

Experiment No.: 3

Creating Web service

Learning Objective: Student should be able to understand creating web services using and also different web services components.

Theory: A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

The architecture of web service interacts among three roles: service provider, service requester, and service registry. The interaction involves the three operations: publish, find, and bind. These operations and roles act upon the web services artifacts. The web service artifacts are the web service software module and its description.

Web Service Architecture

There are three roles in web service architecture:

- **Service Provider :** From an architectural perspective, it is the platform that hosts the services.
- **Service Requestor :** Service requestor is the application that is looking for and invoking or initiating an interaction with a service. The browser plays the requester role, driven by a consumer or a program without a user interface.
- **Service Registry :** Service requestors find service and obtain binding information for services during development.

Different process in web service Architecture :

- Publication of service descriptions (**Publish**)
- Finding of services descriptions (**Find**)
- Invoking of service based on service descriptions (**Bind**)

Publish: In the publish operation, a service description must be published so that a service requester can find the service.

Find: In the find operation, the service requestor retrieves the service description directly. It can be involved in two different lifecycle phases for the service requestor:

- At design, time to retrieve the service's interface description for program development.
- And, at the runtime to retrieve the service's binding and location description for invocation.

Bind: In the bind operation, the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service.

Artifacts of the web service

There are two artifacts of web services:

- Service
- Service Registry

Service: A service is an **interface** described by a service description. The service description is the implementation of the service. A service is a software module deployed on network-accessible platforms provided by the service provider. It interacts with a service requestor. Sometimes it also functions as a requestor, using other Web Services in its implementation.

Service Description: The service description comprises the details of the interface and implementation of the service. It includes its data types, operations, binding information, and network location. It can also categorize other metadata to enable discovery and utilize by service requestors. It can be published to a service requestor or a service registry.

Web Service Implementation :

Requirements Phase : The objective of the requirements phase is to understand the business requirement and translate them into the web services requirement. The requirement analyst should do requirement elicitation (it is the practice of researching and discovering the requirements of the system from the user, customer, and other stakeholders). The analyst should interpret, consolidate, and communicate these requirements to the development team. The requirements should be grouped in a centralized repository where they can be viewed, prioritized, and mined for interactive features.

Analysis Phase : The purpose of the analysis phase is to refine and translate the web service into conceptual models by which the technical development team can understand. It also defines the high-level structure and identifies the web service interface contracts.

Design Phase : In this phase, the detailed design of web services is done. The designers define web service interface contract that has been identified in the analysis phase. The defined web service interface contract identifies the elements and the corresponding data types as well as mode of interaction between web services and client.

Coding Phase : Coding and debugging phase is quite similar to other software component-based coding and debugging phase. The main difference lies in the creation of additional web service interface wrappers, generation of WSDL, and client stubs.

Test Phase : In this phase, the tester performs interoperability testing between the platform and the client's program. Testing to be conducted is to ensure that web services can bear the maximum load and stress. Other tasks like profiling of the web service application and inspection of the SOAP message should also perform in the test phase.

Deployment Phase : The purpose of the deployment phase is to ensure that the web service is properly deployed in the distributed system. It executes after the testing phase. The primary task of deployer is to ensure that the web service has been properly configured and managed. Other optional tasks like specifying and registering the web service with a UDDI registry also done in this phase.

Web Service Protocol Stack :

A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers.

Service Transport

This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).

XML Messaging

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

Service Description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

Service Discovery

This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

- **Service transport** is responsible for actually transporting XML messages between two computers.

Hyper Text Transfer Protocol (HTTP)

Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XMLRPC or SOAP messages to masquerade as HTTP messages. This is good if you want to integrate remote applications, but it does raise a number of security concerns.

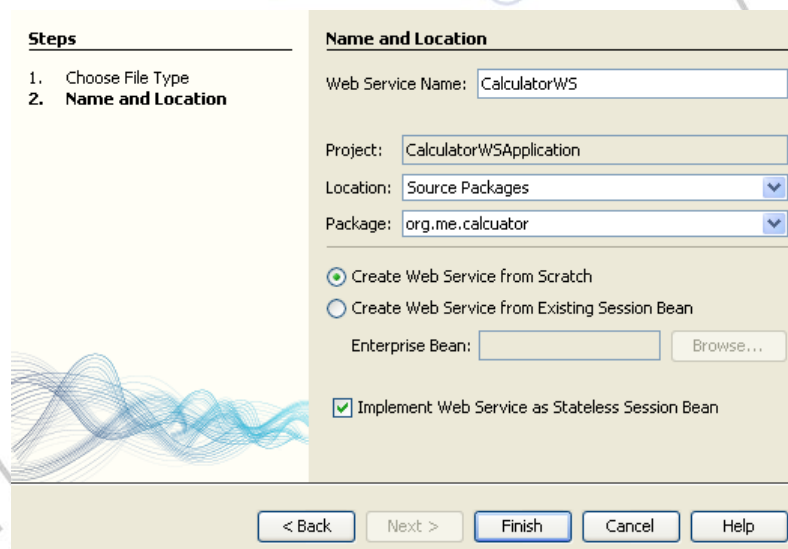
Blocks Extensible Exchange Protocol (BEEP)

This is a promising alternative to HTTP. BEEP is a new Internet Engineering Task Force (IETF) framework for building new protocols. BEEP is layered directly on TCP and includes a number of built-in features, including an initial handshake protocol, authentication, security, and error handling. Using BEEP, one can create new protocols for a variety of applications, including instant messaging, file transfer, content syndication, and network management.

SOAP is not tied to any specific transport protocol. In fact, you can use SOAP via HTTP, SMTP, or FTP. One promising idea is therefore to use SOAP over BEEP.

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 project on GlassFish or WebLogic, select Implement Web Service as a Stateless Session Bean.



- 4.
5. 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:

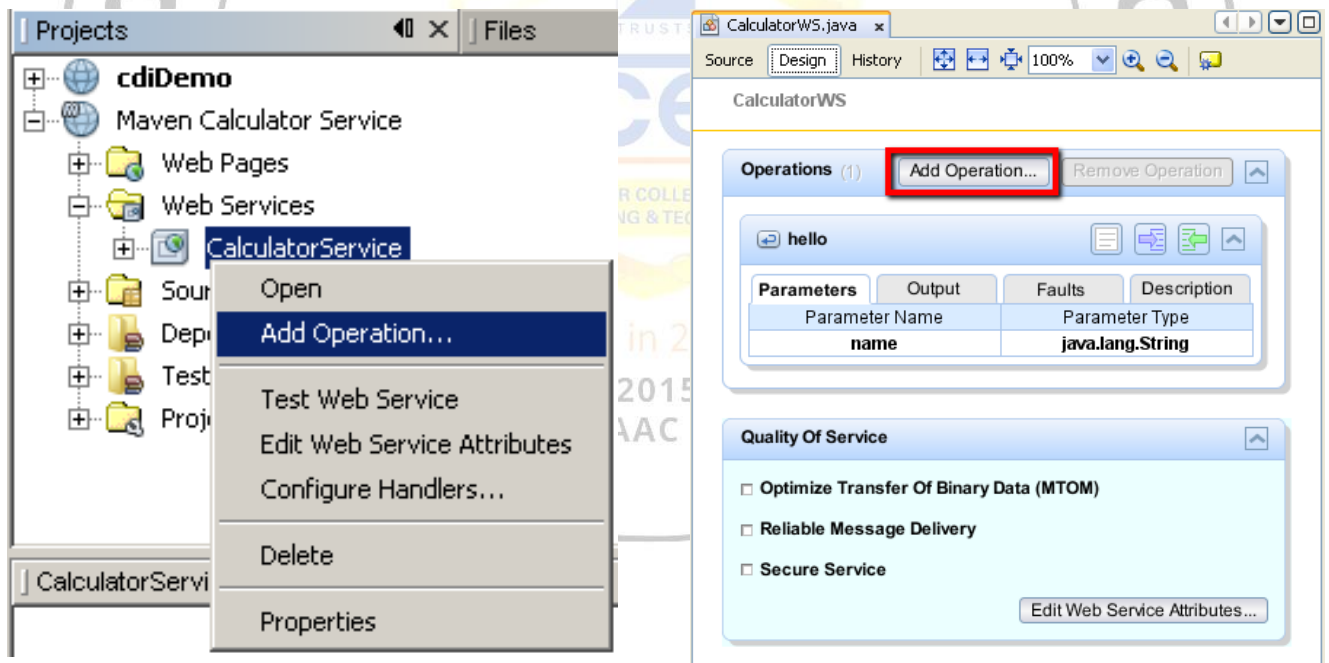
1. Either:

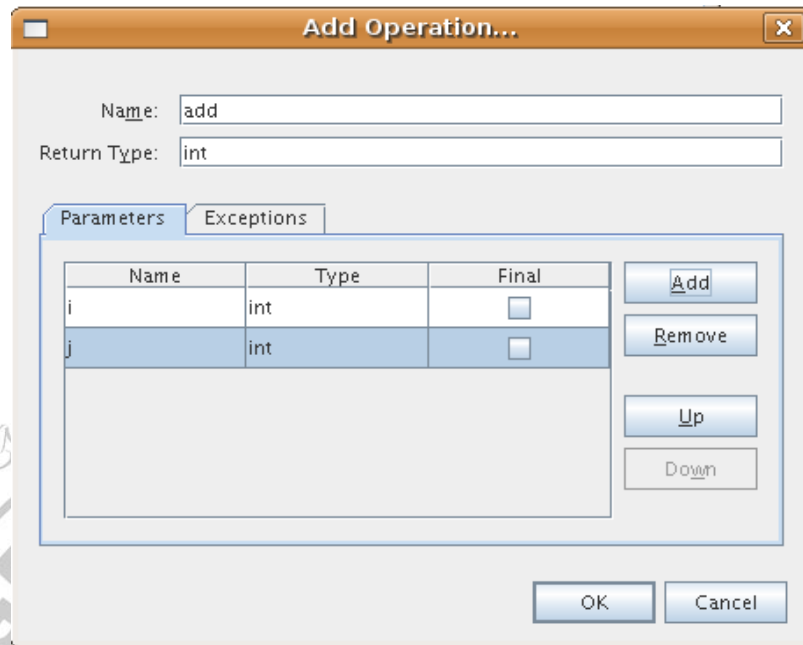
- Change to the Design view in the editor.

Or:

- Find the web service's node in the Projects window. Right-click that node. A context menu opens.
1. Click Add Operation in either the visual designer or the context menu. The Add Operation dialog opens.
 2. In the upper part of the Add Operation dialog box, type add in Name and type int in the Return Type drop-down list.
 3. In the lower part of the Add Operation dialog box, click Add and create a parameter of type int named i .
 4. Click Add again and create a parameter of type int called j .

You now see the following:





Click OK at the bottom of the Add Operation dialog box. You return to the editor.

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:

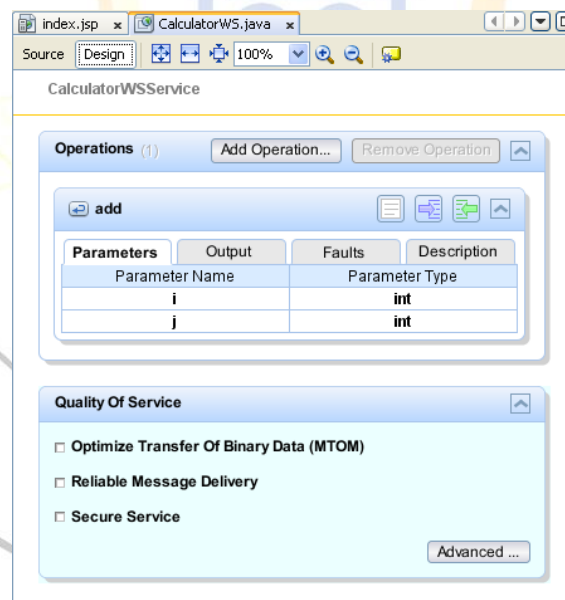


Figure 2. Web service visual designer showing added operation

- Click Source and view the code that you generated in the previous steps. It differs whether you created the service as an Java EE stateless bean or not. Can you see the difference in the screenshots below? (A Java EE 6 or Java EE 7 service that is not implemented as a stateless bean resembles a Java EE 5 service.)



```

package org.me.calculator;

import javax.ws.WebMethod;
import javax.ws.WebParam;
import javax.ws.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.calculator;

import javax.ws.WebMethod;
import javax.ws.WebParam;
import javax.ws.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;
    }

}
  
```

Note. In NetBeans IDE 7.3 and 7.4 you will notice that in the generated @WebService annotation the service name is specified explicitly: @WebService(serviceName = "CalculatorWS").

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

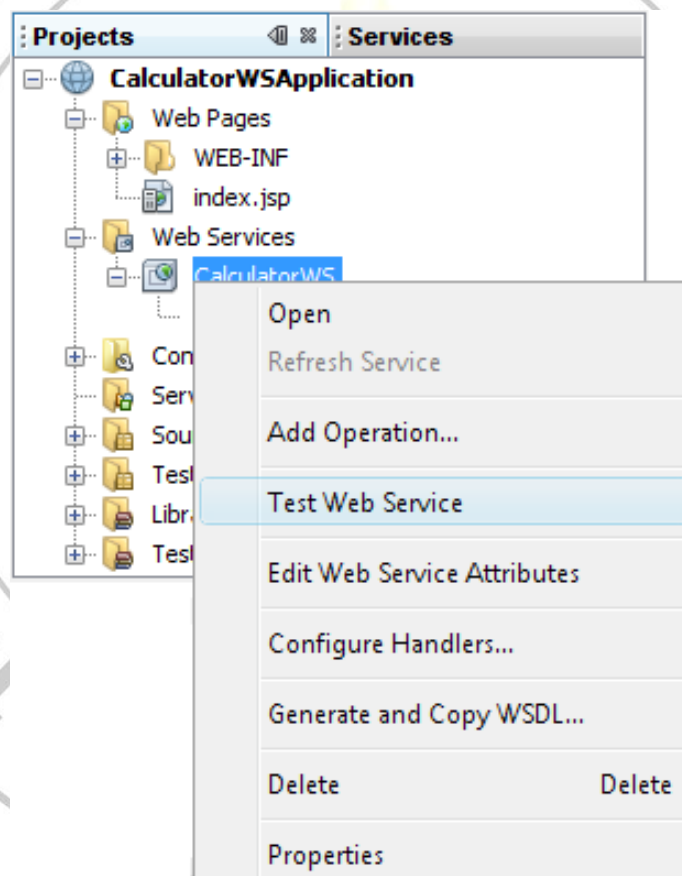
After you deploy a web service to a server, you can use the IDE to open the server's test client, if the server has a test client. The GlassFish and WebLogic servers provide test clients.

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:

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.

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:

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)
```

(2 , 3)

add Method invocation

Method parameter(s)

Type	Value
int	2
int	3

Method returned

int : "5"

You can open a complete Java EE 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 > Web Services > Calculator (EE6).

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

Result and Discussion:

The module on creating web services equipped us with a solid foundation in understanding the core concepts, components, and technologies behind web services. This knowledge prepares us to design, develop, and integrate web services effectively in various software projects, contributing to the modern interconnected digital landscape.

Learning Outcomes: Students should have the ability to

LO1: Define Web services.

LO2: Identify different phases of web services.

LO3: Explain web service protocol.

Course Outcomes: Upon completion of the course students will be able to know about web services and its implementation.

Conclusion:

In the realm of modern software development, web services stand as the cornerstone of seamless communication and integration between disparate applications. Through this learning module, we gained a comprehensive understanding of web services, their essential components, and the diverse technologies that underpin their functionality.

Viva Questions:

1. Define web services.
2. Explain artifacts of web services.
3. Explain different phases in web service Implementation.
4. Explain Service Transport in web services.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	