
    <!--auto-attributes mode="OUT" entity-name="HelloPerson" include="nonpk" optional="true"/-->
    <attribute name="helloPersonIdOut" mode="OUT" type="String" optional="false"/>
    <attribute name="firstName" mode="OUT" type="String" optional="true"/>
    <attribute name="lastName" mode="OUT" type="String" optional="true"/>
</service>


<!-- this will be implemented in soap engine -->
<service name="BabelFishService" engine="soap" export="true"
    location="http://services.xmethods.net:80/perl/soaplite.cgi" invoke="BabelFish">
    <description>invoke the remote Web Service from OFBiz</description>
    <namespace>urn:xmethodsBabelFish</namespace>
    <attribute name="translationmode" mode="IN" type="String" optional="false"/>
    <attribute name="sourcedate" mode="IN" type="String" optional="false"/>
    <attribute name="return" mode="OUT" type="String" optional="false"/>
</service>
```

content of services.xml

Three services are defined here, here are some explanations:

- **createHelloPerson**: this service is implemented through a Java class (engine="java"), the class and the method to call are defined via the attributes location="org.ofbiz.hello3.Hello3Services" and invoke="createHelloPerson". The service can be also accessed externally, via RMI or SOAP for instance, since export="true". The service has one output parameter called "helloPersonId" of type String that is mandatory. The automatic mapping of all the input parameters to the fields of the entity "HelloPerson" has been adopted, all output parameters are optional.
- **searchHelloPerson**: this service is implemented through a Java class (engine="java"), the class and the method to call are defined via the attributes location="org.ofbiz.hello3.Hello3Services" and invoke="searchHelloPerson". The service can be also accessed externally since export="true". The service has one input parameter called "helloPersonId" of type String that is optional. The service has three output parameters called "helloPersonIdOut", "firstName" and "lastName", two of them are optional. Note that the automatic mapping of all the output parameters has been disabled since I've had some problems when accessing externally via SOAP wrapper.
- **BabelFishService**: this service is implemented through a SOAP engine (engine="soap"), it wraps the remote Web Service BabelFish (invoke="BabelFish") available over the Internet and reachable at the endpoint location="http://services.xmethods.net:80/perl/soaplite.cgi". You can get the WSDL at this url "http://www.xmethods.net/sd/2001/BabelFishService.wsdl". In this way you can access this service as you can do with all other OFBiz services, all the implementation details about the SOAP connection and remote invocation are done by OFBiz automatically. More details on ithis service in a dedicated section of this document : "Testing the service of OFBiz that wraps a remote Web Service".

Note:

There is also another service developed via minilang (createHelloPersonHobby) that is not documented here, see the tutorial by Open Source Strategies to get details on it.

You would also need to reference the service resource in your ofbiz-component.xml as well. In addition, you must create <classpath> directives in ofbiz-component.xml to tell it where to load up the apps. Have a look at the config file.

Now to create the services. A Java service goes inside a src/ directory in your application and is written in a standard fashion: A public class with public static methods which take two parameters, a DispatchContext for getting objects like delegators, dispatchers, locale, and security, and a Map called context which are your input parameters and returns a map of results:

```
public class Hello3Services {

    public static final String module = Hello3Services.class.getName(); // used for debugging

    /*
      this method implements the Ofbiz service createHelloPerson
      as defined in services.xml
      it creates a new person in the entity HelloPerson
    */
    public static Map createHelloPerson(DispatchContext dctx, Map context) {
        GenericDelegator delegator = dctx.getDelegator();  // always passed in with DispatchCon

        try {
            String helloPersonId = delegator.getNextSeqId("HelloPerson"); // gets next availabl
            Debug.logInfo("helloPersonId = " + helloPersonId, module); // prints to the console
            GenericValue helloPerson = delegator.makeValue("HelloPerson",
                    UtilMisc.toMap("helloPersonId", helloPersonId)); // create a GenericValue f
            helloPerson.setNonPKFields(context); // move non-primary key fields from input para
            delegator.create(helloPerson); // store the generic value, ie persists it

            Map result = ServiceUtil.returnSuccess(); // gets standard Map for successful servi
            result.put("helloPersonId", helloPersonId); // puts output parameter into Map to re
            return result; // return Map
        } catch (GenericEntityException ex) { // required if you use delegator in Java
            return ServiceUtil.returnError(ex.getMessage());
        }
    }
    .....
```

implementation of the service createHelloPerson

The service creates a new record in the entity HelloPerson, the primary key is auto-generate via a sequence, all other fields are directly taken from the input parameters. Take your time to learn how to interact with entities, load input parameters from the context and return output values.

The next picture shows the implementation of the service searchHelloPerson:

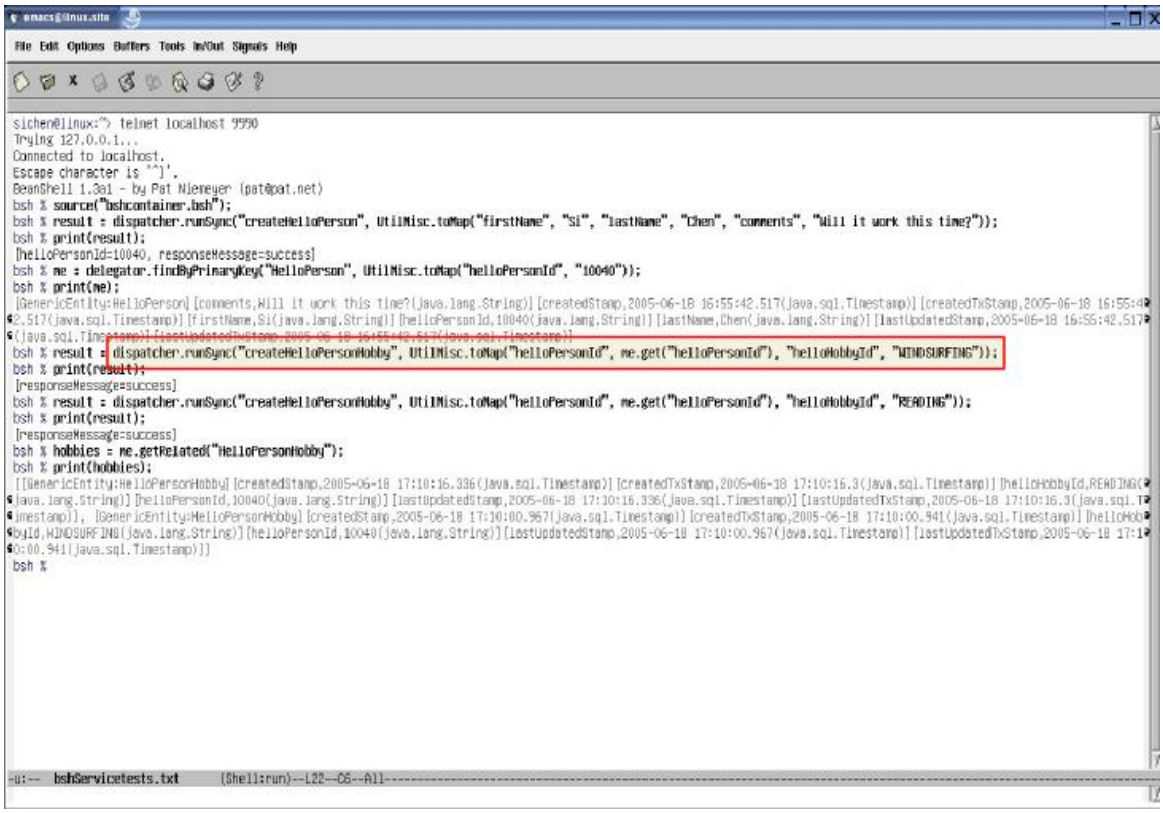
```
public static Map searchHelloPerson(DispatchContext dctx, Map context) {
    GenericDelegator delegator = dctx.getDelegator();  // always passed in with DispatchC

    try {
        String helloPersonId;

        // get the input parameter helloPersonId from the context
        helloPersonId = (String) context.get("helloPersonId");
        Debug.logInfo("helloPersonId = " + helloPersonId, module);

        // query the entity HelloPerson looking based on a findByPrimaryKey method
        Map queryResult = delegator.findByPrimaryKey("HelloPerson",
                                        UtilMisc.toMap("helloPersonId",helloPersonId));
        Debug.logInfo("queryResult = " + queryResult, module);

        // create the output Map object
        Map result = UtilMisc.toMap("helloPersonIdOut", helloPersonId);
        // fill the output object with the query results if any
        if ( queryResult != null) {
            result.put("firstName",queryResult.get("firstName"));
            result.put("lastName",queryResult.get("lastName"));
        }
        Debug.logInfo("result = " + result, module);

        // return the Map
        return result;
    } catch (GenericEntityException ex) { // required if you use delegator in Java
        return ServiceUtil.returnError(ex.getMessage());
    }
}
```

implementation of the service searchHelloPerson

The service queries the entity HelloPerson trying to get a record where the primary key field is equal to the input parameter "helloPersonId". If that is successful, then it returns also the firstName and lastName, otherwise only the helloPersonIdOut is returned.

Java services will also need to be compiled, with knowledge of the proper classpaths for other OFBiz apps. This involves using ant and a build.xml build script, which you can usually copy over from another application. Simply launch ant from the directory hello3 to compile the application.

Finally, to test it, re-start OFBiz to load all the new definitions in ofbiz-component.xml and services.xml. Then, open a beanshell window (that is connect via telnet to the port 9990 on you host) and test our service:



trying to test our services

Here, beanshell calls the service dispatcher to run the service, and they were successful, so the right values are created. Note that you need to start the script bshcontainer.bsh to have a dispatcher, a delegator and the utility UtilMisc. In the example above, the services are invoked through the method runSync of the dispatcher object.

6. Creating the Web Application

This is fully covered in the tutorial by Open Source Strategies, look at this if you are interested.

7. Accessing the services of OFBiz via RMI

The services described in the services.xml file are accessible form external tools (i.e. outside OFBiz) since they have been set with export="true". In order to access them via RMI, it is initially necessary to modify the files rmi-containers.xml and ofbiz-containers.xml of your OFBiz installation (but perhaps only the second file has to be changed).

```
<!-- RMI Service Dispatcher -->
<container name="rmi-dispatcher" class="org.ofbiz.service.rmi.RmiServiceContainer">
    <property name="bound-name" value="RMIDispatcher"/>
    <property name="bound-host" value="127.0.0.1"/>
    <property name="bound-port" value="1099"/>
    <property name="delegator-name" value="default"/>
    <!-- VINCENZO disable SSL in RMI (START) -->
    <!--
    <property name="client-factory"
              value="org.ofbiz.service.rmi.socket.ssl.SSLClientSocketFactory"/>
    <property name="server-factory"
              value="org.ofbiz.service.rmi.socket.ssl.SSLServerSocketFactory"/>
    <property name="ssl-client-auth" value="true"/>
    -->
    <property name="client-factory"
              value="org.ofbiz.service.rmi.socket.zip.CompressionClientSocketFactory"/>
    <property name="server-factory"
              value="org.ofbiz.service.rmi.socket.zip.CompressionServerSocketFactory"/>
    <property name="ssl-client-auth" value="false"/>

    <!--  VINCENZO disable SSL in RMI (END) -->
</container>
```

disabling SSL in rmi dispatcher

The modification is necessary since the certificate server side is expired (at least in my OFBiz distribution) and I do not want to generate a new one, it is only a test application. Now everything is ready and we just have to write a test client java class.

```
package testClientRmi;
import java.rmi.*;
import java.util.*;
import org.ofbiz.entity.GenericDelegator;
import org.ofbiz.service.GenericDispatcher;
import org.ofbiz.service.rmi.*;
import org.ofbiz.base.util.*;
import org.ofbiz.entity.util.*;
/*
    test class to access some services exposed by Ofbiz via rmi
*/
public class Hello3Client {
    public static void main(String args[]) {
    String helloPersonId,firstName,lastName,endpoint;
    Map resultCreate, resultQuery;
    endpoint = "rmi://127.0.0.1:1099/RMIDispatcher";
    firstName = "Donald";
    lastName  = "Duck";
        try {
            // look up the remote object by name in the rmi registry
            RemoteDispatcher rd = (RemoteDispatcher) Naming.lookup(endpoint);
            // create Donald Duck by invoking the Ofbiz service createHelloPerson
            resultCreate = rd.runSync("createHelloPerson",
                                      UtilMisc.toMap("firstName",firstName,"lastName",lastName));
            helloPersonId = (String) resultCreate.get("helloPersonId");
            System.out.println("created person:" + resultCreate);
            // search for Donald Duck by invoking the Ofbiz service searchHelloPerson
            resultQuery = rd.runSync("searchHelloPerson",
                                      UtilMisc.toMap("helloPersonId",helloPersonId));
            System.out.println("query the previously created person:" + resultQuery);
            // search for non existing person by invoking the Ofbiz service searchHelloPerson
            resultQuery = rd.runSync("searchHelloPerson",
                                      UtilMisc.toMap("helloPersonId","99999"));
            System.out.println("query a non existing person:" + resultQuery);
        } catch (Exception e) {
        .....
```

code snippet of the test client class

The code is very simple, after having received the remote handler for the rmi dispatcher, it is just needed to invoke the runSync method on it, exactly as it was done during the tests via bsh shell.

Note:

It seems that there is a problem with the RMI dispatcher if you are running OFBiz within eclipse development environment. In case of a trouble simply do not use it.

8. Accessing the services of OFBiz via SOAP

The services described in the services.xml file are accessible form external tools (i.e. outside OFBiz) since they have been set with export="true". You can access them via SOAP as well, for example if you want to get the wdsl descriptor of the service "searchHelloPerson" , just point to the following URL: "http://127.0.0.1:8080/webtools/control/SOAPService/searchHelloPerson?WSDL" with your web browser.

Here is a test client:

```
public class Hello3ClientSoap {
    public static void main(String[] args) {
        String message = "";
        Map output;
        ArrayList outputList;
        String helloPersonIdOut,firstName,lastName,endpoint,inputParam;
    try {
        endpoint = "http://127.0.0.1:18080/webtools/control/SOAPService/";
        inputParam = "10000";
        // create the standard JAX-RPC Call object, set the endpoint and method to invoke
        Call call = (Call) new Service().createCall();
        call.setTargetEndpointAddress(new URL(endpoint));
        call.setOperationName(new QName("searchHelloPerson", "searchHelloPerson"));
        // define the name of the input parameter
        call.addParameter("helloPersonId",
                          org.apache.axis.Constants.XSD_STRING,
                          javax.xml.rpc.ParameterMode.IN);
        call.setReturnType(org.apache.axis.Constants.XSD_STRING);
        // execute the remote method exposed by the WS
        // responseWS will contain the first returned parameter (helloPersonIdOut)
        Object responseWS = call.invoke(new Object[]{inputParam});
        System.out.println( "Receiving response: " +  (String) responseWS);
        // retrieve the other parameters
        output = call.getOutputParams();
        // get the output parameter firstName
        try {
            firstName = (String) output.get(new QName("", "firstName"));
        } catch (Exception _exception) {
            firstName = (String) org.apache.axis.utils.JavaUtils.convert(output.get(new QN
        }
        // get the output parameter lastName
        try {
            lastName = (String) output.get(new QName("", "lastName"));
        .....
```

code snippet of the test client class

Some comments...

- The port I am using (18080) is not the default one (8080), this is due to the fact that I trace the TCP traffic via the TCP monitor bundled with axis.
- Naming the parameters is normally not needed with axis, the input parameters are automatically named arg0, arg1, arg2 and so on. But these default names are not known by the service implemented in OFBiz, therefore the service invocation isn't successful since the validation phase could not be passed. The method addParameter is used to name the parameters (it just necessary to name all input parameters) according to the service definition.
- In case there are more than one output parameters, the first one is got as returned value from the call.invoke(..) and the other ones via the call.getOutputParameters() method. This is quite strange for me but it is exactly how axis 1.4 works and it seems not to be a bug.
- It is possible to retrieve the output parameters either by name or by position. The code snippet shown above displays only the first method, the code of the extended hello3 application uses both methods.

9. Testing the service of OFBiz that wraps a remote Web Service

It's time to test the OFBiz service "BabelFishService". As defined in services.xml, it is a service accessible like any other service of OFBiz, but its business logic is remote, in particular it is implemented as a Web Service published over the Internet.

The service is very simple, it has two input parameters, a translation string that defines the origin and the destination language for the translation and a sourcedata that is the string to translate. The returned parameter contains the translated string.

Before testing the service you should do two things:

- If you access the Internet through a proxy, you should tell Axis the address of the proxy, the port used by it and the addresses that you do not want to pass through the proxy. This means that you should modify the OFBiz startup command (startofbiz.bat in Windows), you should have something like this:

```
"%JAVA_HOME%\bin\java" -Dhttp.proxyHost=myproxy.mydomain.com
-Dhttp.proxyPort=8080
  -Dhttp.nonProxyHosts=localhost -Xms256M -Xmx512M -jar ofbiz.jar >
logs\console.log
```

- Then you should modify the class org.ofbiz.service.engine.SOAPClientEngine in order to avoid to use the method call.setOperation (well, at least this is what i've done to make it working, even if maybe it is not the best way to proceed). The modified class SOAPClientEngine is in the files attached to this document.

make a telnet connection to your host on the port 9990 and invoke the service:

```
BeanShell 1.3a1 - by Pat Niemeyer (pat@pat.net)
bsh % source("bshcontainer.bsh");
bsh % result =
dispatcher.runSync("BabelFishService",UtilMisc.toMap("translationmode",
"en_fr","sourcedate","I am"));
bsh % print(result);
[return=je suis ]
bsh %
```