# **SOAP Tutorial**

A simple tutorial to setup and get started using SOAP via JAX-WS

## Definition of Terms

|  |  |
| --- | --- |
| Web service | Service available with network / web.  Difference from website, the website is for human consumption / interaction, while the web service is code consumption. Two applications (using different programming language) communicating with each other |
| WSDL | Web Service Definition Language  Describe the service in XML format  Contract for web service and technology independent |
| UDDI | Universal Description and Discovery Integration  Serves as yellow pages for web service  Directory where any publisher can publish the web service  Consumer can query the directory and get access to all different web service |
| SOAP | A messaging protocol  How to send the XML data  XML based messaging protocol that defines a set of rules for performing request-response dialogues  Format to encode and decode messages  Not tied to any particular transport protocol, though HTTP is popular |
| SEI | Service Endpoint Interface  Convert the object to soap message  Translate the whole service call to a soap message  Interface to web service that provides a way for client application to call web service irrespective of technology.  A service that is publish / expose outside |
| Web Service Client | The party who access the published service |

## Tools:

java version "1.7.0\_79"

SoapUI 5.3.0

Eclipse - Version: Neon.1a Release (4.6.1)

Summary

1. Create Web service interface and implementation by annotating the class with @WebService and defining the value for endpointInterface.
2. Create Endpoint Publisher
3. Create WS Client
4. Trace the request and response

## Simple Web Service

Step 1: Create a web service endpoint interface

|  |
| --- |
| package com.lea.soap.tutorial.helloWorld;  import javax.jws.WebMethod;  import javax.jws.WebService;  import javax.jws.soap.SOAPBinding;  /\*\*  \* Default style was Document, so we need to explicitly assigned RPC  \* @author Lea \*  \*/  @WebService  @SOAPBinding(style = SOAPBinding.Style.RPC)  public interface HelloWorld {    @WebMethod  public String greetings();    } |

Step 2: Create WS Endpoint Implementation

|  |
| --- |
| package com.lea.soap.tutorial.helloWorld;  import javax.jws.WebMethod;  import javax.jws.WebService;  /\*\*  \* Mandatory: endpointInterface  \* @WebService is the annotation to define that the class is a webservice  \* @author Lea  \*  \*/  @WebService(endpointInterface = "com.lea.soap.tutorial.helloWorld.HelloWorld")  public class HelloWorldImpl implements HelloWorld {  private HelloWorldService helloWorldService;    HelloWorldImpl (){  helloWorldService = new HelloWorldService();  }    @Override  @WebMethod  public String greetings() {  return helloWorldService.greetings();  }  } |

\*\* if endpointInterface was not define, when we run the client, it will display:

Exception in thread "main" javax.xml.ws.WebServiceException: Undefined port type: {http://helloWorld.tutorial.soap.lea.com/}HelloWorld

at com.sun.xml.ws.client.WSServiceDelegate.getPort(WSServiceDelegate.java:472)

at com.sun.xml.ws.client.WSServiceDelegate.getPort(WSServiceDelegate.java:479)

at javax.xml.ws.Service.getPort(Service.java:188)

at com.lea.soap.tutorial.helloWorld.HelloWorldClient.main(HelloWorldClient.java:21)

Also create the service for separation of implementation and web service

|  |
| --- |
| **package** com.lea.soap.tutorial.helloWorld;  **public** **class** HelloWorldService {  **public** String greetings(){  **return** "Hello World";  }  } |

Step 3: Create WS publisher (this is another option, it is either deploy on server or define a simple publisher using Java)

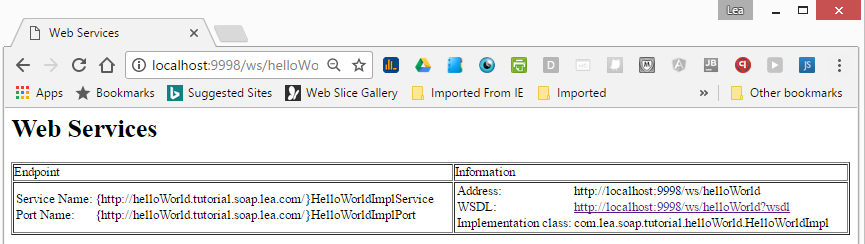
|  |
| --- |
| **package** com.lea.soap.tutorial.helloWorld;  **import** javax.xml.ws.Endpoint;  /\*\*  \* Endpoint publisher  \* No need for a server  \* **@author** Lea  \*  \*/  **public** **class** HelloWorldPublisher {  /\*\*  \* http://localhost:9998/ws/helloWorld - a user defined value  \* **@param** args  \*/  **public** **static** **void** main(String[] args) {  Endpoint.*publish*("http://localhost:9998/ws/helloWorld", **new** HelloWorldImpl());  }  } |

Run the publisher class.

This will be displayed on the console:

|  |
| --- |
| Jun 10, 2017 5:33:42 PM com.sun.xml.ws.server.MonitorBase createRoot  INFO: Metro monitoring rootname successfully set to: com.sun.metro:pp=/,type=WSEndpoint,name=HelloWorldImplService-HelloWorldImplPort |

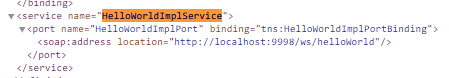
Step 4: Test via URL: <http://localhost:9998/ws/helloWorld>



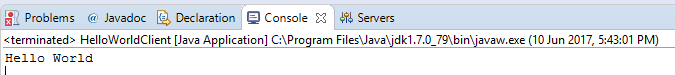
## Web Service Client

Option 1: Create WS Client to invoke the service

|  |
| --- |
| package com.lea.soap.tutorial.helloWorld;  import java.net.MalformedURLException;  import java.net.URL;  import javax.xml.namespace.QName;  import javax.xml.ws.Service;  public class HelloWorldClient {  public static void main(String[] args) throws MalformedURLException {  //The Parameter is the same value defined on HelloWorldPublisher  URL url = new URL("http://localhost:9998/ws/helloWorld?wsdl");    /\*\*  \* First argument is the backward naming convention of the package where we define the webservice  \* second argument - the name specified when we run the publisher - also displayed on WSDL  \* Example, the HelloWorldImpl is in package 'com.lea.soap.tutorial.helloWorld', this means  \*  \*/  QName qname = new QName("http://helloWorld.tutorial.soap.lea.com/", "HelloWorldImplService");  Service service = Service.create(url, qname);  HelloWorld hello = service.getPort(HelloWorld.class);  System.out.println(hello.greetings());  }  } |



Once run, the ‘Hello World’ will be displayed on the console.



The service name is the second parameter value when we instantiate QName.

Option 2: Another way to test was using SOAPUI (https://www.soapui.org/downloads/soapui.html)

|  |  |
| --- | --- |
| **Request** | **Response** |
| <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:hel="http://helloWorld.tutorial.soap.lea.com/">  <soapenv:Header/>  <soapenv:Body>  <hel:greetings/>  </soapenv:Body>  </soapenv:Envelope> | <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  <S:Body>  <ns2:greetingsResponse xmlns:ns2="http://helloWorld.tutorial.soap.lea.com/">  <return>Hello World</return>  </ns2:greetingsResponse>  </S:Body>  </S:Envelope> |

Option 3: Using wsimport command – to generate java class based on WSDL

Step 1: Open command prompt and type:

wsimport -keep <http://localhost:9998/ws/helloWorld?wsdl>

or if you download the WSDL file:

wsimport -keep is src <wsdl directory>

Step 2: Copy the generated class on separate folder. I created wsdl directory and placed the 2 generated java class on that directory.

I only copied the HelloWorldImplService.java and changed the package of HelloWorld to com.lea.soap.tutorial.helloWorld.

Step 3: Create WS Client based on generated Java Class

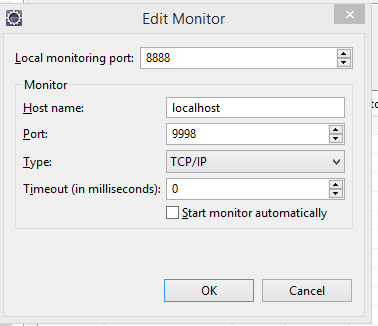
|  |
| --- |
| **package** com.lea.soap.tutorial.helloWorld;  **import** com.lea.soap.tutorial.helloWorld.wsdl.HelloWorldImplService;  **public** **class** HelloWorldClientWSDL {  **public** **static** **void** main(String[] args) {  HelloWorldImplService helloService = **new** HelloWorldImplService();  HelloWorld hello = helloService.getHelloWorldImplPort();  System.***out***.println(hello.greetings());  }  } |

Step 4: Run and verify that ‘Hello World’ is displayed on console

## Intercept web service

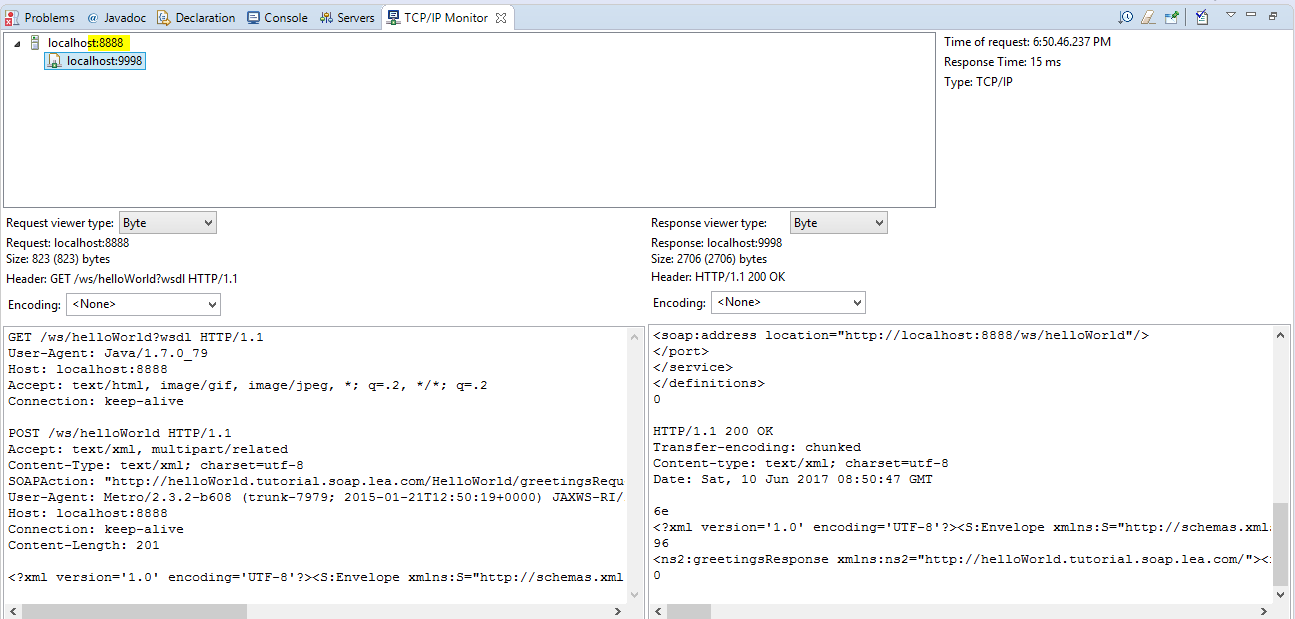
To display what is happening at the background, or the request and response when we run our client, we can intercept it by enabling it from the eclipse:

Windows Menu > Preferences > Run/Debug > TCP / IP Monitor



For our WS Client, we need to changed the port from 9998 to 8888.

URL url = new URL("http://localhost:8888/ws/helloWorld?wsdl");



|  |  |
| --- | --- |
| Request | Response |
| GET /ws/helloWorld?wsdl HTTP/1.1  User-Agent: Java/1.7.0\_79  Host: localhost:8888  Accept: text/html, image/gif, image/jpeg, \*; q=.2, \*/\*; q=.2  Connection: keep-alive  POST /ws/helloWorld HTTP/1.1  Accept: text/xml, multipart/related  Content-Type: text/xml; charset=utf-8  SOAPAction: "http://helloWorld.tutorial.soap.lea.com/HelloWorld/greetingsRequest"  User-Agent: Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revision#unknown  Host: localhost:8888  Connection: keep-alive  Content-Length: 201  <?xml version='1.0' encoding='UTF-8'?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:greetings xmlns:ns2="http://helloWorld.tutorial.soap.lea.com/"/></S:Body></S:Envelope> | HTTP/1.1 200 OK  Transfer-encoding: chunked  Content-type: text/xml;charset=utf-8  Date: Sat, 10 Jun 2017 08:50:46 GMT  87c  <?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revision#unknown. --><!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revision#unknown. --><definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1\_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://helloWorld.tutorial.soap.lea.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://helloWorld.tutorial.soap.lea.com/" name="HelloWorldImplService">  <types/>  <message name="greetings"/>  <message name="greetingsResponse">  <part name="return" type="xsd:string"/>  </message>  <portType name="HelloWorld">  <operation name="greetings">  <input wsam:Action="http://helloWorld.tutorial.soap.lea.com/HelloWorld/greetingsRequest" message="tns:greetings"/>  <output wsam:Action="http://helloWorld.tutorial.soap.lea.com/HelloWorld/greetingsResponse" message="tns:greetingsResponse"/>  </operation>  </portType>  <binding name="HelloWorldImplPortBinding" type="tns:HelloWorld">  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>  <operation name="greetings">  <soap:operation soapAction=""/>  <input>  <soap:body use="literal" namespace="http://helloWorld.tutorial.soap.lea.com/"/>  </input>  <output>  <soap:body use="literal" namespace="http://helloWorld.tutorial.soap.lea.com/"/>  </output>  </operation>  </binding>  <service name="HelloWorldImplService">  <port name="HelloWorldImplPort" binding="tns:HelloWorldImplPortBinding">  <soap:address location="http://localhost:8888/ws/helloWorld"/>  </port>  </service>  </definitions>  0  HTTP/1.1 200 OK  Transfer-encoding: chunked  Content-type: text/xml; charset=utf-8  Date: Sat, 10 Jun 2017 08:50:47 GMT  6e  <?xml version='1.0' encoding='UTF-8'?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body>  96  <ns2:greetingsResponse xmlns:ns2="http://helloWorld.tutorial.soap.lea.com/"><return>Hello World</return></ns2:greetingsResponse></S:Body></S:Envelope>  0 |

**Source Code**

Part 1 – hello World



**Reference:**

Mkyong.com

JavaBrains

<http://www.soapuser.com/>

stackoverflow