Webservices with Axis2 and the n (WTP) - Tutorial

Lars Vogel and Waldemar Geppart

Version 0.6

04.07.2009

Revision History		
Revision 0.1- 0.4	25.09.2008	Lars Vogel / Waldemar Geppart
First versions		
Revision 0.5	27.10.2008	Lars Vogel
Added Axis2 Overview		
Revision 0.6	04.07.2009	Lars Vogel
Updated simple w eb-service example		

Abstract

This article is a guide to develop Java Web services with Axis2 and Eclipse WTP.

This article assume that you have already basic Eclipse knowledge. The article was developed with JDK 1.6 and Eclipse 3.5 (Galileo) and Axis2 1.4.1.

Table of Contents

1. Introduction

1.1. Eclipse Web Tool Platform (WTP)

Eclipse WTP provides functionality to develop web functionality with Java. For an introduction to Eclipse WTP please see Servlet and JSP with Eclipse WTP .

1.2. Axis2

Axis2 is a popular open source webservice framework. Axis2 uses Stax based XML parsing and uses internally the Axiom object model to represents the XML message. Based on this object model Axis2 provides an adapter which is Jax-WS compliant

2. Installation

2.1. Eclipse WTP and Tomcat

If not yet down, install now Eclipse WTP. For an installation instruction of Eclipse WTPplease see Installation of Eclipse WTP

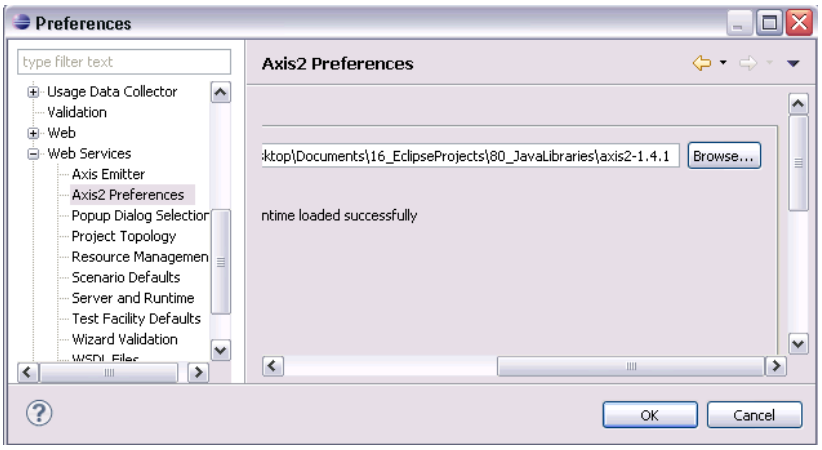
Also install Tomcat Tomcat Installation

2.2. Axis2

> Currently Axis 2 1.5 is not supported. Make sure you download Axis 1.4.1 which can be found here Axis 2 1.4.1. Download

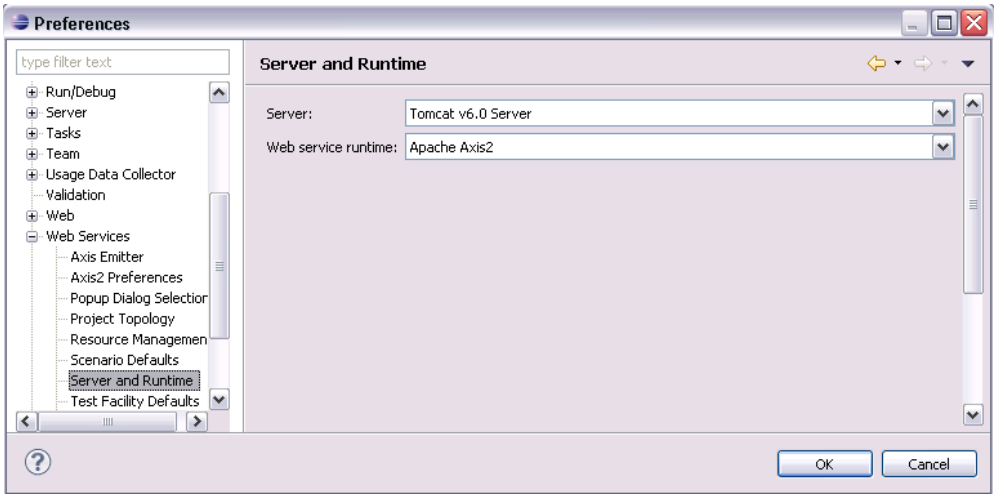
This chapter requires an Axis2 runtime, the general download site is here .

Extract the Axis 2 zip and point Eclipse WTP on it as shown below.



Click OK.

Select Window -> Preference and set Axis2 as the default engine.



3. Create a Web Service

3.1. Overview

The W3C defines a Web service as a software system designed to support interoperable machine to machine interaction over a network.

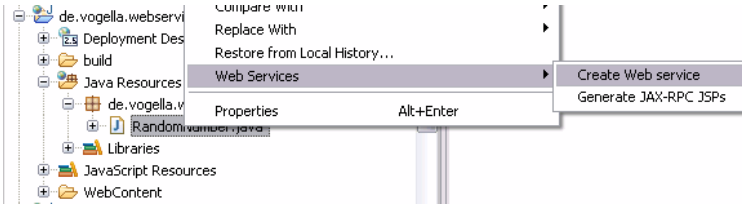
In common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard. The server usually provides a machine readable description of the operations supported by the server, a description in the Web Services Description Language (WSDL).

Create a new Dynamic Web project "de.vogella.webservice.soap.axis2". Create a package "de.vogella.webservice.soap.axis2" and the following class.

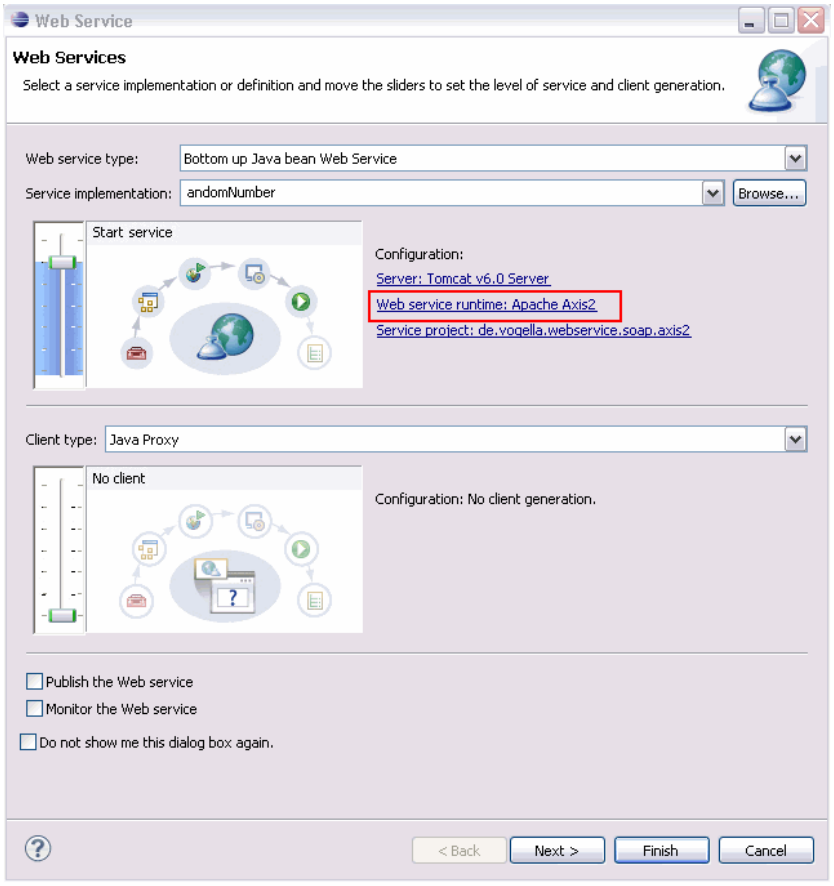
```
package de.vogella.webservice.soap.axis2;

import java.util.Random;

public class RandomNumber {
        public long getNumber(){
                Random random = new Random(1000);
                return random.nextLong();
        }
}
```

3.3. Create the web service

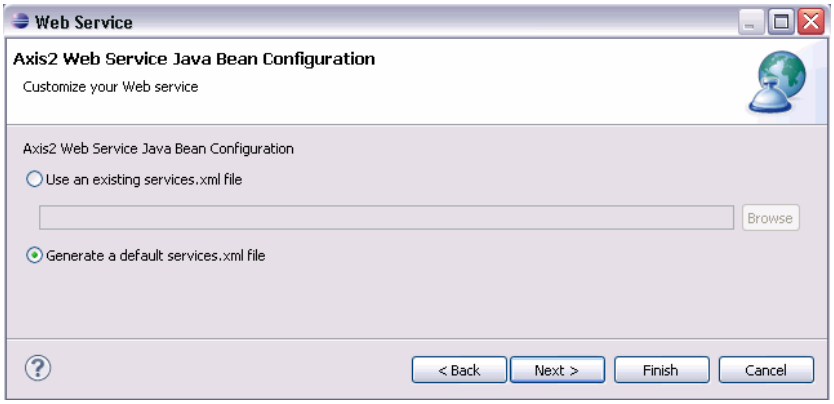
Now you are ready to create a Web Service for your class. Right click on "RandomNumber.java" and select Web Services -> Create Web Service.



Make sure the selected Web service runtime is "Apache Axis2". If not, click on the link "Web service runtime: Apache Axis2" and select Axis2.

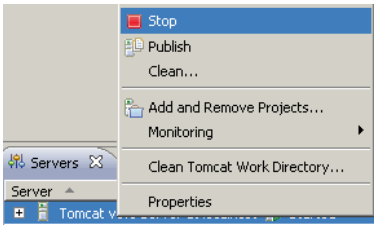


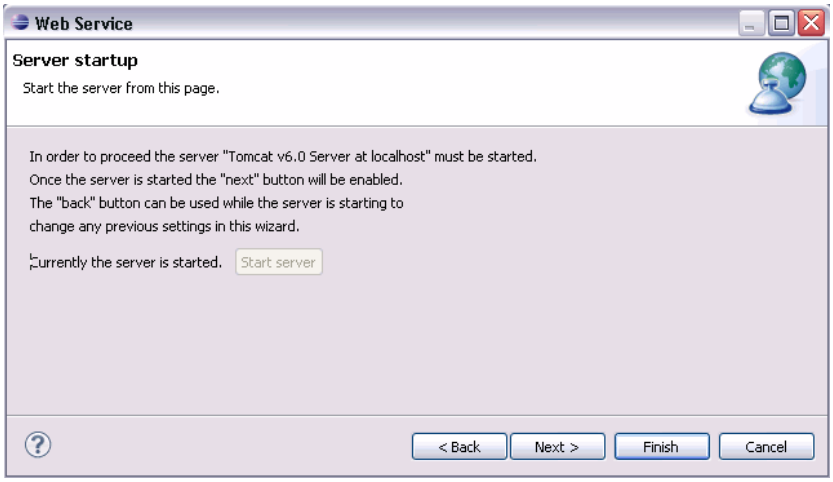
Press next.



Click "Next". If your server is not running, you will see this dialog. Click "Start server", wait a few seconds and then click "Next".

If an error occurs, you have to stop the server manually: Right click on the server in the "Servers" view and select "Stop".





Click "Finish".

Congratulations! You created a Web Service.

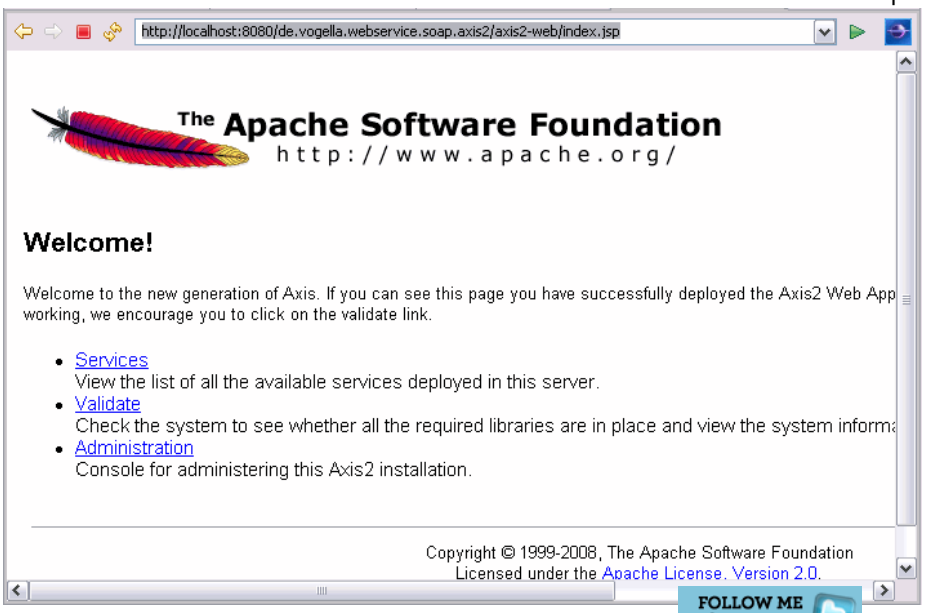
3.4. Create the webservice client

To use your webservice you need a client to. For this you need the URL of the Web Service. Right click on the project and select Run As -> Run on Server. Check the check box and click "Finish".

Most likely you will find the axis services under the web-address http://localhost:8080/de.vogella.webservice.soap.axis2/axis2-web/index.jsp.

You can check the deployment settings in your project de.vogella.webservice.soap.axis2-> Deployment Descriptor -> Servlets -> AxisServlet

You should see the following Axis2 main page.

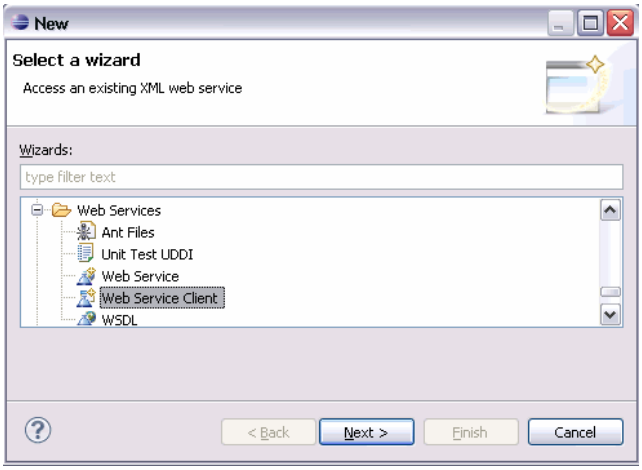


Click on the link "Services".

Click on the link "RandomNumber" and you will see the wsdl

Copy the URL, e.g. http://localhost:8080/de.vogella.webservice.soap.axis2/services/RandomNumber?wsdl , because you will need it later.

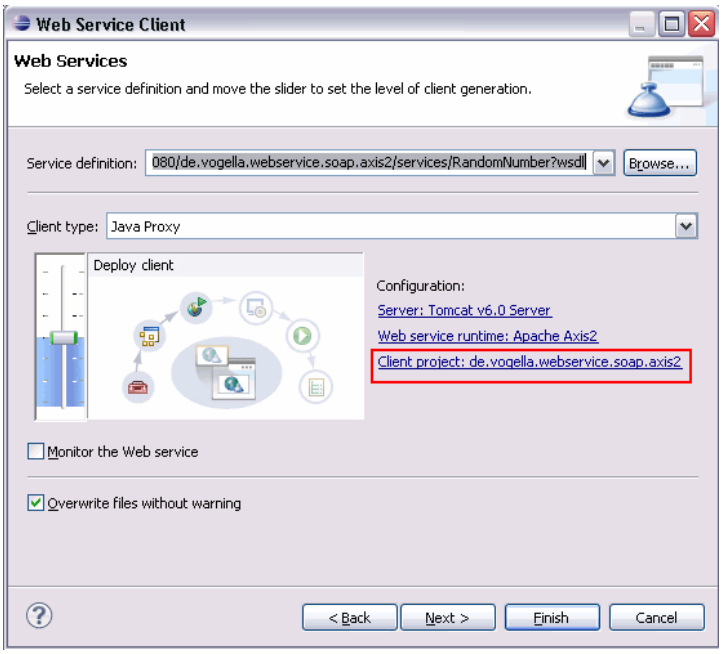
Right click on your project and select New -> Other… -> Web Services -> Web Service Client. We will now create a client. Through a setting later we will later on create the client in a separate project.



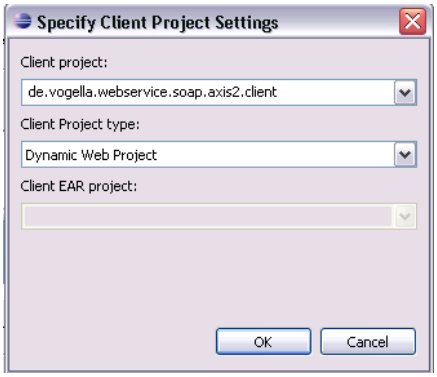
Click "Next".

Paste your copied service URL into the "Service definition" field.

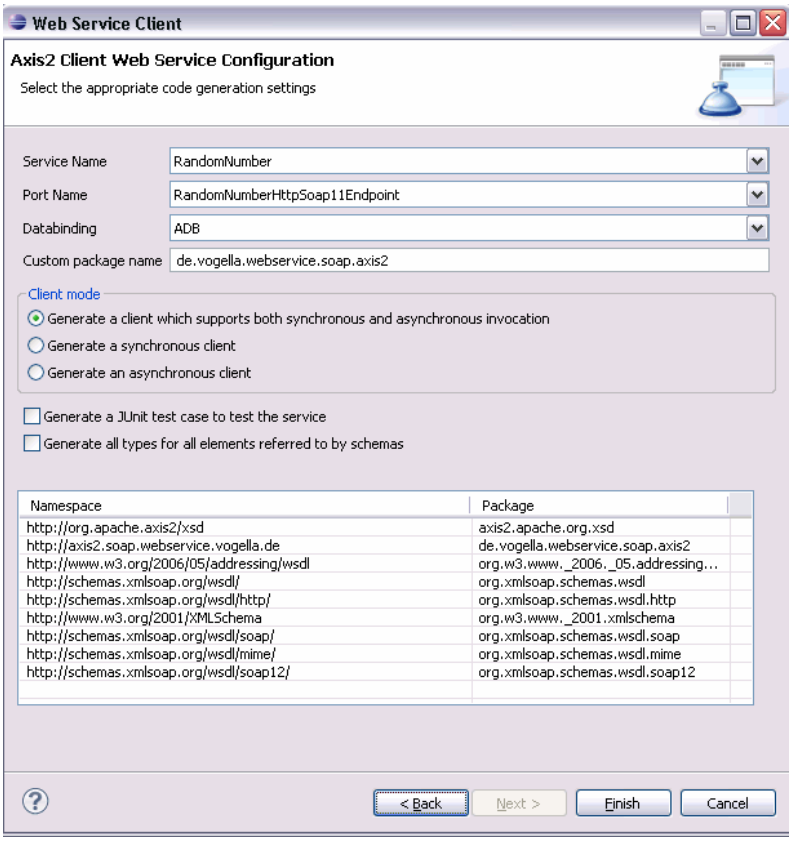
> Make sure your server and service are running, otherwise the wizard won't accept the URL!



Open the link "Client project: ..." and enter "de.vogella.webservice.soap.axis2.client" to create a new project for your client.



Click "OK" and "Next".



Click "Finish".

You have generated a Web Service Client!

3.5. Test the webservice

Now we are going to test our Web Service using this client. So we create a test class in "de.vogella.webservice.soap.axis2.client" project. Select your package "de.vogella.webservice.soap.axis2.client". Create the following class ServiceTest.java.

```
package de.vogella.webservice.soap.axis2;

import java.rmi.RemoteException;

import de.vogella.webservice.soap.axis2.RandomNumberStub.GetNumberResponse;

public class ServiceTest {
        public static void main(String[] args) throws RemoteException   {
                RandomNumberStub stub = new RandomNumberStub();
                GetNumberResponse res = stub.getNumber();
        System.out.println("Counter: "+res.get_return());
        }
}
```

Right click "ServiceTest.java" -> Run As -> Java Application. You should see the result of your call in the command line.

4. Create a complex Web Service

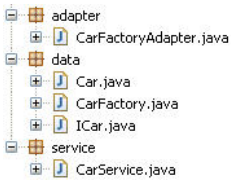
4.1. Overview

After we created a rather simple Web Service, we will create a complex one. Complex means, that it will use Java Interfaces and Collections.

The first step is to create a "Dynamic Web Project" (e.g. "ComplexWebService") like it is described here .

4.2. Datamodel

We need an Interface "ICar" and an implementing class "Car". The "Car" objects will be managed in the "CarFactory" class. That means, "CarFactory" has a Collection of "Cars". But Axis2 cannot handle Java Collections, so we need an Adapter class ("CarFactoryAdapter") to transform our Collection to an Array. The last thing we need is our "CarService" class. Your package structure should look like this:



Make sure that "ICar" and "Car" are in the same package, otherwise there will be a problem with the namespace.

```
package data;

public interface ICar {

        public int getMaxSpeed();

        public String getModel();

        public void setMaxSpeed(int maxSpeed);

        public void setModel(String model);
}
```

```
package data;

public class Car implements ICar {

        private String model;
        private int maxSpeed;

        public Car(String model, int maxSpeed)
        {
                this.model = model;
                this.maxSpeed = maxSpeed;
        }

        public String getModel() {
                return model;
        }

        public void setModel(String model) {
                this.model = model;
        }

        public int getMaxSpeed() {
                return maxSpeed;
        }

        public void setMaxSpeed(int maxSpeed) {
                this.maxSpeed = maxSpeed;
        }
}
```

```
package data;

import java.util.Vector;

public class CarFactory {

        private Vector<ICar>    cars    = new Vector<ICar>();

        public Vector<ICar> getCars() {
                return cars;
        }

        public void setCars(Vector<ICar> cars) {
                this.cars = cars;
        }
}
```

```
package adapter;

import java.util.Vector;

import data.Car;
import data.CarFactory;
import data.ICar;

public class CarFactoryAdapter {

        private ICar[] cars;
```

```
                cars = carList.toArray(new Car[carList.size()]);
        }

        public ICar[] getCars() {
                return cars;
        }

        public void setCars(ICar[] cars) {
                this.cars = cars;
        }

}
```

```
package service;

import adapter.CarFactoryAdapter;
import data.Car;
import data.CarFactory;

public class CarService {

        private CarFactory carFact = new CarFactory();

        public CarFactoryAdapter getCarFactory()
        {
                carFact.getCars().add(new Car("BMW", 250));
                carFact.getCars().add(new Car("Porsche", 300));
                carFact.getCars().add(new Car("Maserati", 350));

                return new CarFactoryAdapter(carFact);
        }
}
```

4.3. Create and test the Web Service

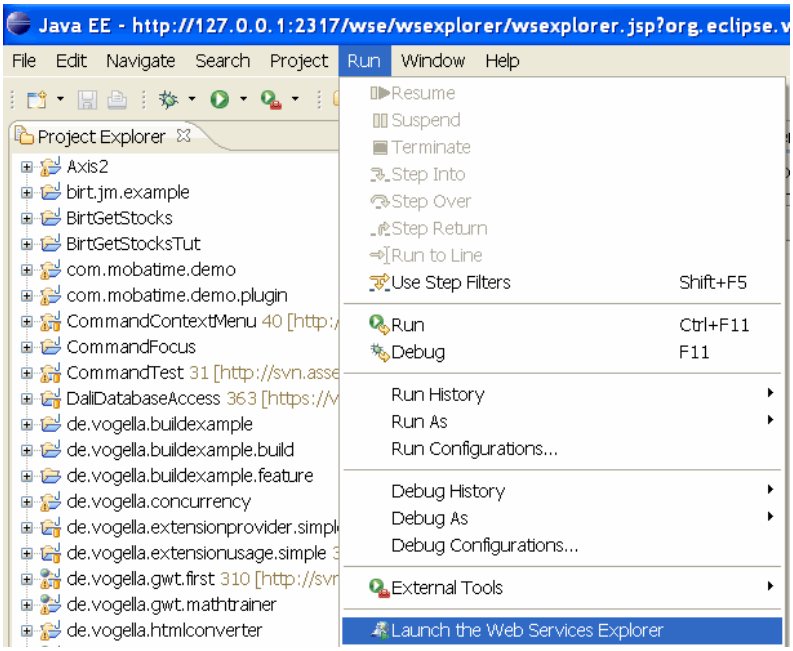
Create a Web Service for the CarService class, create a webservice client and a test as described Create a first Webservice .

The Test class looks like the following.

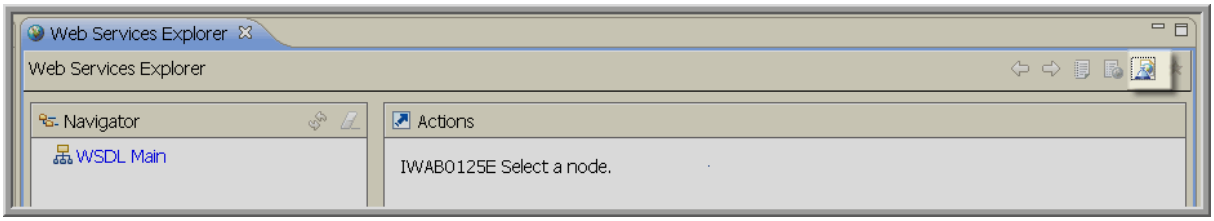
```
package service;

import java.rmi.RemoteException;

import service.CarServiceStub.CarFactoryAdapter;
import service.CarServiceStub.GetCarFactoryResponse;
import service.CarServiceStub.ICar;

public class Test {

        public static void main(String[] args) throws RemoteException {
                CarServiceStub stub = new CarServiceStub();

                GetCarFactoryResponse res = stub.getCarFactory();
                CarFactoryAdapter carFact = res.get_return();
                ICar[] cars = carFact.getCars();

                for (int i = 0; i < cars.length; i++) {
                        System.out.println(cars.clone()[i].getModel());
                        System.out.println(cars.clone()[i].getMaxSpeed());
                        System.out.println("------------------------------");
                }
        }
}
```

5. Test a Web Service

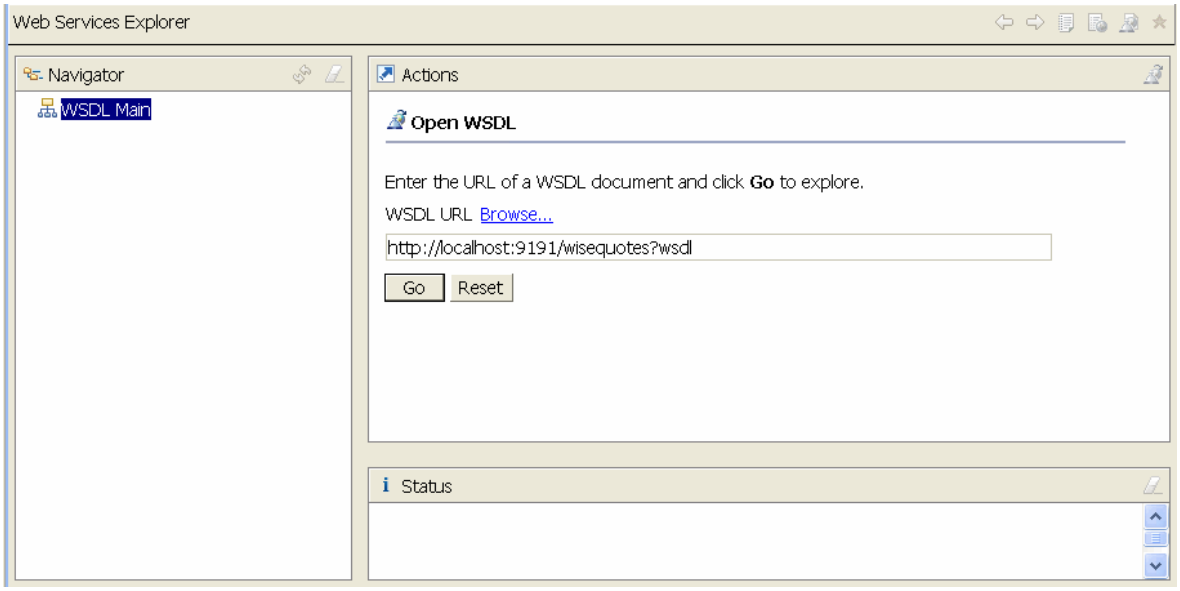
Eclipse allows to test webservices. In the Java EE perspective select Run-> Launch the Webservice Explorer



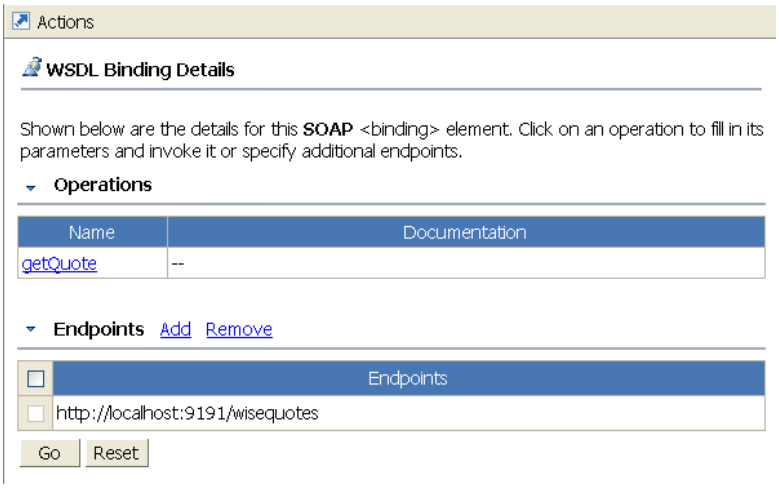
Open the WSDL page.

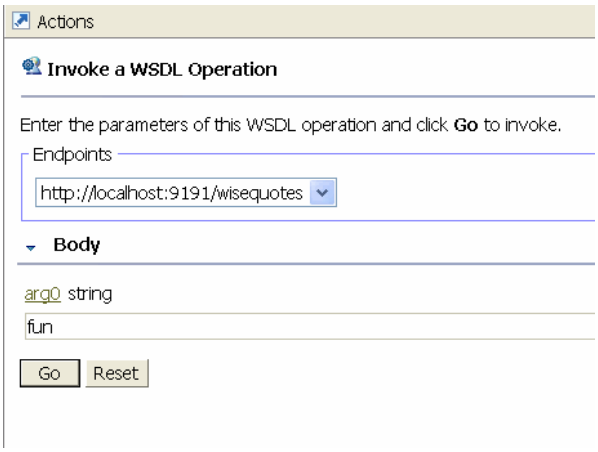


Double-click on the WSDL Main and maintain the URL to your webservice.



Press go. The system will list your operations. Click on them to test the. Here you can maintain your parameter, test and see the return parameter.





6. Thank you

Thank you for practicing with this tutorial.

If you like the information of this article, please help me by donating.

7. Questions and Discussion

For questions and discussion around this article please use the www.vogella.de Google Group . Also if you note an error in this article please post the error and if possible the correction to the Group.

> The following tries to help you in asking good questions: 10 golden rules of asking questions in the OpenSource community .

8. Links and Literature

8.1. Source Code

Source Code of Examples

8.2. Web development resources

http://www.ibm.com/developerworks/edu/os-dw-os-eclipse-europa1.html Web development with Eclipse Europa, Part 1: The Java EE for Eclipse

8.3. vogella Resources

Eclipse Tutorials

Web development Tutorials

Android Development Tutorial

GWT Tutorial

Eclipse RCP Tutorial