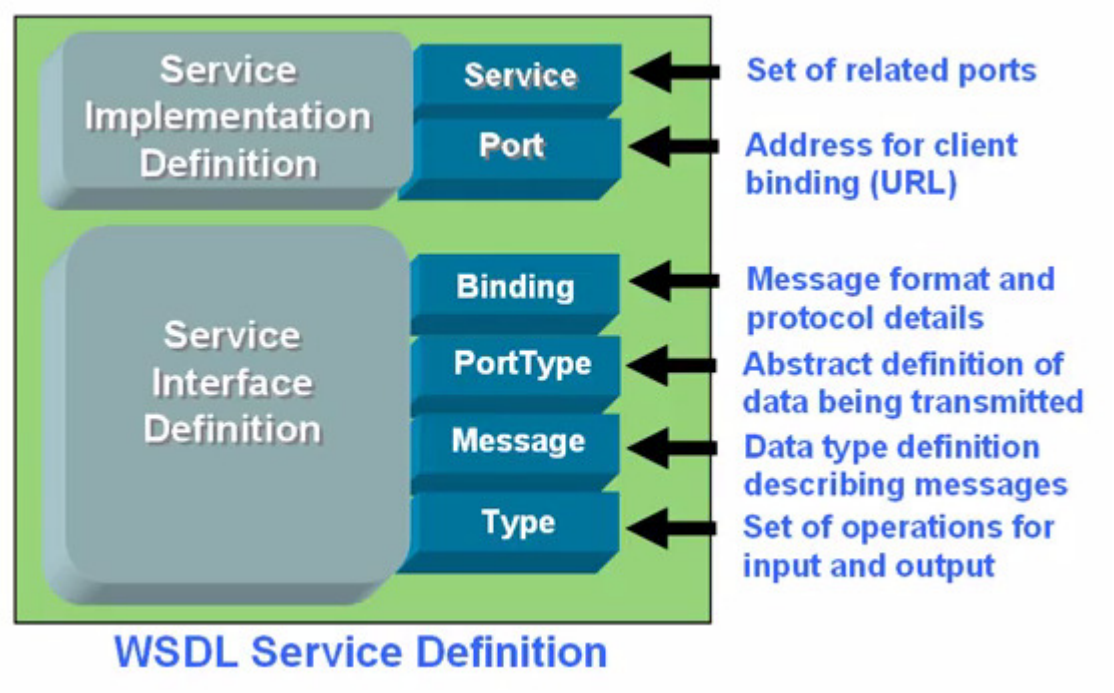


WSDL stands for Web Services Description Language.

WSDL is a language for describing web services and how to access them.

WSDL is written in XML.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- XML
- XML Namespaces
- XML Schema

If you want to study these subjects first, find the tutorials on our Home page.

What is WSDL?

- WSDL stands for Web Services Description Language
- WSDL is written in XML
- WSDL is an XML document
- WSDL is used to describe Web services
- WSDL is also used to locate Web services
- WSDL is a W3C recommendation

WSDL Describes Web Services

WSDL stands for Web Services Description Language.

WSDL is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

WSDL is a W3C Recommendation

WSDL became a W3C Recommendation 26. June 2007.

A WSDL document is just a simple XML document.

It contains set of definitions to describe a web service.

The WSDL Document Structure

A WSDL document describes a web service using these major elements:

Element	Description
<types>	A container for data type definitions used by the web service
<message>	A typed definition of the data being communicated
<portType>	A set of operations supported by one or more endpoints
<binding>	A protocol and data format specification for a particular port type

The main structure of a WSDL document looks like this:

```
<definitions>

<types>
  data type definitions.....
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>
```

```
<binding>
  protocol and data format specification....
</binding>

</definitions>
```

A WSDL document can also contain other elements, like extension elements, and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

WSDL Ports

The **<portType>** element is the most important WSDL element.

It describes a web service, the operations that can be performed, and the messages that are involved.

The <portType> element can be compared to a function library (or a module, or a class) in a traditional programming language.

WSDL Messages

The **<message>** element defines the data elements of an operation.

Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

WSDL Types

The **<types>** element defines the data types that are used by the web service.

For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

WSDL Bindings

The **<binding>** element defines the data format and protocol for each port type.

WSDL Example

This is a simplified fraction of a WSDL document:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>
```

```
<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

In this example the **<portType>** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.

The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".

The **<message>** elements define the **parts** of each message and the associated data types.

Compared to traditional programming, glossaryTerms is a function library, "getTerm" is a function with "getTermRequest" as the input parameter, and getTermResponse as the return parameter.

The **<portType>** element is the most important WSDL element.

WSDL - The <portType> Element

The <portType> element defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.

<portType> defines the connection point to a web service. It can be compared to a function library (or a module, or a class) in a traditional programming language. Each operation can be compared to a function in a traditional programming language.

Operation Types

The request-response type is the most common operation type, but WSDL defines four types:

Type	**Definition**
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response

Notification	The operation can send a message but will not wait for a response

One-Way Operation

A one-way operation example:

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

In the example above, the portType "glossaryTerms" defines a one-way operation called "setTerm".

The "setTerm" operation allows input of new glossary terms messages using a "newTermValues" message with the input parameters "term" and "value". However, no output is defined for the operation.

Request-Response Operation

A request-response operation example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

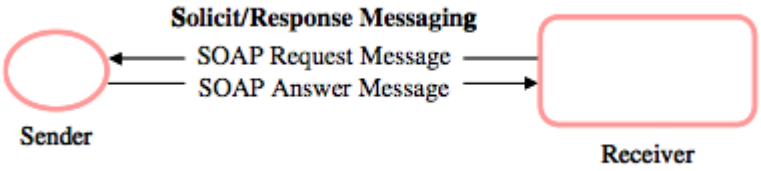
In the example above, the portType "glossaryTerms" defines a request-response operation called "getTerm".

The "getTerm" operation requires an input message called "getTermRequest" with a parameter called "term", and will return an output message called "getTermResponse" with a parameter called "value".

A WSDL binding defines the message format and protocol details for a web service.

Solicit-response and Notification:

A solicit/response operation is an operation in which the service endpoint sends a message and expects to receive an answering message in response.



- This is the opposite of the request/response operation since the service endpoint is initiating the operation (soliciting the client), rather than responding to a request.
- Solicit/response is similar to notification messaging, except that the client is expected to respond to the Web service.
- With this type of messaging the element first declares an tag and then a message definition – exactly the reverse of a request/response operation.

An example of this operation might be a service that sends out order status to a client and receives back a receipt.

1) The client will be available from login to logout.

2) Client will be identified with a unique session-id.

3) Server is always on

Flow is something like

Client 1 ------login()--------------> S

<-----session-id------------ E

R <-- notifications--THIRD PARTY

Client 2------login()--------------> V

R<-----session-id------------ E

4) Here client is a SOAP client and server is notification web-service , which receives notification from external server. This web-services will send a notification to the client as soon it receives any notification from Third party.

Binding to SOAP

A request-response operation example:

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

The **binding** element has two attributes - name and type.

The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "glossaryTerms" port.

The **soap:binding** element has two attributes - style and transport.

The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use. In this case we use HTTP.

The **operation** element defines each operation that the portType exposes.

For each operation the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded. In this case we use "literal".

SOAP stands for Simple Object Access Protocol.

SOAP is a protocol for accessing web services.

SOAP is based on XML.

What You Should Already Know

Before you study SOAP you should have a basic understanding of XML and XML Namespaces.

If you want to study these subjects first, please read our XML Tutorial.

What is SOAP?

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is for communication between applications
- SOAP is a format for sending messages
- SOAP communicates via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C recommendation

Why SOAP?

It is important for application development to allow Internet communication between programs.

Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

SOAP is a W3C Recommendation

SOAP became a W3C Recommendation 24. June 2003.

SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

- An Envelope element that identifies the XML document as a SOAP message
- A Header element that contains header information
- A Body element that contains call and response information
- A Fault element containing errors and status information

All the elements above are declared in the default namespace for the SOAP envelope:

http://www.w3.org/2001/12/soap-envelope

and the default namespace for SOAP encoding and data types is:

http://www.w3.org/2001/12/soap-encoding

Syntax Rules

Here are some important syntax rules:

- A SOAP message MUST be encoded using XML
- A SOAP message MUST use the SOAP Envelope namespace
- A SOAP message MUST use the SOAP Encoding namespace
- A SOAP message must NOT contain a DTD reference
- A SOAP message must NOT contain XML Processing Instructions

Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
```

```
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

A SOAP Example

In the example below, a GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response. The namespace for the function is defined in "http://www.example.org/stock".

A SOAP request:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

The SOAP response:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
```

```
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

WSDL

```
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://cxfjaxws.com/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
        name="helloWorldImplService" targetNamespace="http://cxfjaxws.com/">
        <wsdl:types>
                <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:tns="http://cxfjaxws.com/" elementFormDefault="unqualified"
                        targetNamespace="http://cxfjaxws.com/" version="1.0">

                        <xs:element name="employeeRequest" type="tns:employeeRequest" />

                        <xs:element name="employeeResponse" type="tns:employeeResponse" />

                        <xs:complexType name="employeeRequest">
                                <xs:sequence>
                                        <xs:element minOccurs="1" name="id" type="xs:integer" />
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="employeeResponse">
                                <xs:sequence>
                                        <xs:element minOccurs="1" name="id" type="xs:integer" />
                                        <xs:element minOccurs="0" name="firstname" type="xs:string" />
                                        <xs:element minOccurs="0" name="lastname" type="xs:string" />
                                </xs:sequence>
                        </xs:complexType>

                </xs:schema>
        </wsdl:types>
```

```
31
32          <wsdl:message name="employeeRequestMsg">
33                  <wsdl:part element="tns:employeeRequest" name="parameters" />
34          </wsdl:message>
35          <wsdl:message name="employeeResponseMsg">
36                  <wsdl:part element="tns:employeeResponse" name="parameters" />
37          </wsdl:message>
38
39          <wsdl:portType name="helloWorldPortType">
40                  <wsdl:operation name="getEmployee">
41                          <wsdl:input message="tns:employeeRequestMsg" name="employeeRequestOp" />
42                          <wsdl:output message="tns:employeeResponseMsg" name="employeeResponseOp" />
43                  </wsdl:operation>
44          </wsdl:portType>
45
46          <wsdl:binding name="helloWorldImplServiceSoapBinding" type="tns:helloWorldPortType">
47                  <soap:binding style="document"
48                          transport="http://schemas.xmlsoap.org/soap/http" />
49                  <wsdl:operation name="getEmployee">
50                          <soap:operation soapAction="" style="document" />
51                          <wsdl:input name="employeeRequestOp">
52                                  <soap:body use="literal" />
53                          </wsdl:input>
54                          <wsdl:output name="employeeResponseOp">
55                                  <soap:body use="literal" />
56                          </wsdl:output>
57                  </wsdl:operation>
58          </wsdl:binding>
59          <wsdl:service name="helloWorldImplService">
60                  <wsdl:port binding="tns:helloWorldImplServiceSoapBinding" name="helloWorldImplPort">
61                          <soap:address location="http://localhost:8080/CXFJAXWS-Service-1.0/helloworld" />
62                  </wsdl:port>
63          </wsdl:service>
64  </wsdl:definitions>
```