Getting Started with JAX-WS Web Services

Java API for XML Web Services (JAX-WS), JSR 224, is an important part of the Java EE 5 and EE 6 platforms. A follow-up to the release of Java API for XML-based RPC 1.1(JAX-RPC), JAX-WS simplifies the task of developing web services using Java technology. It addresses some of the issues in JAX-RPC 1.1 by providing support for multiple protocols such as SOAP 1.1, SOAP 1.2, XML, and by providing a facility for supporting additional protocols along with HTTP. JAX-WS uses JAXB 2.0 for data binding and supports customizations to control generated service endpoint interfaces. With its support for annotations, JAX-WS simplifies web service development and reduces the size of runtime JAR files.

This document demonstrates the basics of using the IDE to develop a JAX-WS web service. After you create the web service, you write three different web service clients that use the web service over a network, which is called "consuming" a web service. The three clients are a Java class in a Java SE application, a servlet, and a JSP page in a web application. A more advanced tutorial focusing on clients is Developing JAX-WS Web Service Clients.

Contents

To follow this tutorial, you need the following software and resources.

Software or Resource	Version Required
NetBeans IDE	Java EE download bundle
Java Development Kit (JDK)	JDK 6 or JDK 7
Java EE-compliant web or application server	Tomcat web server 6.x-7.0 GlassFish Server Open Source Edition Oracle WebLogic Server

Note: Both Tomcat and the GlassFish server can be installed with the Web and Java EE distribution of NetBeans IDE. Alternatively, you can visit the the GlassFish server downloads page or the Apache Tomcat downloads page.

Important: Java EE 6 projects require Tomcat 7.x, GlassFish Server 3.x, or Oracle WebLogic Server 12c.

Creating a Web Service

The goal of this exercise is to create a project appropriate to the deployment container that you decide to use. Once you have a project, you will create a web service in it.

Choosing a Container

You can either deploy your web service in a web container or in an EJB container. This depends on your choice of implementation. If you are creating a Java EE 6 application, use a web container in any case, because you can put EJBs directly in a web application. For example, if you plan to deploy to the Tomcat Web Server, which only has a web container, create a web application, not an EJB module.

1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows, ⌘-Shift-N on MacOS). Select Web Application from the Java Web category or EJB Module from the Java EE category.

   You can create a JAX-WS web service in a Maven project. Choose File > New Project (Ctrl-Shift-N on Linux and Windows, ⌘-Shift-N on MacOS) and then Maven Web Application or Maven EJB module from the Maven category. If you haven't used Maven with NetBeans before, see Maven Best Practices.

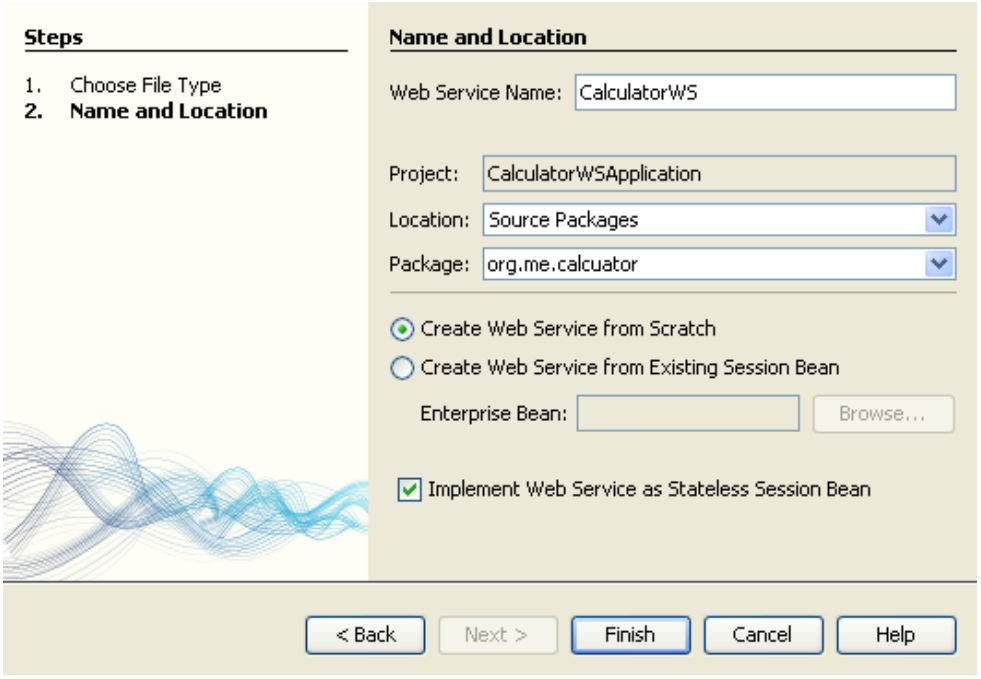
2. Name the project `CalculatorWSApplication`. Select a location for the project. Click Next.
3. Select your server and Java EE version and click Finish.

To use the Oracle WebLogic server, register the server with the IDE. Also, if you are using the WebLogic server, watch the screencast on Deploying a Web Application to Oracle WebLogic.

Creating a Web Service from a Java Class

1. Right-click the `CalculatorWSApplication` node and choose New > Web Service.
2. Name the web service `CalculatorWS` and type `org.me.calculator` in Package. Leave Create Web Service from Scratch selected.
3. If you are creating a Java EE 6 project on GlassFish or WebLogic, select Implement Web Service as a Stateless Session Bean.

   By default, Tomcat web server does not support Enterprise Java Beans (EJBs). Therefore you cannot deploy an EE6 web service as a stateless session bean on an out-of-the-box installation of Tomcat 7. However, see the Apache TomEE project (especially OpenEJB) for extensions to Tomcat.



4. Click Finish. The Projects window displays the structure of the new web service and the source code is shown in the editor area.

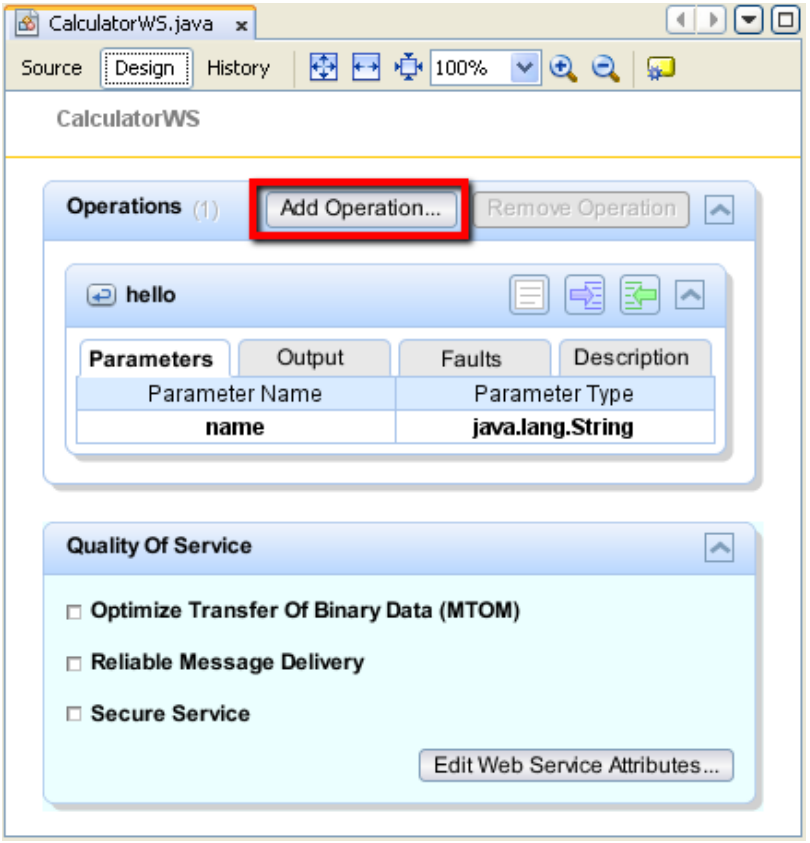
Adding an Operation to the Web Service

The goal of this exercise is to add to the web service an operation that adds two numbers received from a client. The NetBeans IDE provides a dialog for adding an operation to a web service. You can open this dialog either in the web service visual designer or in the web service context menu.

Warning: The visual designer is not available in Maven projects.

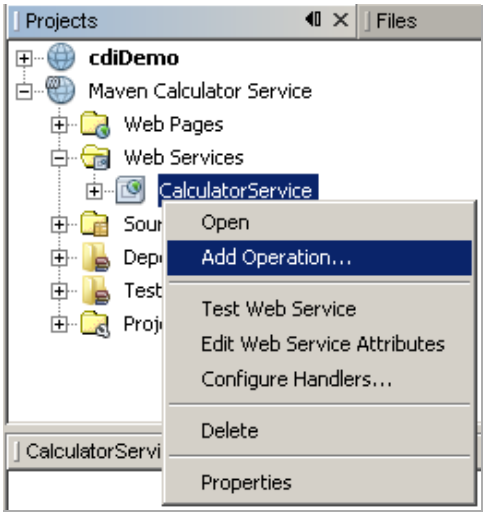
To add an operation to the web service:

1. Either:
   - Change to the Design view in the editor.

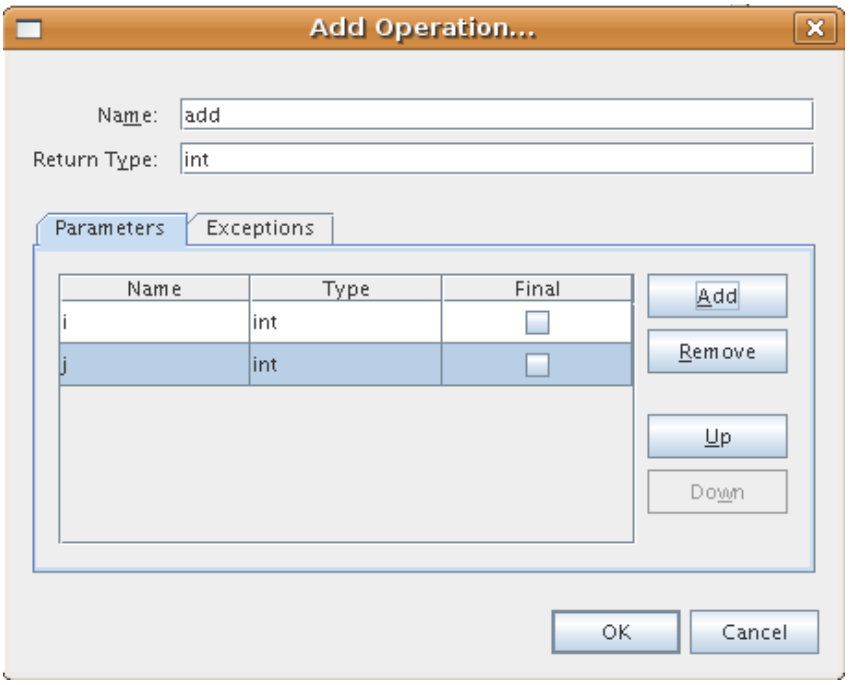


Or:

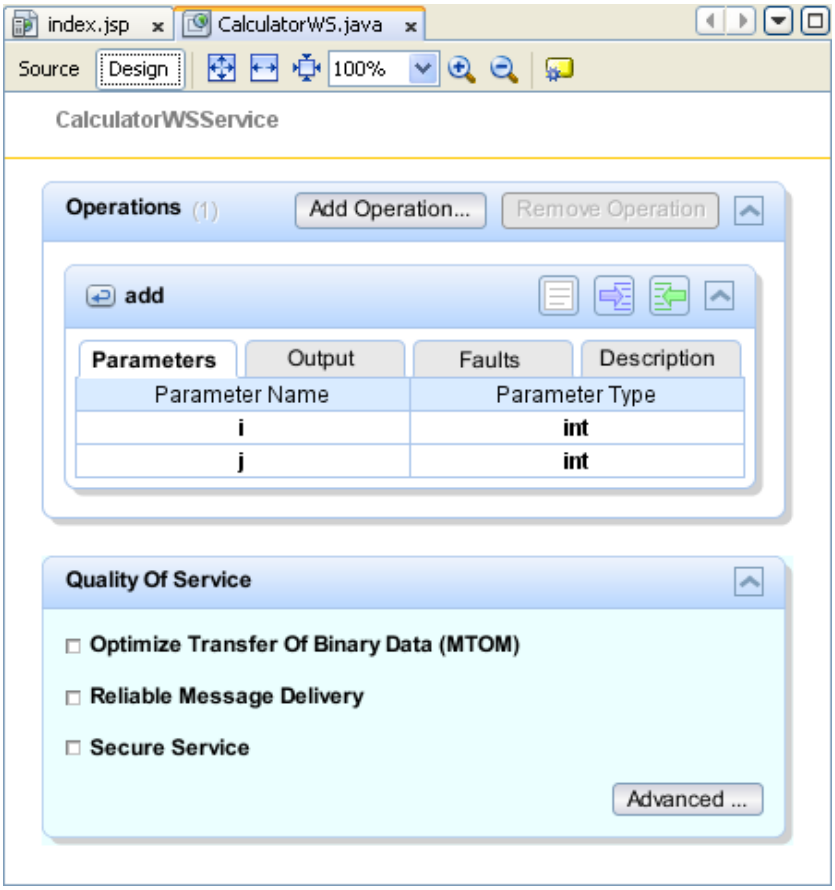
- Find the web service's node in the Projects window. Right-click that node. A context menu opens.



2. Click Add Operation in either the visual designer or the context menu. The Add Operation dialog opens.
3. In the upper part of the Add Operation dialog box, type `add` in Name and type `int` in the Return Type drop-down list.
4. In the lower part of the Add Operation dialog box, click Add and create a parameter of type `int` named `i`.
5. Click Add again and create a parameter of type `int` called `j`.
   You now see the following:



6. Click OK at the bottom of the Add Operation dialog box. You return to the editor.
7. Remove the default `hello` operation, either by deleting the `hello()` method in the source code or by selecting the `hello` operation in the visual designer and clicking Remove Operation.
   The visual designer now displays the following:



8. Click Source and view the code that you generated in the previous steps. It differs whether you created the service as an EE6 stateless bean or not. Can you see the difference in the screenshots below? (An EE6 service that is not implemented as a stateless bean resembles an EE5 service.)

```
package org.me.calculator;

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

/**
 *
 * @author gw152771
 */
@WebService()
public class CalculatorWS {

/**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i")
    int i, @WebParam(name = "j")
    int j) {
//TODO write your implementation code here:
        return 0;
    }

}
```

```
package org.me.calcuator;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.ejb.Stateless;

/**
 *
 * @author jeff
 */
@WebService()
@Stateless()
public class CalculatorWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i")
    int i, @WebParam(name = "j")
    int j) {
        //TODO write your implementation code
        return 0;
    }
}
```

9. In the editor, extend the skeleton `add` operation to the following (changes are in bold):

```
@WebMethod
public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {
    int k = i + j;
    return k;
  }
```

As you can see from the preceding code, the web service simply receives two numbers and then returns their sum. In the next section, you use the IDE to test the web service.

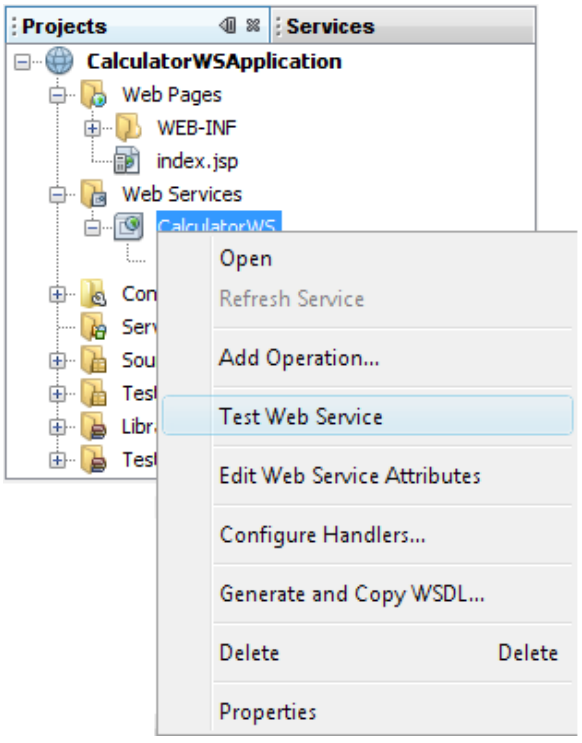
Deploying and Testing the Web Service

After you deploy a web service to a server, you can use the IDE to open the server's test client, if the server has a test client. The GlassFish and WebLogic servers provide test clients.

If you are using the Tomcat Web Server, there is no test client. You can only run the project and see if the Tomcat Web Services page opens. In this case, before you run the project, you need to make the web service the entry point to your application. To make the web service the entry point to your application, right-click the CalculatorWSApplication project node and choose Properties. Open the Run properties and type `/CalculatorWS` in the Relative URL field. Click OK. To run the project, right-click the project node again and select Run.

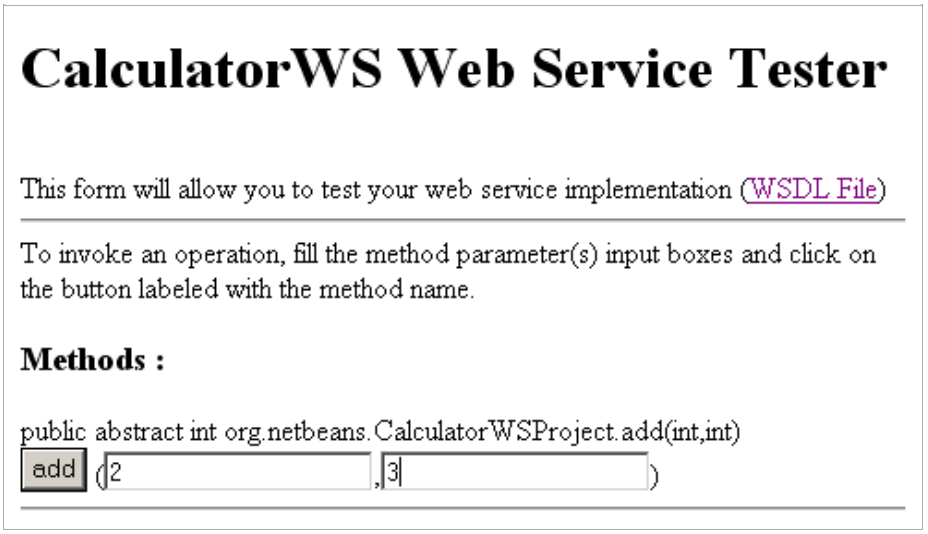
To test successful deployment to a GlassFish or WebLogic server:

1. Right-click the project and choose Deploy. The IDE starts the application server, builds the application, and deploys the application to the server. You can follow the progress of these operations in the CalculatorWSApplication (run-deploy) and the GlassFish server or Tomcat tabs in the Output view.
2. In the IDE's Projects tab, expand the Web Services node of the CalculatorWSApplication project. Right-click the CalculatorWS node, and choose Test Web Service.

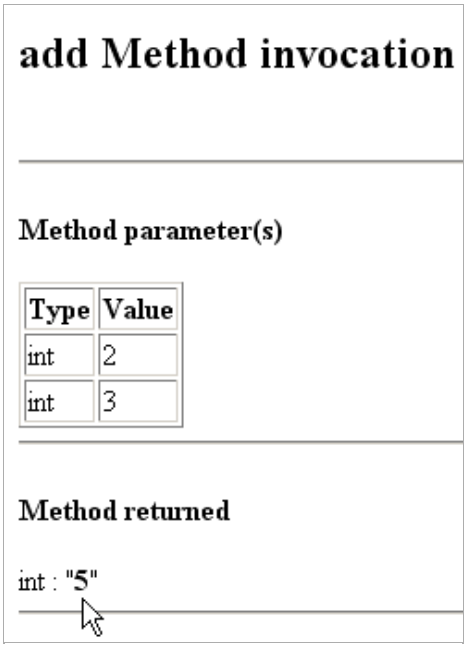


The IDE opens the tester page in your browser, if you deployed a web application to the GlassFish server. For the Tomcat Web Server and deployment of EJB modules, the situation is different:

- If you deployed to the GlassFish server, type two numbers in the tester page, as shown below:



The sum of the two numbers is displayed:



Samples

You can open a complete EE6 stateless bean version of the Calculator service by choosing File > New Project (Ctrl-Shift-N on Linux and Windows, ⌘-Shift-N on MacOS) and navigating to Samples > Java Web Services > Calculator (EE6).

A Maven Calculator Service and a Maven Calculator Client are available in Samples > Maven.

Consuming the Web Service

Now that you have deployed the web service, you need to create a client to make use of the web service's `add` method. Here, you create three clients— a Java class in a Java SE application, a servlet, and a JSP page in a web application.

Note: A more advanced tutorial focusing on clients is Developing JAX-WS Web Service Clients.

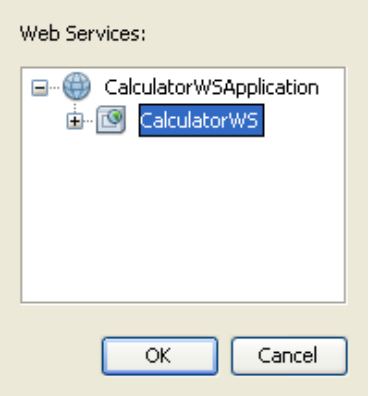
Client 1: Java Class in Java SE Application

In this section, you create a standard Java application. The wizard that you use to create the application also creates a Java class. You then use the IDE's tools to create a client and consume the web service that you created at the start of this tutorial.

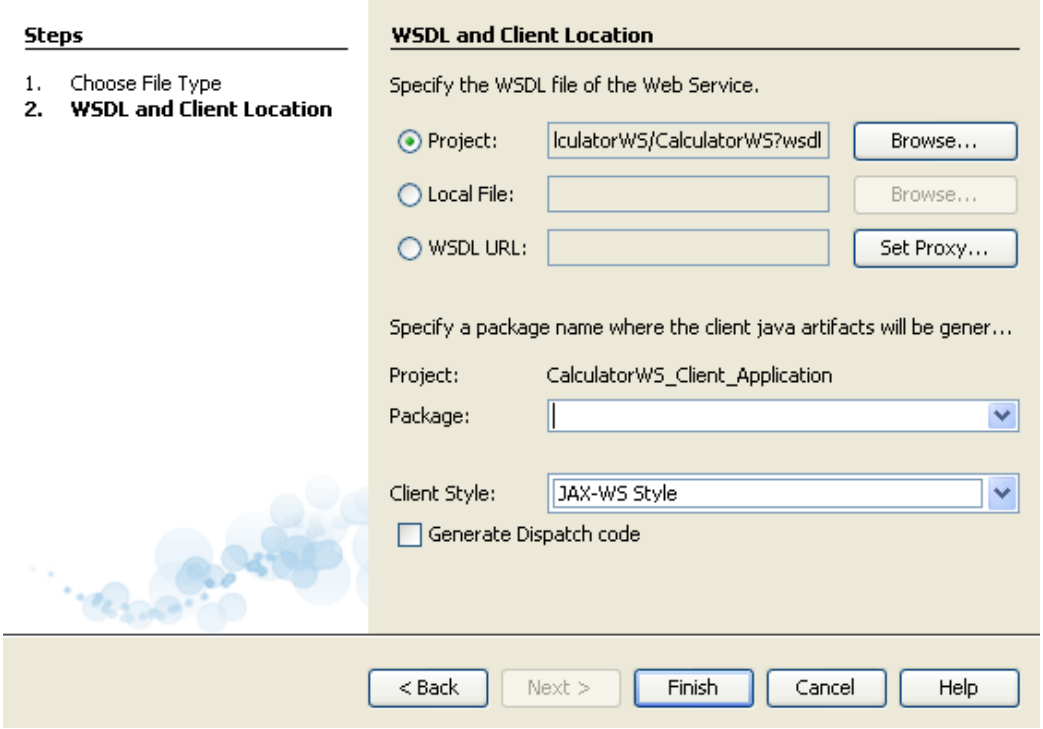
1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows, ⌘-Shift-N on MacOS). Select Java Application from the Java category. Name the project `CalculatorWS_Client_Application`. Leave Create Main Class selected and accept all other default settings. Click Finish.
2. Right-click the `CalculatorWS_Client_Application` node and choose New > Web Service Client. The New Web

Service Client wizard opens.

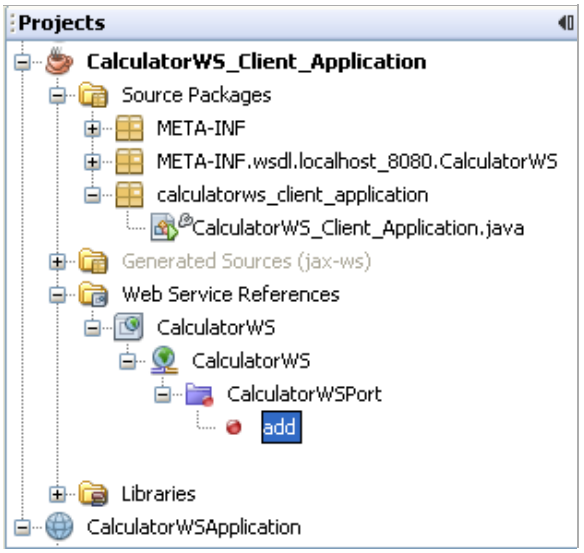
3. Select Project as the WSDL source. Click Browse. Browse to the CalculatorWS web service in the CalculatorWSApplication project. When you have selected the web service, click OK.



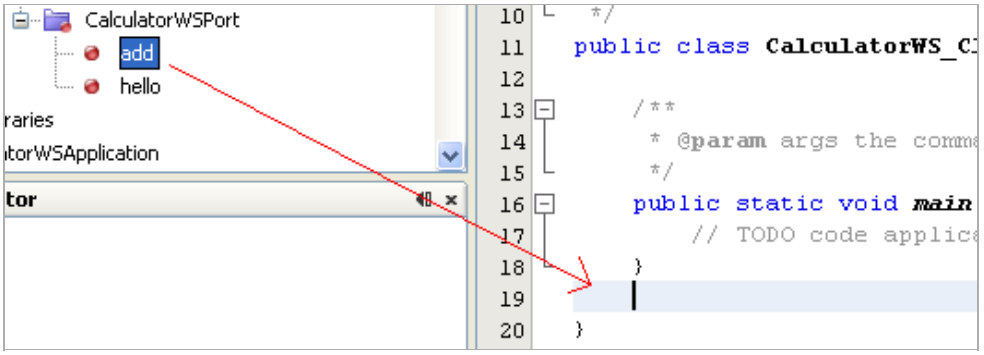
4. Do not select a package name. Leave this field empty.



5. Leave the other settings at default and click Finish.
The Projects window displays the new web service client, with a node for the `add` method that you created:



6. Double-click your main class so that it opens in the Source Editor. Drag the `add` node below the `main()` method.



You now see the following:

```
public static void main(String[] args) {
    // TODO code application logic here
}
private static int add(int i, int j) {
    org.me.calculator.CalculatorWS_Service service = new
org.me.calculator.CalculatorWS_Service();
    org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
    return port.add(i, j);
}
```

Note: Alternatively, instead of dragging the `add` node, you can right-click in the editor and then choose Insert Code > Call Web Service Operation.

7. In the `main()` method body, replace the TODO comment with code that initializes values for `i` and `j`, calls `add()`, and prints the result.

```
public static void main(String[] args) {
    int i = 3;
    int j = 4;
    int result = add(i, j);
    System.out.println("Result = " + result);
}
```

8. Surround the `main()` method code with a try/catch block that prints an exception.

```
public static void main(String[] args) {
    try {
        int i = 3;
        int j = 4;
        int result = add(i, j);
        System.out.println("Result = " + result);
    } catch (Exception ex) {
        System.out.println("Exception: " + ex);
    }
}
```

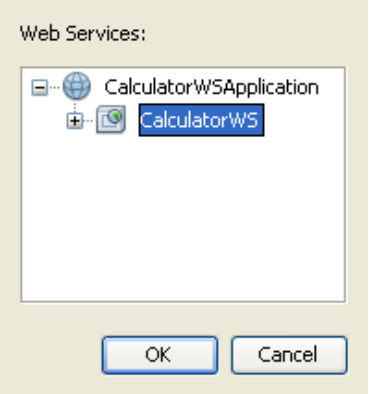
9. Right-click the project node and choose Run.
The Output window now shows the sum:

```
compile:
run:
Result = 7
  BUILD SUCCESSFUL (total time: 1 second)
```

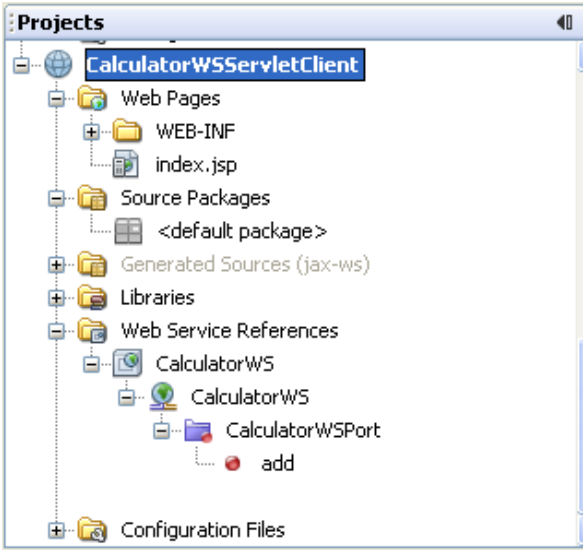
Client 2: Servlet in Web Application

In this section, you create a new web application, after which you create a servlet. You then use the servlet to consume the web service that you created at the start of this tutorial.

1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows, ⌘-Shift-N on MacOS). Select Web Application from the Java Web category. Name the project `CalculatorWSServletClient`. Click Next and then click Finish.
2. Right-click the `CalculatorWSServletClient` node and choose New > Web Service Client.
   The New Web Service Client wizard appears.
3. Select Project as the WSDL source. Click Browse. Browse to the CalculatorWS web service in the CalculatorWSApplication project. When you have selected the web service, click OK.



4. Do not select a package name. Leave this field empty.
5. Leave the other settings at default and click Finish.
   The Web Service References node in the Projects window displays the structure of your newly created client, including the `add` operation that you created earlier in this tutorial:



6. Right-click the `CalculatorWSServletClient` project node and choose New > Servlet. Name the servlet `ClientServlet` and place it in a package called `org.me.calculator.client`. Click Finish.
7. To make the servlet the entry point to your application, right-click the CalculatorWSServletClient project node and choose Properties. Open the Run properties and type `/ClientServlet` in the Relative URL field. Click OK.
8. If there are error icons for `ClientServlet.java`, right-click the project node and select Clean and Build.
9. In the Source Editor, drag the `add` operation anywhere in the body of the `ClientServlet` class. The `add()` method appears at the end of the class code.

   Note: Alternatively, instead of dragging the `add` node, you can right-click in the editor and then choose Insert Code > Call Web Service Operation.

```
private int add(int i, int j) {
    org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
```

```
        return port.add(i, j);
    }
```

10. In the `processRequest()` method, add some empty lines after this line:

```
        out.println("<h1>Servlet ClientServlet at " + request.getContextPath () + "</h1>");
```

11. Add code that initializes values for `i` and `j`, calls `add()`, and prints the result. The added code is in **boldface**:

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ClientServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet ClientServlet at " + request.getContextPath () + "</h1>");

            int i = 3;
            int j = 4;
            int result = add(i, j);
            out.println("Result = " + result);

            out.println("</body>");
            out.println("</html>");

        } finally {
            out.close();
        }
    }
```

12. Surround the added code with a try/catch block that prints an exception.

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ClientServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet ClientServlet at " + request.getContextPath () + "</h1>");
            try {
                int i = 3;
                int j = 4;
                int result = add(i, j);
                out.println("Result = " + result);
            } catch (Exception ex) {
                out.println("Exception: " + ex);
            }
            out.println("</body>");
            out.println("</html>");

        } finally {
            out.close();
        }
```

```
    }
```

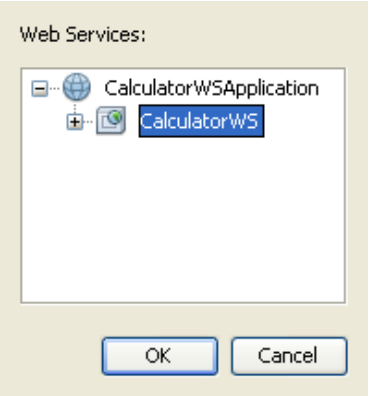
13. Right-click the project node and choose Run.
    The server starts, the application is built and deployed, and the browser opens, displaying the calculation result, as shown below:

Client 3: JSP Page in Web Application

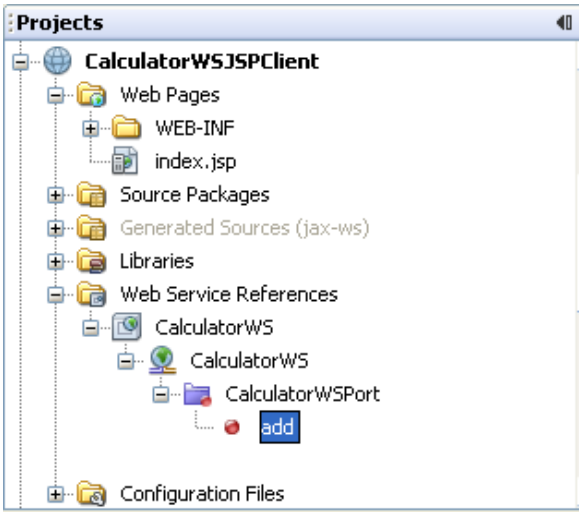
In this section, you create a new web application and then consume the web service in the default JSP page that the Web Application wizard creates.

> **Note:** If you want to run a JSP web application client on Oracle WebLogic, see Running a Java Server Faces 2.0 Application on WebLogic.

1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows, ⌘-Shift-N on MacOS). Select Web Application from the Java Web category. Name the project `CalculatorWSJSPClient`. Click Finish.
2. Right-click the `CalculatorWSJSPClient` node and choose New > Web Service Client.
3. Select Project as the WSDL source. Click Browse. Browse to the CalculatorWS web service in the CalculatorWSApplication project. When you have selected the web service, click OK.



4. Do not select a package name. Leave this field empty.
5. Leave the other settings at default and click Finish.
   The Projects window displays the new web service client, as shown below:



6. In the Web Service References node, expand the node that represents the web service. The `add` operation, which you will invoke from the client, is now exposed.
7. Drag the `add` operation to the client's `index.jsp` page, and drop it below the H1 tags. The code for invoking the service's operation is now generated in the `index.jsp` page, as you can see here:

```
<%
try {
    org.me.calculator.CalculatorWSService service = new org.me.calculator.CalculatorWSService();
    org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
     // TODO initialize WS operation arguments here
    int i = 0;
    int j = 0;
    // TODO process result here
    int result = port.add(i, j);
    out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
```

   Change the value for `i` and `j` from 0 to other integers, such as 3 and 4. Replace the commented out TODO line in the catch block with `out.println("exception" + ex);`.
8. Right-click the project node and choose Run.
   The server starts, if it wasn't running already. The application is built and deployed, and the browser opens, displaying the calculation result:

Send Feedback on This Tutorial

See Also

For more information about using NetBeans IDE to develop Java EE applications, see the following resources:

- Developing JAX-WS Web Service Clients

- Getting Started with RESTful Web Services
- Binding WSDL to Java with JAXB
- Advanced Web Service Interoperability
- Web Services Learning Trail

To send comments and suggestions, get support, and keep informed about the latest developments on the NetBeans IDE Java EE development features, join the nbj2ee@netbeans.org mailing list.