**WSDL - <definitions>**

The **<definitions>** element must be the root element of all WSDL documents. It defines the name of the web service.

Here is the piece of code from the last chapter that uses the *definitions* element.

<definitions name="HelloService"

targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"

xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

................................................

</definitions>

From the above example, we can conclude that *definitions* −

* is a container of all the other elements.
* specifies that this document is called *HelloService*.
* specifies a *targetNamespace* attribute. The *targetNamespace* is a convention of XML Schema that enables the WSDL document to refer to itself. In this example, we have specified a *targetNamespace* of http://www.examples.com/wsdl/HelloService.wsdl
* specifies a default namespace: xmlns=http://schemas.xmlsoap.org/wsdl/. All elements without a namespace prefix, such as *message* or *portType*, are therefore assumed to be a part of the default WSDL namespace.
* specifies numerous namespaces that are used throughout the remainder of the document.

**NOTE** − The namespace specification does not require the document to be present at the given location. The important point is that you specify a value that is unique, different from all other namespaces that are defined.

**WSDL - <types>**

A web service needs to define its inputs and outputs and how they are mapped into and out of the services. WSDL **<types>** element takes care of defining the data types that are used by the web service. Types are XML documents, or document parts.

* The *types* element describes all the data types used between the client and the server.
* WSDL is not tied exclusively to a specific typing system.
* WSDL uses the W3C XML Schema specification as its default choice to define data types.
* If the service uses only XML Schema built-in simple types, such as strings and integers, then *types* element is not required.
* WSDL allows the types to be defined in separate elements so that the types are reusable with multiple web services.

Here is a piece of code taken from W3C specification. This code depicts how a *types* element can be used within a WSDL.

<types>

<schema targetNamespace = "http://example.com/stockquote.xsd"

xmlns = "http://www.w3.org/2000/10/XMLSchema">

<element name = "TradePriceRequest">

<complexType>

<all>

<element name = "tickerSymbol" type = "string"/>

</all>

</complexType>

</element>

<element name = "TradePrice">

<complexType>

<all>

<element name = "price" type = "float"/>

</all>

</complexType>

</element>

</schema>

</types>

Data types address the problem of identifing the data types and the formats you intend to use with your web services. Type information is shared between the sender and the receiver. The recipients of messages therefore need access to the information you used to encode your data and must understand how to decode the data.

**WSDL - <message>**

The **<message>** element describes the data being exchanged between the web service providers and the consumers.

* Each Web Service has two messages: input and output.
* The input describes the parameters for the web service and the output describes the return data from the web service.
* Each message contains zero or more **<part>** parameters, one for each parameter of the web service function.
* Each **<part>** parameter associates with a concrete type defined in the **<types>** container element.

Let us take a piece of code from the WSDL Example chapter −

<message name = "SayHelloRequest">

<part name = "firstName" type = "xsd:string"/>

</message>

<message name = "SayHelloResponse">

<part name = "greeting" type = "xsd:string"/>

</message>

Here, two message elements are defined. The first represents a request message *SayHelloRequest*, and the second represents a response message *SayHelloResponse*.

Each of these messages contains a single part element. For the request, the part specifies the function parameters; in this case, we specify a single *firstName* parameter. For the response, the part specifies the function return values; in this case, we specify a single greeting return value.

**WSDL – <portType>**

The **<portType>** element combines multiple message elements to form a complete one-way or round-trip operation.

For example, a **<portType>** can combine one request and one response message into a single request/response operation. This is most commonly used in SOAP services. A portType can define multiple operations.

Let us take a piece of code from the WSDL Example chapter −

<portType name = "Hello\_PortType">

<operation name = "sayHello">

<input message = "tns:SayHelloRequest"/>

<output message = "tns:SayHelloResponse"/>

</operation>

</portType>

* The portType element defines a single operation, called *sayHello*.
* The operation consists of a single input message *SayHelloRequest* and an
* output message *SayHelloResponse*.

Patterns of Operation

WSDL supports four basic patterns of operation −

One-way

The service receives a message. The operation therefore has a single *input* element. The grammar for a one-way operation is −

<wsdl:definitions .... >

<wsdl:portType .... > \*

<wsdl:operation name = "nmtoken">

<wsdl:input name = "nmtoken"? message = "qname"/>

</wsdl:operation>

</wsdl:portType >

</wsdl:definitions>

Request-response

The service receives a message and sends a response. The operation therefore has one *input* element, followed by one *output* element. To encapsulate errors, an optional *fault* element can also be specified. The grammar for a request-response operation is −

<wsdl:definitions .... >

<wsdl:portType .... > \*

<wsdl:operation name = "nmtoken" parameterOrder = "nmtokens">

<wsdl:input name = "nmtoken"? message = "qname"/>

<wsdl:output name = "nmtoken"? message = "qname"/>

<wsdl:fault name = "nmtoken" message = "qname"/>\*

</wsdl:operation>

</wsdl:portType>

</wsdl:definitions>

Solicit-response

The service sends a message and receives a response. The operation therefore has one *output* element, followed by one *input* element. To encapsulate errors, an optional *fault* element can also be specified. The grammar for a solicit-response operation is −

<wsdl:definitions .... >

<wsdl:portType .... > \*

<wsdl:operation name = "nmtoken" parameterOrder = "nmtokens">

<wsdl:output name = "nmtoken"? message = "qname"/>

<wsdl:input name = "nmtoken"? message = "qname"/>

<wsdl:fault name = "nmtoken" message = "qname"/>\*

</wsdl:operation>

</wsdl:portType >

</wsdl:definitions>

Notification

The service sends a message. The operation therefore has a single *output* element. Following is the grammar for a notification operation −

<wsdl:definitions .... >

<wsdl:portType .... > \*

<wsdl:operation name = "nmtoken">

<wsdl:output name = "nmtoken"? message = "qname"/>

</wsdl:operation>

</wsdl:portType>

</wsdl:definitions>

**WSDL - <binding>**

The **<binding>** element provides specific details on how a *portType* operation will actually be transmitted over the wire.

* The bindings can be made available via multiple transports including HTTP GET, HTTP POST, or SOAP.
* The bindings provide concrete information on what protocol is being used to transfer *portType* operations.
* The bindings provide information where the service is located.
* For SOAP protocol, the binding is **<soap:binding>**, and the transport is SOAP messages on top of HTTP protocol.
* You can specify multiple bindings for a single *portType*.

The binding element has two attributes : *name* and *type* attribute.

<binding name = "Hello\_Binding" type = "tns:Hello\_PortType">

The *name* attribute defines the name of the binding, and the type attribute points to the port for the binding, in this case the "tns:Hello\_PortType" port.

SOAP Binding

WSDL 1.1 includes built-in extensions for SOAP 1.1. It allows you to specify SOAP specific details including SOAP headers, SOAP encoding styles, and the SOAPAction HTTP header. The SOAP extension elements include the following −

* soap:binding
* soap:operation
* soap:body

soap:binding

This element indicates that the binding will be made available via SOAP. The *style* attribute indicates the overall style of the SOAP message format. A style value of *rpc* specifies an RPC format.

The *transport* attribute indicates the transport of the SOAP messages. The value http://schemas.xmlsoap.org/soap/http indicates the SOAP HTTP transport, whereas http://schemas.xmlsoap.org/soap/smtp indicates the SOAP SMTP transport.

soap:operation

This element indicates the binding of a specific operation to a specific SOAP implementation. The *soapAction* attribute specifies that the SOAPAction HTTP header be used for identifying the service.

soap:body

This element enables you to specify the details of the input and output messages. In the case of HelloWorld, the body element specifies the SOAP encoding style and the namespace URN associated with the specified service.

Here is the piece of code from the Example chapter −

<binding name = "Hello\_Binding" type = "tns:Hello\_PortType">

<soap:binding style = "rpc" transport = "http://schemas.xmlsoap.org/soap/http"/>

<operation name = "sayHello">

<soap:operation soapAction = "sayHello"/>

<input>

<soap:body

encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"

namespace = "urn:examples:helloservice" use = "encoded"/>

</input>

<output>

<soap:body

encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"

namespace = "urn:examples:helloservice" use = "encoded"/>

</output>

</operation>

</binding>

**WSDL - <port>**

A **<port>** element defines an individual endpoint by specifying a single address for a binding.

Here is the grammar to specify a port −

<wsdl:definitions .... >

<wsdl:service .... > \*

<wsdl:port name = "nmtoken" binding = "qname"> \*

<-- extensibility element (1) -->

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

* The port element has two attributes: *name* and *binding* .
* The *name* attribute provides a unique name among all ports defined within the enclosing WSDL document.
* The binding attribute refers to the binding using the linking rules defined by WSDL.
* Binding extensibility elements are used to specify the address information for the port.
* A port MUST NOT specify more than one address.
* A port MUST NOT specify any binding information other than address information.

Here is a piece of code from the Example chapter −

<service name = "Hello\_Service">

<documentation>WSDL File for HelloService</documentation>

<port binding = "tns:Hello\_Binding" name = "Hello\_Port">

<soap:address

location = "http://www.examples.com/SayHello/">

</port>

</service>

**WSDL <service>**

The **<service>** element defines the ports supported by the web service. For each of the supported protocols, there is one port element. The service element is a collection of ports.

* Web service clients can learn the following from the service element −
  + where to access the service,
  + through which port to access the web service, and
  + how the communication messages are defined.
* The service element includes a documentation element to provide human-readable documentation.

Here is a piece of code from the Example chapter −

<service name = "Hello\_Service">

<documentation>WSDL File for HelloService</documentation>

<port binding = "tns:Hello\_Binding" name = "Hello\_Port">

<soap:address

location = "http://www.examples.com/SayHello/">

</port>

</service>

The binding attributes of *port* element associate the address of the service with a binding element defined in the web service. In this example, this is *Hello\_Binding*

<binding name =" Hello\_Binding" type = "tns:Hello\_PortType">

<soap:binding style = "rpc"

transport = "http://schemas.xmlsoap.org/soap/http"/>

<operation name = "sayHello">

<soap:operation soapAction = "sayHello"/>

<input>

<soap:body

encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"

namespace = "urn:examples:helloservice" use = "encoded"/>

</input>

<output>

<soap:body

encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"

namespace = "urn:examples:helloservice" use = "encoded"/>

</output>

</operation>

</binding>