1. [WSDL elements](#WSDL)
2. Interview Questions

<http://www.javaexperience.com/jax-ws-webservice-interview-questions/>

1. [What is @webparam in SOAP](#webparam_annotation)
2. [Return custom object from service](#return_custom_object_soap)
3. [SOPA Advantages and Disadvantages](#advnatages_disadvantages_soap)

## What is WSDL

**The Web Services Description Language or WSDL** for short describes a the Web service interface. It consists of messages that are exchanged between the client and server. The messages are described abstractly and then bound to a concrete network protocol and message format. Web service definitions can be mapped to any implementation language, platform, object model, or messaging system.

## [WSDL Example Tutorial Test Sample SOAP Web Service WSDL](http://www.teqlog.com/wp-content/uploads/2011/07/wsdl_structure.gif)WSDL Elements

### ***Definition***

This contains the attribute name, which in turn has the web service name.

### ***Types***

This element uses the XML schema language to declare complex data types and elements that are used elsewhere in the WSDL document.

### ***Message***

This element describes the message’s payload using XML schema built-in types, complex types.

### ***PortType/Interface and Operation***

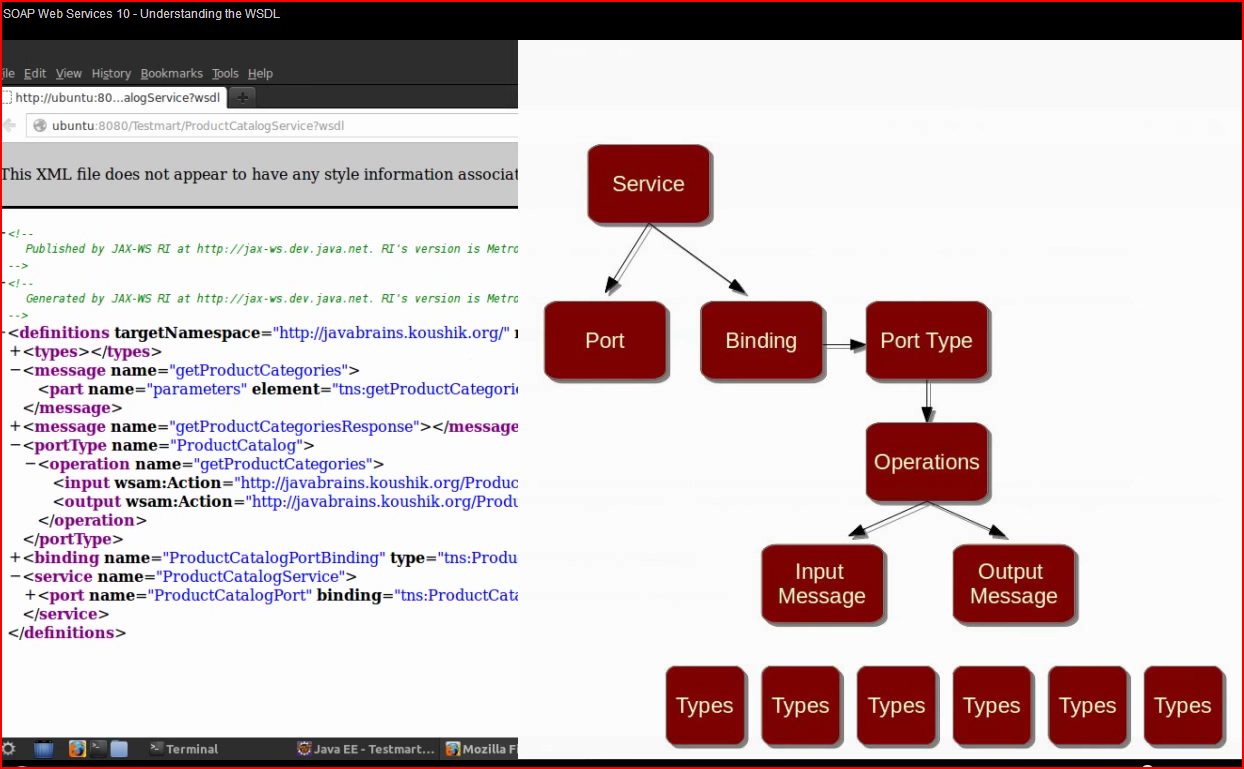
These elements describe a Web service’s interface and define its methods.

### ***Binding***

This element assigns a portType and its operation elements to a particular protocol (for instance, SOAP 1.1) and encoding style.

### ***Service***

This element is responsible for assigning an Internet address to a specific binding.



<http://www.javaexperience.com/understanding-wsdl-structure-and-elements/>

A WSDL is used to provide an abstract view of the [web service](http://www.javaexperience.com/create-and-publish-soap-web-service-on-servlet-container/) to the consumer. There are various parts of web service which includes URL to consume service, messages to be exchanged and encoding for messages. We shall see all the components of WSDL in detail below.

Let us start with a sample WSDL and then pickup each component one by one:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" standalone="yes"?>  <!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.6 in JDK 6. -->  <definitions targetNamespace="http://sample.com/" name="StockPriceImplService" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://sample.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">    <types>      <xsd:schema>        <xsd:import namespace="http://sample.com/" schemaLocation="StockPriceImplService\_schema1.xsd"/>      </xsd:schema>    </types>    <message name="getStockP">      <part name="parameters" element="tns:getStockP"/>    </message>    <message name="getStockPResponse">      <part name="parameters" element="tns:getStockPResponse"/>    </message>    <message name="NumberFormatException">      <part name="fault" element="tns:NumberFormatException"/>    </message>    <portType name="StockPrice">      <operation name="getStockP">        <input message="tns:getStockP"/>        <output message="tns:getStockPResponse"/>        <fault message="tns:NumberFormatException" name="NumberFormatException"/>      </operation>    </portType>    <binding name="StockPriceImplPortBinding" type="tns:StockPrice">      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>      <operation name="getStockP">        <soap:operation soapAction=""/>        <input>          <soap:body use="literal"/>        </input>        <output>          <soap:body use="literal"/>        </output>        <fault name="NumberFormatException">          <soap:fault name="NumberFormatException" use="literal"/>        </fault>      </operation>    </binding>    <service name="StockPriceImplService">      <port name="StockPriceImplPort" binding="tns:StockPriceImplPortBinding">        <soap:address location="REPLACE\_WITH\_ACTUAL\_URL"/>      </port>    </service>  </definitions> |

The root element “definitions”, consists of the five elements:

* types
* message
* portType
* binding
* service

Apart from these five, one major element is “operation” which is a sub-element of “portType”. Let us cover each one of them:

**1) types :** It refers to the data type information. When SOAP messages are exchanged, parameters need to be passed as with normal function calls. Each of the parameter has a type information associated with it. For RPC style web services, the type information includes the standard [XML data types](http://www.javaexperience.com/strip-invalid-characters-from-xml/) like String, int, long etc. But for DOCUMENT style web services, the types element refers to an XSD which can be used to describe complex data types by using XSD constructs. You may want to read about [RPC vs DOCUMENT web services](http://www.javaexperience.com/jax-ws-rpc-vs-document-style-web-services/). In our WSDL, StockPriceImplService\_schema1.xsd is the XSD which contains the data type information for each message. The path of XSD could be absolute or relative to the WSDL.

**2) message :** For every invocation of SOAP based web services, there are messages being exchanged between the server and client. These messages include the request and response messages. The “message” elements list down all messages that can be exchanged in the web service including the fault messages (exceptions in Java). The message elements are then further referred to from other elements. The names for message elements in WSDL is used from @WebMethod [annotation](http://www.javaexperience.com/annotations-behind-the-scenes/) if defined in the web service interface.

**3) portType :** This element defines the operations/transactions that can occur in a web service invocation. Each invocation is termed as operation under this WSDL element. Further, each operation consist of multiple messages (input, output or fault). By looking at the messages involved in an operation, one can easily tell the [Message Exchange Pattern (MEP)](http://www.javaexperience.com/soap-message-exchange-patterns-mep/) being followed by the operation. For example: If there is only an input message in an operation then it is a “one way” operation. In our WSDL, getStockP is the portType and getStockP, getStockPResponse and NumberFormatException are the messages.

**4) binding :** This element defines the transport protocol to be used and the encoding style. For transport, mostly it is “HTTP” and style is “literal”. The binding element has structure which is similar to “portType” element with style attributes added to the messages.

**5) service :** This element is used by the web service consumers to locate the web service. It consists of the endpoint URL and the binding name. The above WSDL is created using wsgen tool and hence the URL is left to us to define. The port sub-element of this element can be compared to a logical port in socket programming. A port is the point which uniquely identifies a web service. After identifying the web service, one could invoke multiple operations.

A good [understanding of the WSDL elements](http://www.javaexperience.com/understanding-wsdl-structure-and-elements/) is very important for exposing and consuming a SOAP based web service.

Read more: <http://www.javaexperience.com/understanding-wsdl-structure-and-elements/#ixzz3Ps0DIThF>

<http://www.coderanch.com/t/625824/Web-Services/java/webparam>

USER OF @webparam

Hi   
I have one requirement, in my SOAP i have to use fields which has - (hiphens) , which cannot be used in Java objects as field names .   
For example <ValidatePinRequest>   
<user-pin></user-pin>   
<user-id></user-id>   
</ValidatePinRequest>   
  
And in my Java Object i have a class ValidatePinRequest ,and it has fields userPin, userId ( i cannot name it user-pin,user-id).   
Is there anyway we can use webparam annotation to map it ?   
  
thanks in advance.

@WebService

public interface ValidatePinWS {

    @WebMethod

    public boolean ValidatePinRequest(

            @WebParam(name = "user-id") String userId,

            @WebParam(name = "user-pin") String userPin) ;

}

@WebService (endpointInterface = "com.validation.pin.ValidatePinWS")

public class ValidatePinImpl implements ValidatePinWS {

    @Override

    public boolean ValidatePinRequest(String userId, String userPin) {

        if(userId.equals("abhay") && userPin.equals("pin001")) {

            return true;

        }

        return false;

    }

}

public class ValidatePinPublisher {

    public static final String URI = "<http://localhost:8085/ValidatePin>";

    public static void main(String[] args) {

        //Create instance of service implementation

        ValidatePinImpl impl = new ValidatePinImpl();

        //Make available

        Endpoint endpoint = Endpoint.publish(URI, impl);

        //Test that it is available

        boolean status = endpoint.isPublished();

        System.out.println("Web service status = " + status);

    }

}

How to return custom object from webservice

User JAXB for the object and complextype

Hello   
  
Finally found simple solution to your problem. It just striked me few minutes back. Just remember that XML generation is backed by JAXB.   
I have used JAXB annotations to solve your problem http://cache-www.coderanch.com/images/smilies/3b63d1616c5dfcf29f8a7a031aaa7cad.gif   
  
Below mentioned is the code. 

[?](http://www.coderanch.com/t/625824/Web-Services/java/webparam)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | package com.validation.pin;    import javax.xml.bind.annotation.XmlAccessType;  import javax.xml.bind.annotation.XmlAccessorType;  import javax.xml.bind.annotation.XmlElement;  import javax.xml.bind.annotation.XmlRootElement;    @XmlRootElement(name="UserDetails")  @XmlAccessorType(XmlAccessType.FIELD)  public class UserDetails {      @XmlElement(name="user-id")      String userId;        @XmlElement(name="user-pin")      String userPin;        public String getUserId() {          return userId;      }      public void setUserId(String userId) {          this.userId = userId;      }      public String getUserPin() {          return userPin;      }      public void setUserPin(String userPin) {          this.userPin = userPin;      }  }      import javax.jws.WebMethod;  import javax.jws.WebService;  import javax.jws.soap.SOAPBinding;    @WebService  @SOAPBinding(parameterStyle =SOAPBinding.ParameterStyle.BARE)  public class NewValidatePinImpl {        @WebMethod      public boolean ValidatePinRequest(UserDetails userDetails) {          if(userDetails.getUserId().equals("abhay") && userDetails.getUserPin().equals("pin001")) {              return true;          }          return false;      }  }      import javax.xml.ws.Endpoint;    public class NewValidatePinPublisher {        public static final String URI = "<http://localhost:8085/ValidateUserDetails>";      public static void main(String[] args) {          //Create instance of service implementation          NewValidatePinImpl impl = new NewValidatePinImpl();          //Make available          Endpoint endpoint = Endpoint.publish(URI, impl);          //Test that it is available          boolean status = endpoint.isPublished();          System.out.println("Web service status = " + status);      }  } |

**SOAP Request** 

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:pin="http://pin.validation.com/">   
<soapenv:Header/>   
<soapenv:Body>   
<pin:ValidatePinRequest>   
<!--Optional:-->   
<user-id>abhay</user-id>   
<!--Optional:-->   
<user-pin>pin001</user-pin>   
</pin:ValidatePinRequest>   
</soapenv:Body>   
</soapenv:Envelope>

**SOAP Response** 

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">   
<S:Body>   
<ns2:ValidatePinRequestResponse xmlns:ns2="http://pin.validation.com/">true</ns2:ValidatePinRequestResponse>   
</S:Body>   
</S:Envelope>

SOAP Advantages and Disadvantages

SOAP advantages:

1. platform and language independent
2. it supports both Http or https and SMTP

*SOAP Disadvantages*

SOAP messages are composed of XML. This adds overhead to the client and server when parsing the message, and can potentially be a slow operation.  When SOAP is relying on HTTP/S as its transfer protocol and is not using **WS-Addressing**, the interfacing parties are fixed.