**SOAP-Based Web Services**

**S**imple **O**bject **A**ccess **P**rotocol - SOAP

The most widely adopted technologies when building web services are SOAP and REST.

These allow you to communicate with external applications in a standardized and scalable manner.

To learn:

* Setup and use **Apache Maven** to create a web service project
* Use the **JAX-WS** library
* Build a web service class and a method which can be invoked by a remote client application
* Deploy the service to a URL endpoint where clients can access it
* Glimpse into a number of artifacts which JAX-WS automatically generates, including the **WSDL** file
* User the WSDL file to build a client app, which you will then use to interact with the server-side web service

**Setting up Apache Maven and Configuring a Maven Webservice Project**

Apache Maven is a build automation and independence management tool.

It’s primarily used with Java project and can be used with C# and Scala as well.

1. Download Apache Maven: <https://maven.apache.org/download.cgi>
2. Place the downloaded Maven folder (unzipped) to a location of your preference
3. Set path and environment variable for Maven

Graphical user interface, text, application

Description automatically generated

1. Run $ **mvn –version** in a terminal to check if Maven is working fine  
   Text

   Description automatically generated

|  |  |  |
| --- | --- | --- |
| |  | | --- | | 1. **Create a maven webservice project using command line:**   **$ mvn archetype:generate -D groupId=com.paulchheang -D**  **artifactId=soap-webservice -D archetypeArtifactId=maven-archetype-webapp -D archetypeVersion=1.4 -D**  **interactiveMode=false**   1. **Open Webapp project in IDE (Intellij IDEA)**   Graphical user interface, application  Description automatically generated | |  | |

1. Add **jaxws-rt** as dependency in the ‘pom.xml’ file:

Text

Description automatically generated

1. Create a new folder named “java” in src\main\
2. Create a new package in the “java” folder, example name: com.jaxwsservice
3. Create a java class in the newly created package

A screenshot of a computer

Description automatically generated with medium confidence

**Defining Web Service Server App**

The server app will receive requests from clients and send back the responses in the form of SOAP message.

To create a SOAP web service:

1. Define a web service app
2. Publish the web service app to an end point (URL)
3. Add Jakarta Activation dependency in pom.xml file: <https://mvnrepository.com/artifact/jakarta.activation/jakarta.activation-api/>

* **Define Web service:**

*MyWebServiceImplementation.java*

|  |
| --- |
| package com.jaxwsservice;    import jakarta.jws.WebMethod;  import jakarta.jws.WebService; //For older version of Apache Tomcat, it uses javax.jws.WebService    //Add annotation to point to the interface "MyWebServiceInterface.java"  @WebService  //Definition of a basic web service that will execute a method invoked by a client  public class MyWebServiceImplementation {  @WebMethod  public String myGreetingMessage(String mystr) {  return "Hello, " + mystr + ". Welcome to the world of Web Services! ";  }  } |

* **Publish Web Service:**

*MyWebServicePublisher.java*

|  |
| --- |
| package com.jaxwsservice;    import jakarta.xml.ws.Endpoint;    public class MyWebServicePublisher {    public static void main(String[] args) {    Endpoint.*publish*(<http://localhost:8080/webservice/greeting>,  new MyWebServiceImplementation());  }  } |

* **Add Jakarta Activation Dependency:**

|  |
| --- |
| <!-- <https://mvnrepository.com/artifact/jakarta.activation/jakarta.activation-api> -->  <dependency>  <groupId>jakarta.activation</groupId>  <artifactId>jakarta.activation-api</artifactId>  <version>2.1.0</version>  </dependency> |

**Notes:**

* When deploying the Web Service Server App, it will create a **WSDL** file for client app to pull in order to make request to server app.
* **WSDL** stands for Web Service Description Language

Graphical user interface, text, application

Description automatically generated

**Creating Web Service Client**

1. Pull WSDL file from Web Service Server using WS Import tool
2. Add 2 dependencies in pom.xml: jaxws-rt and jakarka.activation
3. Write code for client app

* **Pull WSDL file using WS Import – tool can be executed from the command line**

1. Open a terminal
2. Locate to the path where the SOA client project file is located
3. Run: cd src/main/java
4. Run: which wsimport

This should show /usr/bin/wsimport

1. Run this command to pull WSDL file: wsimport -keep -p client <http://locahost:8080/webservice/greeting?wsdl>

* **Add dependencies**

|  |
| --- |
| <dependency>  <groupId>com.sun.xml.ws</groupId>  <artifactId>jaxws-rt</artifactId>  <version>3.0.0</version>  </dependency>  <!-- <https://mvnrepository.com/artifact/jakarta.activation/jakarta.activation-api> -->  <dependency>  <groupId>jakarta.activation</groupId>  <artifactId>jakarta.activation-api</artifactId>  <version>2.1.0</version>  </dependency> |

* **Code for client app:**

*GreetingClient.java*

|  |
| --- |
| package client;    import java.net.URL;    import javax.xml.namespace.QName;  import jakarta.xml.ws.Service;    public class GreetingClient {    public static void main(String[] args) throws Exception {    URL wsdlUrl = new URL(<http://localhost:8080/webservice/greeting?wsdl>);    QName qname = new QName(<http://jaxwsservice.com/>,  "MyWebServiceImplementationService");    Service service = Service.*create*(wsdlUrl, qname);    MyWebServiceInterface myService = service.getPort(MyWebServiceInterface.class);    System.*out*.println(myService.myGreetingMessage("Alice"));    }  } |