

Web Services (JAX-WS) in Java EE 5

Contributed and maintained by Geertjan Wielenga, Manisha Umbarje, Martin Grebac, Milan Kuchtiak, and Radko Najman

[Java API for XML Web Services \(JAX-WS\) 2.0, JSR 224](#), is an important part of the Java EE 5 platform. A follow-on 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 takes you through the basics of using the IDE to develop a JAX-WS web service and to consume it in three different clients—either a Java class in a Java SE application, or a servlet or JSP page in a web application. The three clients that you create in this document are separate applications, all consuming the same web service.

Expected duration: 25 minutes

Software Needed for the Tutorial

Before you begin, you need to install the following software on your computer:

- [NetBeans IDE 5.5 \(download\)](#).
- Java Standard Development Kit (JDK) version 5.0 or version 6.0 ([download](#)).
- Sun Java System Application Server 9.0 (if not bundled with your NetBeans IDE installation, [download](#) and install it separately).

Tutorial Exercises

- [Installing and Configuring the Tutorial Environment](#)
- [Creating a Web Service](#)
- [Coding the Web Service](#)
- [Deploying and Testing the Web Service](#)
- Consuming the Web Service in
 - [a Java class in a Java SE Application](#)
 - [a servlet in a web application](#)
 - [a JSP page in a web application](#)

Installing and Configuring the Tutorial Environment

If you have not registered an instance of the Sun Java System Application Server 9.0, you must do so before you can begin developing Java EE 5 applications:

1. Choose Tools > Server Manager from the main window.
2. Click Add Server. Select Sun Java System Application Server and give a name to the instance. Then click Next.
3. Specify the server information, the location of the local instance of the application server, and the domain to which you want to deploy.
4. Click Finish.

Note: If you want to deploy to the Tomcat Web Server, you can do so, except that, since it only has a web container, you should create a web application, not an EJB module, in the next section. JAX-WS web services, unlike JSR-109 web services, can deploy successfully to the Tomcat web container.

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.

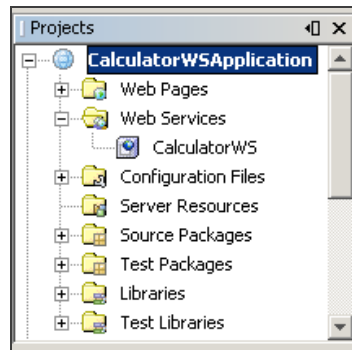
Choosing a Container

You can either deploy your web service in a web container or in an EJB container. This depends on implementation choices. For example, if you plan to deploy to the Tomcat Web Server, which only has a web container, you should choose to create a web application, and not an EJB module.

1. Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Web category or EJB Module from the Enterprise category.
2. Name the project CalculatorWSApplication.
3. Depending on the deployment server that you want to use, do the following:
 - For the Sun Java System Application Server, set the J2EE Version to Java EE 5.
 - For the Tomcat Web Server, unselect the Set Source Level to 1.4 checkbox.
4. Click Finish.

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, type org.me.calculator in Package, and click Finish.
The Projects window displays the new web service. For example, for web applications the Projects window now looks as follows:



The IDE automatically creates the deployment descriptors required for the server, if any. For the Sun Java System Application Server, no deployment descriptor is needed. For web services deployed to the Tomcat Web Server, `sun-jaxws.xml` and a `WSServlet` item in `web.xml` are added.

Summary

In this exercise, you created a NetBeans project and set up the web service.

Coding the Web Service

The goal of this exercise is to do something meaningful with the files and code that the IDE has generated for you. You will add an operation that will add two numbers received from a client.

Adding Business Logic to the Web Service

1. Expand the Web Services node and double-click the `CalculatorWS` node. The web service opens in the Source Editor. Note that an operation exists in the code already. It is commented out. Here, we create a new operation from scratch. Another way of creating the operation would be to remove the lines that comment out the code.
2. Right-click within the body of the class, either above or below the code that is commented out, and choose Web Service > Add Operation.
3. In the upper part of the Add Operation dialog box, type `add` in Name and choose `int` from 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`. Click OK.
5. Click Add again and create a parameter of type `int` called `j`.
6. Click OK at the bottom of the Add Operation dialog box. Notice that a skeleton for the `add` method has been added to the Source Editor:

```
@WebMethod
public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {
    // TODO implement operation
    return 0;
}
```

7. Change the `add` method 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;
}
```

Summary

In this exercise, you added code to the web service.

Deploying and Testing the Web Service

When you deploy a web service to a web container, the IDE lets you test the web service to see if it functions as you expect. The Tester application, provided by the Sun Java System Application Server, is integrated into the IDE for this purpose. For the Tomcat Web Server, there is a similar tool. However, while the Sun Java System Application Server's Tester page lets you enter values and test them, the Tomcat Web Server does not. In the latter case, you can only see that the web service is deployed, you cannot test the values. No facility for testing whether an EJB module is deployed successfully is currently available.

To test successful deployment to a web container:

- Right-click the project node, choose Properties, and click Run. Depending on the deployment server that you want to use, do the following:
 - For the Sun Java System Application Server, type `/CalculatorWSService?Tester` in the Relative URL field.
 - For the Tomcat Web Server, type `/CalculatorWS?Tester` in the Relative URL field.

Note: Since the result of a deployed EJB module is not displayed in a browser, you cannot take the step above if you are working with an EJB module.

- Right-click the project node and choose Run Project. The IDE starts the application server, builds the application, and opens the tester page in your browser, if you deployed a web application to the Sun Java System Application Server. For the Tomcat Web Server and deployment of EJB modules, the situation is different:

- If you deployed to the Tomcat Web Server, you will see the following instead, which indicates that you have successfully deployed your web service:

Web Services		
Port Name	Status	Information
CalculatorWS	ACTIVE	Address: http://localhost:8084/CalculatorWSApplication/CalculatorWS WSDL: http://localhost:8084/CalculatorWSApplication/CalculatorWS?wsdl Port QName: {http://pkg.me.org/} CalculatorWSPort Implementation class: org.me.pkg.CalculatorWS

- If you deployed an EJB module, successful deployment is indicated by the following messages in the Output window:

```
Deployment of application CalculatorWSApplication completed successfully
Enable of CalculatorWSApplication in target server completed successfully
Enable of application in all targets completed successfully
All operations completed successfully
run-deploy:
run:
BUILD SUCCESSFUL
```

- If you deployed to the Sun Java System Application Server, type two numbers in the tester page, as shown below:

CalculatorWS Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

```
public abstract int org.netbeans.CalculatorWSProject.add(int,int)
```

(,)

- The sum of the two numbers is displayed:

add Method invocation

Method parameter(s)

Type	Value
int	2
int	3

Method returned

int : "5"

Summary

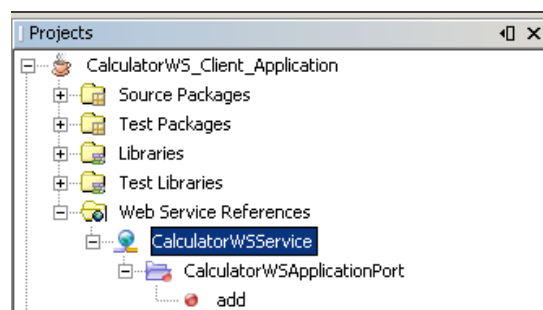
In this exercise, you deployed a web service and tested it.

Consuming the Web Service

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

Client 1: Java Class in Java SE Application

- Choose `File > New Project (Ctrl-Shift-N)`. Select `Java Application` from the `General` category. Name the project `CalculatorWS_Client_Application`.
Note: At the time of writing, issue [Issue 10](#) was unresolved; therefore, you must create your web service client in a project folder that does *not* contain spaces in its path. For example, the path should not be `"C:\Documents and Settings\..."`.
 Click `Finish`.
- Right-click the `CalculatorWS_Client_Application` node and choose `New > Web Service Client`.
- In `Project`, click `Browse`. Browse to the web service that you want to consume. When you have selected the web service, click `OK`.
- Type `org.me.calculator.client` in `Package`, and click `Finish`.
 The `Projects` window displays the new web service client:



- Double-click `Main.java` so that it opens in the `Source Editor`. Delete the `TODO` comment and right-click in that line. Choose `Web Service Client Resources > Call Web Service Operation`.
- Browse to the `Add` operation and click `OK`.
- Change the line that is underlined in red to the following:

```
System.out.println("Sum: " + port.add(3,4));
```

- Right-click the project node and choose `Run Project`.
 The `Output` window should now show the following:

```

compile:
run:
Sum: 7
BUILD SUCCESSFUL (total time: 1 second)

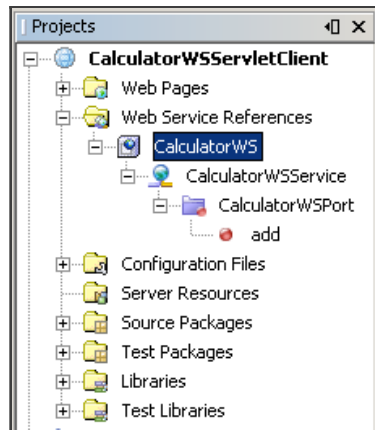
```

Client 2: Servlet in Web Application

1. Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Web category. Name the project `CalculatorWSServletClient`. **Note:** At the time of writing, issue [Issue 10](#) was unresolved; therefore, you must create your web service client in a project folder that does *not* contain spaces in its path. For example, the path should not be "C:\Documents and Settings\...".

Click Finish.

2. Right-click the `CalculatorWSServletClient` node and choose New > Web Service Client.
3. In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK.
4. Type `org.me.calculator.client` in Package, and click Finish. The Projects window displays the new web service client:



5. Right-click the project node and choose New > Servlet. Name the servlet `ClientServlet` and house it in a package called `org.me.calculator.client`. Click Finish. To make the servlet the entry point to your application, right-click the project node, choose Properties, click Run, and type `/ClientServlet` in Relative URL. Click OK.
6. In the Source Editor, remove the line that comments out the body of the `processRequest` method. This is the line that starts the section that comments out the code:

```
/* TODO output your page here
```

Next, delete the line that ends the section of commented out code:

```
*/
```

Add some empty lines after this line:

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

Now, right-click in one of the empty lines that you added. Choose Web Service Client Resources > Call Web Service Operation. The Select Operation to Invoke dialog box appears.

7. Browse to the add operation and click OK. The `processRequest` method now looks as follows (the added code is in bold below):

```

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    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 { // Call Web Service Operation
        org.me.calculator.client.CalculatorWSService service = new org.me.calculator.client.CalculatorWSServi
        org.me.calculator.client.CalculatorWS port = service.getCalculatorWSPort();
        // TODO initialize WS operation arguments here
        int arg0 = 0;
        int arg1 = 0;
        // TODO process result here
        int result = port.add(arg0, arg1);
        System.out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }

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

```

Change the value for `arg0` and `arg1` to other numbers, such as 3 and 4.

Change the `System.out.println` statement to `out.println`.

Add a line that prints out an exception, if an exception is thrown.

The try/catch block should now look as follows (new and changed lines are highlighted):

```

try { // Call Web Service Operation
    org.me.calculator.client.CalculatorWSService service = new org.me.calculator.client.CalculatorWSService()
    org.me.calculator.client.CalculatorWSApplication port = service.getCalculatorWSApplicationPort();
    // TODO initialize WS operation arguments here
    int arg0 = 3;
    int arg1 = 4;
    // TODO process result here
    int result = port.add(arg0, arg1);
    out.println("<p>Result: " + result);
} catch (Exception ex) {
    out.println("<p>Exception: " + ex);
}

```

8. Right-click the project node and choose Run Project.
The server starts, if it wasn't running already; the application is built and deployed, and the browser opens, displaying the calculation result.

Client 3: JSP Page in Web Application

1. Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Web category. Name the project `CalculatorWSJSPClient`.
Note: At the time of writing, issue [Issue 10](#) was unresolved; therefore, you must create your web service client in a project folder that does *not* contain spaces in its path. For example, the path should not be "C:\Documents and Settings\...".

Click Finish.

2. Right-click the `CalculatorWSJSPClient` node and choose New > Web Service Client.
3. In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK.
4. Type `org.me.calculator.client` in Package, and click Finish.
The Projects window displays the new web service client.
5. In the Web Pages folder, double-click `index.jsp` so that it opens in the Source Editor.
6. In the Web Service References node, expand the node that represents the web service. The add operation, which you want to 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.
Change the value for `arg0` and `arg1` to other numbers, such as 3 and 4.

8. Right-click the project node and choose Run Project.
The server starts, if it wasn't running already; the application is built and deployed, and the browser opens, displaying the calculation result:



[Send Us Your Feedback](#)

Next Steps

For more information about using NetBeans IDE 5.5 to develop Java EE applications, see the following resources:

- [Introduction to Java EE 5 Technology](#)
- [Java Persistence in the Java EE 5 Platform](#)
- [Comparing Java EE 5 Platform and J2EE 1.4 Platform](#)
- [Java EE Applications Learning Trail](#)

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