



# JAVA SE WEB SERVICES

**Create, consume and  
deploy basic web  
services productively  
using NetBeans**

Milan Kuchtiak

This article illustrates how a web service can be created and tested as a NetBeans standard Java Project. The new NetBeans 5.5 integrated support for JAX-WS 2.0 enables you to easily create and consume web services.

## **Creating the web service**

Start NetBeans 5.5, then create a new Java Application project and name it "GeometricalWS". In the project's *Properties* dialog select the *Libraries* category and add "JAX-WS 2.0" (see **Figure 1**). This step

is necessary only if you're using Java SE 5 or lower. Java SE 6 already includes the JAX-WS APIs.

Now create a Java class named "CircleFunctions" with the code shown in **Listing 1**. The `@WebService` annotation makes the class a web service. The other annotations declare the web service's operations and their parameters, influencing the automatic generation of a WSDL document for this class.

Using the `javax.xml.ws.Endpoint.publish()` method, the web service can be deployed to a simple web server provided by the JAX-WS runtime. Update the project's **Main**

class with the code from **Listing 2**. Notice that the `publish()` method requires the URL address for the web service and an instance of the `CircleFunction` class. The latter will be invoked to serve requests.

Run the application. The message in the output window will notify you that the web service was published successfully:

```
Web service was published successfully.
WSDL URL: http://localhost:8765/GeometricalWS/CircleFunctions?WSDL
```

To check that the web service was really published, launch your web browser and open the web service URL: `http://localhost:8765/GeometricalWS/CircleFunctions?WSDL`. The browser should show a WSDL file.

## Steps to create a Client

Developing a web service client with NetBeans 5.5 is even simpler.

### Listing 1. The CircleFunctions class, a full web service implementation.

```
package geometricalws;

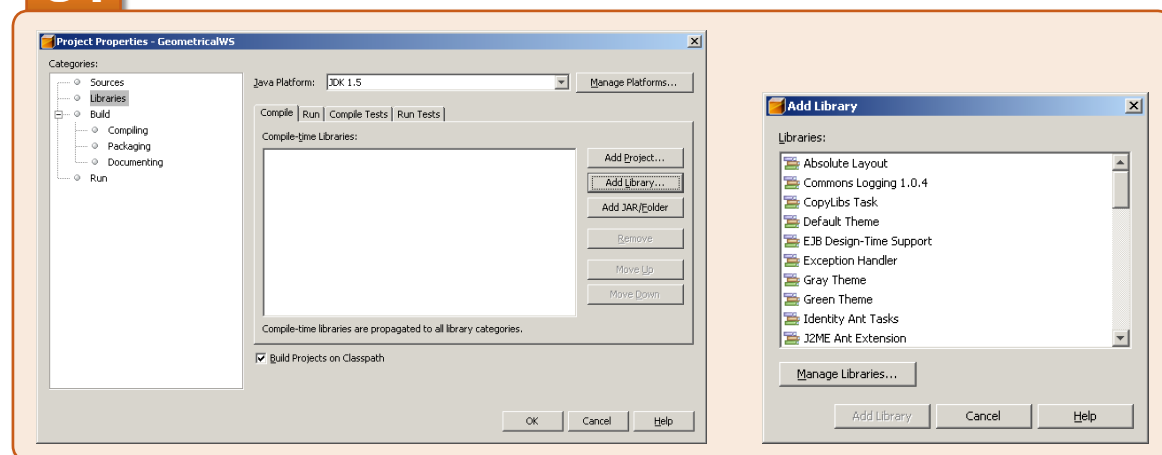
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService(name="Circle", serviceName="CircleService", portName="CirclePort")
@SOAPBinding(style=SOAPBinding.Style.RPC)
public class CircleFunctions {

    @WebMethod(operationName="area")
    public double getArea(@WebParam(name="r") double r) {
        return Math.PI * (r * r);
    }

    @WebMethod(operationName="circumference")
    public double getCircumference(@WebParam(name="r") double r) {
        return 2 * Math.PI * r;
    }
}
```

1



**Figure 1.** Adding JAX-WS support to a common Java project (only necessary with Java SE 5).

[blogs.sun.com/pblaha/entry/developing\\_web\\_services\\_for\\_mustang](http://blogs.sun.com/pblaha/entry/developing_web_services_for_mustang)

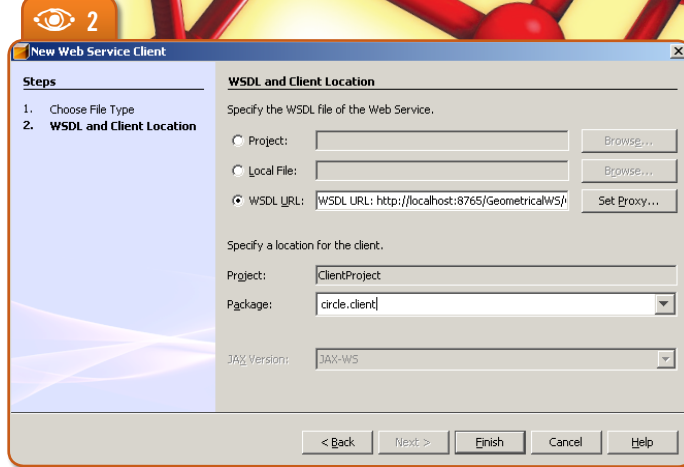
Petr Blaha's blog: Developing Web services for Mustang in Netbeans.

[java.sun.com/javase/5-downloads/index.jsp](http://java.sun.com/javase/5-downloads/index.jsp)

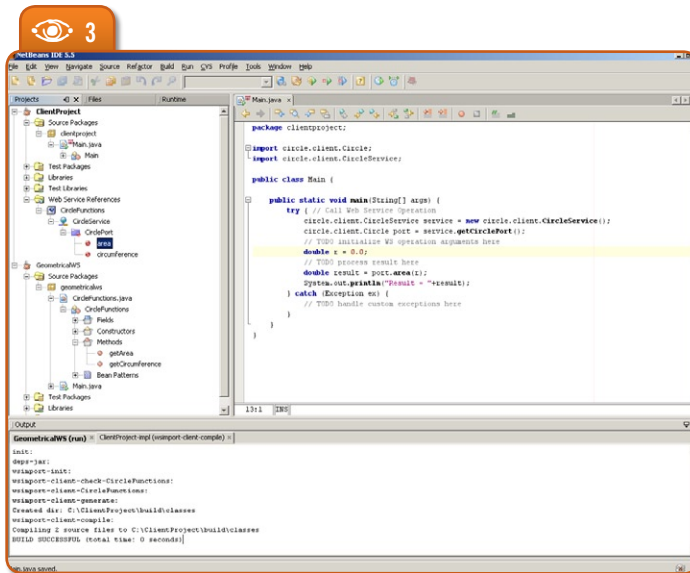
Java EE 5 SDK/SJSAS, Java EE 5's Reference Implementation.



**Figure 2.** Creating a web service client from its live WSDL document.



**Figure 3.** Creating the web service client invocation interactively.



**Listing 2.** The Main class, a full web service server.

```
package geometricalws;

import javax.xml.ws.Endpoint;

public class Main {
    public static void main (String[] args) {
        String wsAddress = "http://localhost:8765/GeometricalWS/CircleFunctions";
        Endpoint.publish(wsAddress, new CircleFunctions());
        System.out.println("Web service was published successfully.\n"+
            "WSDL URL: " + wsAddress + "?WSDL");

        // Keep the local web server running until the process is killed
        while (Thread.currentThread().isAlive()) try {
            Thread.sleep(10000);
        } catch (InterruptedException ex) {}
    }
}
```

Create another Java Application project and name it (say) “ClientProject”. If you’re using Java SE 5, add the JAX-WS 2.0 library to the project, as before.

Right-click on the project and choose *New>Web Service Client*. Then fill the *WSDL URL* field with the URL for the web service we just published (see **Figure 2**). Also set the package name for the client artifacts (these are Java classes which will be generated from the WSDL); I used “*circle.client*”. Click *Finish*, and a node named *CircleFunctions* should be created under *Web Service References*.

Open *Main.java* in the source editor, expand *Web Service References* and locate the node *CircleFunctions-CircleService-CirclePort-area*. Then drag it to the editor inside the *main()* method in the *Main* class. NetBeans will generate code that invokes that operation (see **Figure 3**). Next, change the value for the web service operation argument (*r*). **Listing 3** contains the finished source (after tweaking the generated code a little). Finally, run the client project. For *r = 10.0*, the following message should appear in the output:

Robert Eckstein and Rajiv Mordani’s article about JAX-WS 2.0 With the Java SE 6 Platform.

[java.sun.com/developer/technicalArticles/J2SE/jax\\_ws\\_2](http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2)

Article comparing various (SOAP) binding styles.

[www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/](http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/)

Result = 314.1592653589793

## Deploying the service in a Java EE container

As we've seen up to now, a Java EE application server is not required to bring up a web service. But what if we had to make the service available in a Java EE container? We can just create a "wrapper" Web application that reuses all code written for our Java SE app.

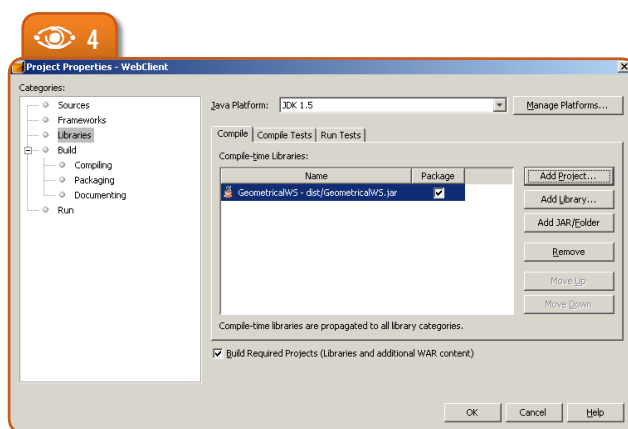
Start by creating a Web app in NetBeans: *File|New Project>Web>Web Application*. Then set its target server to Sun Java System Application Server (if asked). Go to *Properties>Libraries*, click *Add Project* and select the root directory of the *GeometricalWS* project, as in **Figure 4**.

Deploy the web application to the container, with the project's *Deploy Project* action. If you open SJSAS's admin console, the *Circle* web service should be among the listed web services (**Figure 5**).

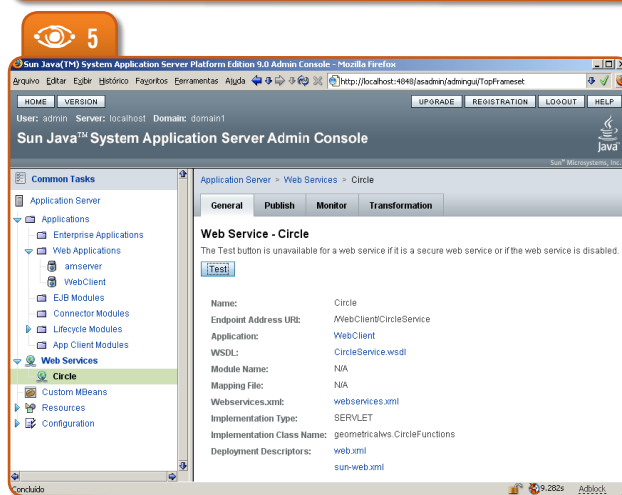
The *Test* button allows you to test the web service from the admin console (**Figure 6**). After invoking an operation, you can see its request and response.

## Conclusions

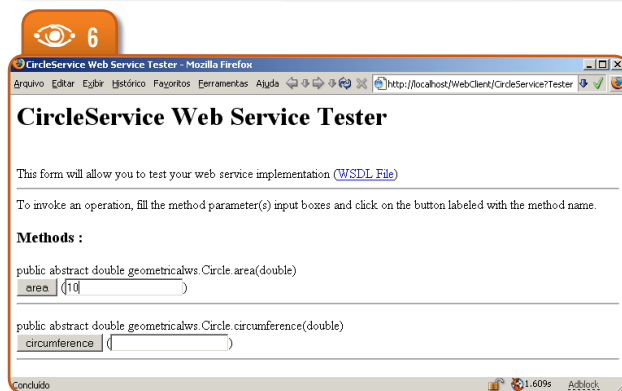
In this tutorial, we created a web service with just a few lines of web-service specific code (like annotations in the service implementation and publishing in the main class). Then we created a client without manually writing any code – just by using NetBeans' wizards and drag-and-drop features. Finally, we deployed the same web service in a Java EE server, again not having to write additional code. This shows how you can develop web services that are reusable both in Java SE and Java EE environments. ☒



**Figure 4.** Creating a Web application that depends on the web service project.



**Figure 5.** Inspecting the web service, deployed as a Java EE web application in SJSAS.



**Figure 5.** Testing the web service with SJSAS's automatic test page.

### Listing 3. The client project's Main class.

```
package clientproject;

import circle.client.Circle;
import circle.client.CircleService;

public class Main {
    public static void main(String[] args) {
        try { // Call Web Service Operation
            Circle port =
                new CircleService().getCirclePort();
            double r = 10.0;
            double result = port.area(r);
            System.out.println("Result = "+ result);
        } catch (Exception ex) {
            // TODO handle custom exceptions here
        }
    }
}
```



**Milan Kuchtiak**  
(milan.kuchtiak@sun.com) has worked six years for Sun Microsystems in the NetBeans team, currently on the IDE support for Web application development and Web Services (JAX-RPC and JAX-WS).