Guide: WSDL

WSDL stands for Web Service Definition Language. It is essentially an abstract interface definition that spells out concrete bindings to on-the-wire formatting of the messages.

Here is an example WSDL file that we will use in this guide:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions
name="TestWSDL"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
 <types>
   <xsd:schema>
     <xsd:element name="User" type="UserRecordType"/>
     <xsd:complexType name="UserRecordType">
       <xsd:sequence>
         <xsd:element name="name" type="xsd:string"/>
         <xsd:element name="id" type="xsd:int"/>
       </xsd:sequence>
     </xsd:complexType>
   </xsd:schema>
 </types>
 <message name="RecordInput">
   <part name="user" element="User"/>
 </message>
 <message name="RecordOperationResult">
   <part name="return" type="xsd:int"/>
 </message>
 <portType name="RecordOperations">
   <operation name="addRecord">
     <input message="RecordInput"/>
     <output message="RecordOperationResult"/>
   </operation>
   <operation name="deleteRecord">
     <input message="RecordInput"/>
     <output message="RecordOperationResult"/>
   </operation>
 </portType>
 <binding name="RecordBindings" type="RecordOperations">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="addRecord">
     <soap:operation style="document" soapAction="addRecord"/>
     <input>
       <soap:body use="literal"/>
     </input>
     <output>
       <soap:body use="literal"/>
     </output>
   </operation>
   <operation name="deleteRecord">
     <soap:operation style="document" soapAction="deleteRecord"/>
     <input>
       <soap:body use="literal"/>
     </input>
     <output>
       <soap:body use="literal"/>
     </output>
   </operation>
 </binding>
 <service name="RecordService">
   <port name="RecordServicePort" binding="RecordBindings">
     <soap:address location="http://localhost:8090/RecordOps"/>
   </port>
 </service>
</definitions>
```

Definitions

The root element (disregard the *<?xml ...?>* prolog) of any WSDL must be the *wsdl:definitions* element:

```
<definitions
   name="TestWSDL"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
```

This declares the named WSDL (in this case *TestWSDL*) and defines any namespaces that will be used in the rest of the document. You will almost always see the following namespace declarations:

http://schemas.xmlsoap.org/wsdl/

This is the WSDL namespace and it is often declared as the default namespace so the WSDL elements don't have to have namespace prefixes.

http://www.w3.org/2001/XMLSchema

This is the XSD (XML Schema Definitions) namespace and it is used to declare schemas and any simple types.

http://schemas.xmlsoap.org/wsdl/soap/

This is the SOAP binding namespace and is used to tie this WSDL to SOAP messages (see below).

Sometimes you will see targetNamespace declarations, which essentially place any defined objects into the specified namespace, so they will have to be accessed through that URI. This is especially useful when you have multiple WSDL files which may define similar operations or types.

Type Declarations

User-defined types may be declared in one of two ways: either in the WSDL itself, or in a separate schema file. To import a schema file, use the *wsdl:import* element:

<import namespace="[URI in which to place the declared types]" location="[file name]"/>

This will pull the given schema into the WSDL and all of the types will be available through the given namespace URI. The other way to declare types is to place the schema in a *wsdl:types* element. The types element contains one child: the xsd:schema element. For more information on schemas and schema structure, please see <link to XSD KB article or other Schema informational source>.

For our purposes in our example, we will deal with a simple user-defined type.

Messages

Messages are the basis for all input/output for a WSDL and its operations. They form the core of all data transfer mechanisms for WSDLs.

```
<message name="RecordInput">
  <part name="user" element="User"/>
</message>
<message name="RecordOperationResult">
  <part name="return" type="xsd:int"/>
</message>
```

All *messages* are essentially maps of named *parts*. In this case, we have two *messages*, each with one *part*. Below, we will show an example of *messages* with more than one *part*.

Parts must have a *name* and either an *element* or type declaration. If the *part* is defined as an *element* via the *element="foo"* directive, then it must directly correspond to a *<xsd:element name="foo" type="..."*> definition in a schema. If the *part* is defined as a *type* via the type=*"fooType"*directive, then it must correspond to either a *<xsd:complexType name="fooType"*> or a *<xsd:simpleType name="fooType"*> in a schema.

The distinction between *type* and *element* here is slight but important. *Parts* that are *elements* will contain the specified element, whereas *parts* that are *typ es* will become that specified type. This will become important when we begin to go over actual instance documents in the SOAP message section below.

Operations

Operations in WSDL are grouped by interface grouping, or port type. These operations are specified in abstract in the WSDL and the concrete implementation essentially implements this abstract interface. The concrete specifications are supplied in the bindings section.

```
<portType name="RecordOperations">
  <operation name="addRecord">
    <input message="RecordInput"/>
    <output message="RecordOperationResult"/>
  </operation>
  <operation name="deleteRecord">
    <input message="RecordInput"/>
    <output message="RecordOperationResult"/>
  </operation>
</portType>
```

This defines a port type grouping called *RecordOperations* with two operations: *addRecord* and *deleteRecord*. Both of these operations take the same input and output: a *RecordInput* message in and a *RecordOperationResult* out.

A port type declaration can have 0 to N operations, although you will typically have at least one operation for each port type declaration.
Each operation can define an input message, an output message, and as many fault messages as necessary. The combination of input and/or output declarations determines the operation types as well as the types of errors the operation can return:

Request-Response

```
<input .../>
<output .../>
<fault ...>*
```

One-way

```
<input .../>
```

Solicit-Response

```
<output .../>
<input .../>
<fault ...>*
```

Notification

```
<output .../>
```

Request-Response is the most commonly-used standard WSDL operation. A request is sent, the *operation* is executed, and a response is returned. While the *operation* is being executed, the client connection (the side that sent the request) will wait for the response. This can cause problems with HTTP, which has a timeout period, after which the socket will be reset with an error. Any number of *faults* (including none) can be specified. These dictate what sorts of error responses the client can expect to receive from the *operation*. It is much like the *throws* specifier in Java and C++.

One-way is other standard supported WSDL *operation* type. Basically, a request is sent and the *operation* is executed, but the client need not wait around for a response, because no response will ever be sent. This is useful with *operations* that are performing some asynchronous operation that does not need to return a success condition, or any data.

Solicit-Response and Notification operations are only supported by extensions to the WSDL bindings or by specialized WSDL implementations.

These diagrams illustrate the differences between the operation types:

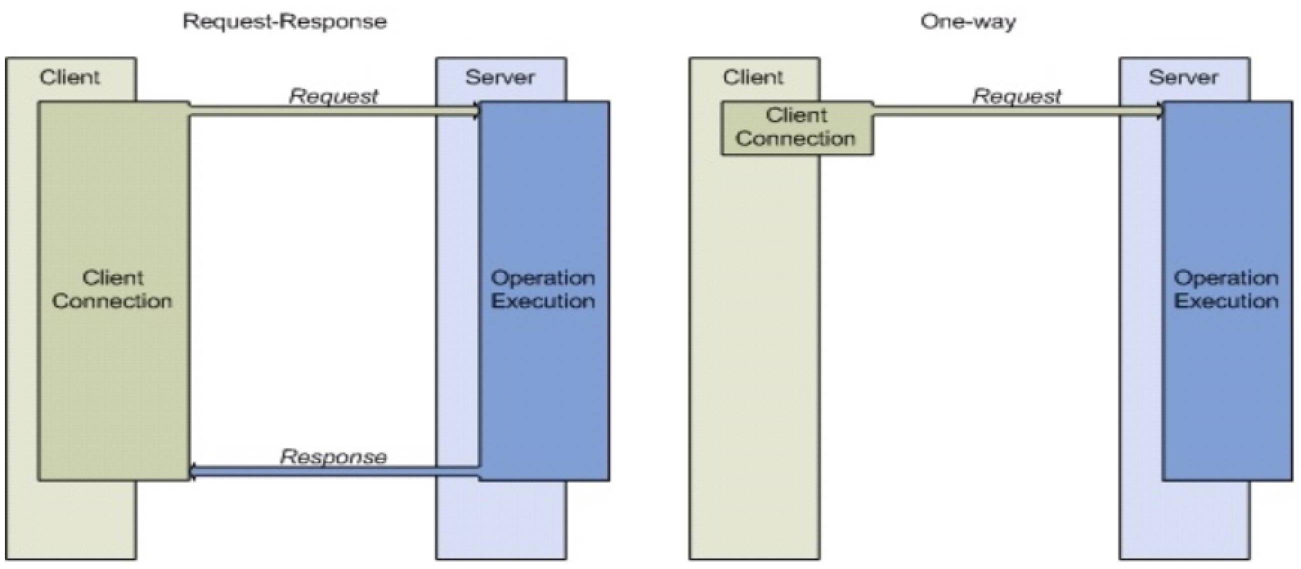

Figure 1: The Operation Types

Bindings

The *binding* section attaches an abstract interface to a concrete messaging structure. By far, the most common type of *binding* is a SOAP binding (discussed below in the Guide: SOAP article). But basically, the binding section of a WSDL has as its first child element, a concrete *binding* element. The binding element namespace dictates the concrete binding to use. Different concrete binding elements expect different attributes. The next elements under the WSDL binding are the *operations*. These should match the *operations* specified in the *port* type element. Under each *operation* is a concrete operation element. The concrete operation element's namespace and structure is dictated by the specific concrete implementation (such as the SOAP Binding namespace in our example). The same input, output, and fault elements should be present here as were declared in the port type section above.

These concrete bindings are all wrapped up into a named *binding*, which will be exposed on a port (see the next section).

```
<binding name="RecordBindings" type="RecordOperations">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="addRecord">
    <soap:operation style="document" soapAction="addRecord"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="deleteRecord">
    <soap:operation style="document" soapAction="deleteRecord"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

This WSDL defines a *binding* called *RecordBindings*, tied concretely to a SOAP binding and using HTTP as its transport mechanism. The style set in the *soap:binding* here is applicable to all of the *operations* for this *binding*, unless a concrete SOAP binding overrides the setting individually. This *binding* contains within it two *operations: addRecord*, and *deleteRecord*, which were defined in the *port* type section. Both of these *operations* are bound concretely as SOAP operations and will be sent in *document* style. The *input* and *output* for both *operations* are both bound to SOAP bodies and will be sent without any encoding (*literal*). Neither *operation* defines any specific *faults* that they will send. Note that the *operations* set their own *style*, although they match *document* style set in the *soap:binding*. In this case, the *style* attributes in the *operations* can be removed, but the *style* CAN be overridden at an operation level if desired. Another point of interest is to notice that different *operations* can specify different *styles*, *encodings*, and whether to use parts of elements or of types, than other *operations*, or even different options in the request than the response. However, it is highly suggested that all of the SOAP options be kept consistent for ease-of-use, maintainability, and compliance.

The service section of the WSDL exposes this binding to the outside world.

Services and Ports

In order to make an *operation* available to the outside world, it must be exposed by a *port*. A *port* in a WSDL and a port in TCP are similar concepts. In TCP, you can have multiple *ports* on an IP that are entry points to services on a single machine. In WSDL, one server can expose *operations* on different *ports*. These *ports* are then bound with a concrete address element, which, like the *bindings* above, are declared in a different namespace and have their own attributes. The most commonly used address is the SOAP *address* as in this example:

```
<service name="RecordService">
  <port name="RecordServicePort" binding="RecordBindings">
    <soap:address location="http://localhost:8090/RecordOps"/>
  </port>
</service>
```

This WSDL snippet creates a WSDL *service* named *RecordService* and populates it with one *port* named *RecordServicePort.* This *port* ties the *RecordBindings* binding declared above to an HTTP *address: http://localhost:8090/RecordOps*. A WSDL service can have multiple *ports*, which can tie different *binding* objects to different *addresses*. This WSDL *service* is the external endpoint used to access the ports and the operations defined in the WSDL.

For a SOAP *address*, the *port* ties a *binding* to an HTTP URI. This URI is sent in the HTTP message as the *location*. When that *location* is received, the server knows which *binding* is attached to that *address*, and by the SOAP *message* (either through RPC, or through the soapAction header), the server knows which *operation* in that *binding*. Thus, it follows that only one *binding* can be mapped to a particular *location*, because otherwise the system wouldn't know which *binding* to pick. However, a single *binding* can be mapped to multiple *addresses*, because the system would still know which *binding* to use based on the *address*.

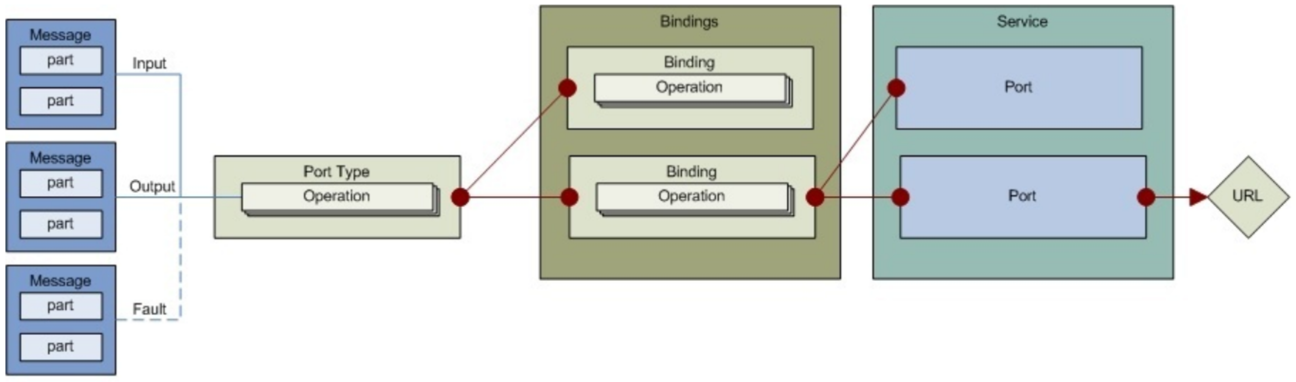

Figure 2: The Connected Pieces of the WSDL Puzzle

References

WS-I Profile 1.2: http://www.ws-i.org/Profiles/BasicProfile-1.2.html

WSDL 1.1 spec: http://www.w3.org/TR/wsdl

SOAP 1.1 spec: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

XSD data types spec: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html

Guide to choosing WSDL styles: http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/