

Introduction to WSDL

Web Services Description Language is an XML-based language used to define Web services and describe how to access them. Fortunately, you do not need to learn all the nitty gritty details because there are tools that generate WSDL for you. This article gives you just enough WSDL knowledge to understand what's going on and to be able to tweak tool-generated WSDL files and troubleshoot WSDL-related problems. For example, you'll need to know WSDL if your Web service methods accept objects as parameters. This is because you'll need to define the data types corresponding to those objects in the service's WSDL file. To get the most benefit out of this article you'll need to understand XML Namespaces and the basics of XML Schema Definition Language. You can start by reading my [XML Namespaces for VB programmers](#) tutorial. I'll be writing a tutorial on XSD soon, so be sure to check back here soon.

As a VB programmer, you probably know that type libraries are used to describe COM components. A type library contains information about the component's unique identifier (the CLSID), the interfaces that the component implements, and the method signatures for each interface. An application trying to invoke a method on this COM component uses the type library to know which component to instantiate and how to call it. Think of Web services as COM components (please note: Web services do **not** have to be COM components, but the analogy helps VB developers capture the essence of WSDL). WSDL is used to describe the Web service, specify its location, and describe the operations (i.e. methods) it exposes, similar to how a type library is used to describe a COM component. In the remainder of this article, I'll explain the fundamentals of WSDL and walk you through an example of a WSDL file generated by the [Microsoft SOAP Toolkit V2 RC 0](#).

Figure 1 shows an example Web service and a client invoking it in two different ways: Using SOAP and using HTTP GET. Each invocation consists of a request and a response message. Figure 2 shows the same example with WSDL terminology pointing to the various things that WSDL describes. You'll want to refer to figure 2 to visualize the WSDL elements as I explain them throughout this article.

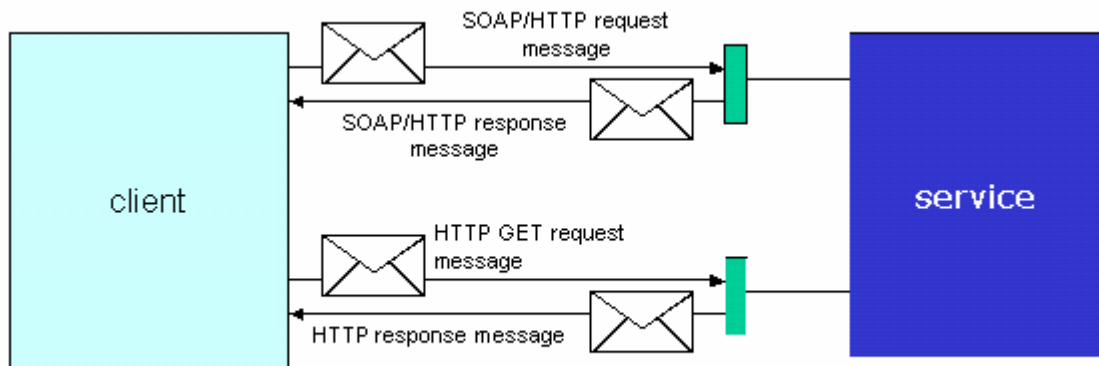


Figure 1. A client invoking a Web service.

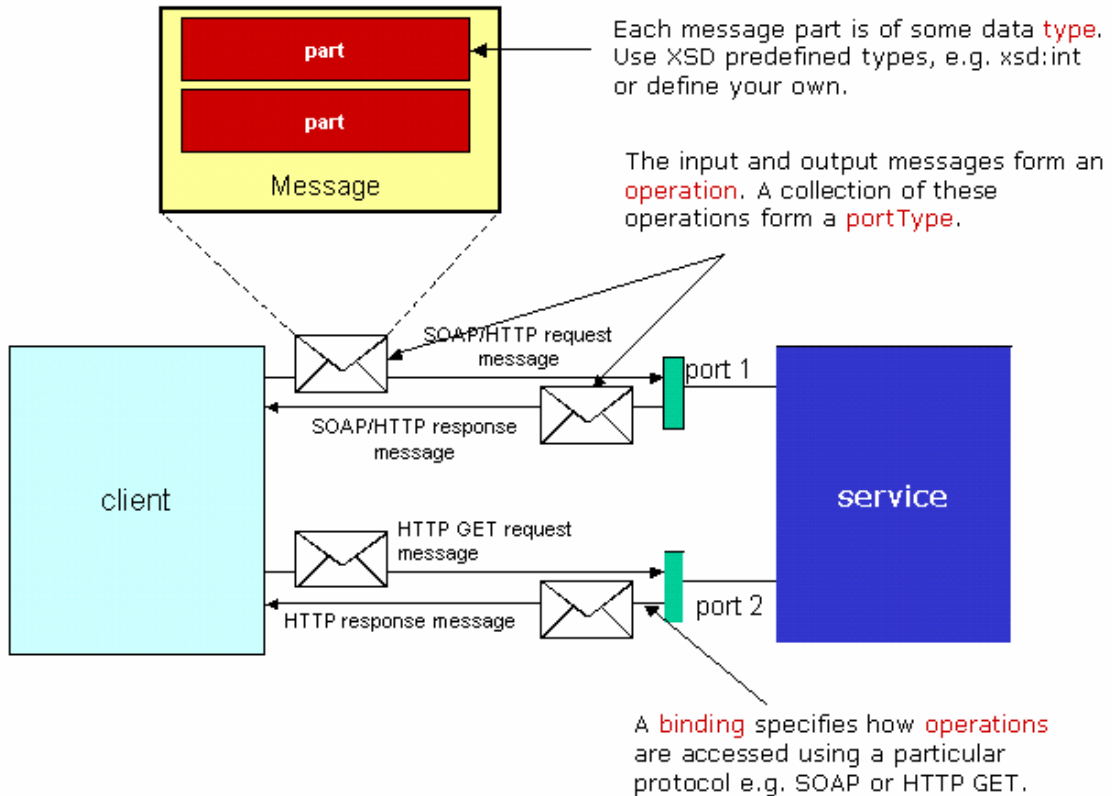


Figure 2. WSDL terminology used for describing Web services.

Defining Services

I built a simple VB class to use for this article. The class is called `Weather` and has one method (for now) called `GetTemperature`:

```
Public Function GetTemperature(ByVal zipcode As String, _  
                               ByVal celsius As Boolean) As Single  
    'just sends a hardcoded value for now  
    If celsius Then  
        GetTemperature = 21.7  
    Else  
        GetTemperature = 71.06  
    End If  
End Function
```

You can think of the class as the Web **service** and the `GetTemperature` method as an **operation** on that service. To describe this service, you use the WSDL `<service>` element. All WSDL elements belong to the WSDL namespace, which is defined as:

<http://schemas.xmlsoap.org/wsdl/> (Note that at the time of this writing, section 1.2 of the WSDL 1.1 specification has a typo where it defines this namespace). As an example, consider a service that you call `weatherservice`, the service would be defined using WSDL like this:

```
<definitions name='weatherservice'  
  xmlns='http://schemas.xmlsoap.org/wsdl/' >  
  <service name='WeatherService' >  
    .....  
  </service>  
</definitions>
```

Visit www.LearnXmlWS.com for more resources

The <definitions> element is the root element of the WSDL document. Here, we declare the WSDL namespace as the default namespace for the document so all elements belong to this namespace unless they have another namespace prefix. I omitted all other namespace declarations from this example to keep it clear.

Each service is defined using a service element. Inside the service element, you specify the different ports on which this service is accessible. A port specifies the service address, for example, `http://localhost/demos/wsd1/devxpert/weatherservice.asp`

The port definition would be like this:

```
<port name='WeatherSoapPort' binding='wsdl:ns:WeatherSoapBinding' >
  <soap:address
    location='http://localhost/demos/wsd1/devxpert/weatherservice.asp' />
</port>
```

Each port has a unique name and a binding attribute. We'll discuss the binding attribute later in this article. When using SOAP, the port element contains a `<soap:address/>` element with the actual service address. Here, the soap namespace prefix refers to the namespace `http://schemas.xmlsoap.org/wsdl/soap/`

This namespace is used for SOAP-specific elements within WSDL. Such elements are also known as WSDL SOAP extension elements. We'll see some more examples of WSDL extension elements throughout this document.

A Web service does not have to be exposed using SOAP. For example, if your Web service is exposed via HTTP GET, the port element would contain an `<http:address/>` element similar to this:

```
<http:address location="http://localhost/demos/wsd1/devxpert/weatherGET.asp"/>
```

A Web service may be accessible on many ports. For example, you might make your service available via SOAP and HTTP GET and possibly even via SMTP. For this Web service, you would have three ports each one with a different name.

What's Your Message

We'll make a transition now and start discussing how to define your service's request and response messages. A message is protocol independent, that is, a message may be used with SOAP, HTTP GET, or any other protocol. To use Web services in a remote procedure call (RPC) model, there are two messages you must describe. There's the input or request message, which is sent from the client to the service and there's the output or response message, which is sent back the opposite way. When you're using SOAP, keep in mind that the word message here refers to the payload of the SOAP request or response. That is, the message does not include the SOAP envelope, headers, or fault. The WSDL spec does not specify a naming convention for messages. You can call the messages whatever you like using their name attribute. You'll probably use a tool to generate the WSDL and that tool will probably follow its own naming convention for messages.

To describe the message structures, you use the WSDL <message> element. Each <message> contains zero or more <part> elements. A <part> corresponds to a parameter or a return value in the RPC call.

The request message will contain all ByVal and ByRef parameters and the response message will contain all ByRef parameters as well as the return value if the service returns something (i.e. if it's a Function not a Sub). Each <part> must have the same name and data type as the parameter it represents (this naming rule is part of the SOAP specification not WSDL, but you'll be using SOAP most of the time so that's what I'm focusing on). For example, the GetTemperature method would correspond to two messages: A request message sent from client to service and a response message sent back to the client:

```
<message name='Weather.GetTemperature' >
  <part name='zipcode' type='xsd:string' />
```

Visit www.LearnXmlWS.com for more resources

```
<part name=' Celsius' type=' xsd: boolean' />
</message>
<message name=' Weather.GetTemperatureResponse' >
  <part name=' Result' type=' xsd: float' />
</message>
```

You'll note that the data types are prefixed with the xsd namespace prefix (assuming it was declared earlier in the document). XSD defines many data types that you can draw from when defining the message parts. The Microsoft SOAP Toolkit's documentation lists the supported XSD types and their mapping to VB data types. This list is repeated in table 1 for your convenience. Note that there are more data types in XSD than in VB that's why they're listed by the XSD type first.

XSD (Soap) Type	VB	Comments
anyURI	String	
base64Binary	Byte()	
boolean	Boolean	
byte	Integer	Range validated on conversion.
date	Date	Time set to 00:00:00
dateTime	Date	
double	Double	
duration	String	No validation or conversion performed
ENTITIES	String	No validation or conversion performed
ENTITY	String	No validation or conversion performed
float	Single	
gDay	String	No validation or conversion performed
gMonth	String	No validation or conversion performed
gMonthDay	String	No validation or conversion performed
gYear	String	No validation or conversion performed
gYearMonth	String	No validation or conversion performed
ID	String	No validation or conversion performed
IDREF	String	No validation or conversion performed
IDREFS	String	No validation or conversion performed
int	Long	
integer	Variant	Range validated on conversion.
language	String	No validation or conversion performed
long	Variant	Range validated on conversion.
Name	String	No validation or conversion performed
NCName	String	No validation or conversion performed
negativeInteger	Variant	Range validated on conversion.
NMTOKEN	String	No validation or conversion performed
NMTOKENS	String	No validation or conversion performed
nonNegativeInteger	Variant	Range validated on conversion.
nonPositiveInteger	Variant	Range validated on conversion.
normalizedString	String	
NOTATION	String	No validation or conversion performed
number	Variant	
positiveInteger	Variant	Range validated on conversion.
QName	String	No validation or conversion performed
short	Integer	
string	String	
time	Date	Day set to December 30, 1899
token	String	No validation or conversion performed
unsignedByte	Byte	
unsignedInt	Variant	Range validated on conversion.
unsignedLong	Variant	Range validated on conversion.

unsignedShort	Long	Range validated on conversion.
----------------------	------	--------------------------------

This extensive list of XSD types is sufficient for all your simple data type needs. However, if your service uses User Defined Types, you'll need to define those types in WSDL yourself. You define these types using XSD just as you would when creating a regular XSD schema. I'll save the details of how you do this for another article.

Port Types and Operations

If you've been following closely, you'll note that just defining your messages does not tie them together as a request-response pair corresponding to a method call. To do this you define operations using the WSDL `<operation>` element. An operation specifies which message is the input and which message is the output like this:

```
<operation name='GetTemperature' parameterOrder='zipcode celcius' >
  <input message='wsdl ns: Weather. GetTemperature' />
  <output message='wsdl ns: Weather. GetTemperatureResponse' />
</operation>
```

The `parameterOrder` attribute is optional and may be used to specify a space-delimited list of part names to indicate the order of parameters when making the RPC call. Inside the `<operation>` you specify `<input>` and `<output>` elements. Each refers to the corresponding message by its fully qualified name, e.g. `wsdl ns: Weather. GetTemperature`. The collection of all operations (i.e. methods) exposed by your service is called a `portType` and is defined using the WSDL `<portType>` element like this:

```
<portType name='WeatherSoapPort' >
  <operation name='GetTemperature' parameterOrder='zipcode celcius' >
    <input message='wsdl ns: Weather. GetTemperature' />
    <output message='wsdl ns: Weather. GetTemperatureResponse' />
  </operation>
  <!-- other operations would go here -->
</portType>
```

So the `<operation>` element is a child of `<portType>`. You can call the `portType` whatever you want.

Binding It All Together

If you're familiar with SOAP, you'll notice there are still some more details we must specify (if you're not familiar with SOAP, stay tuned for my upcoming SOAP article, meanwhile, take my word for it). For example, `GetTemperature` takes a string and a Boolean, how do you physically represent these in the request message? If the Boolean is True, do you represent it as `-1`, or `1`, or maybe `TRUE`? Therefore, you must specify how your service expects the data to be encoded. The SOAP specification contains predefined rules for doing this, so you don't have to invent your own (that's a relief), but you must indicate that your service is abiding by those rules.

We are now making a transition from abstract data types, messages, and operations to concrete physical representation of messages on the wire. To define the concrete aspects of operations, you use the WSDL `<binding>` element:

```
<binding name='WeatherSoapBinding' type='wsdl ns: WeatherSoapPort' >
  ...
</binding>
```

Again, the name of the binding can be whatever you want. However, you must use this same name for the binding attribute on the `<port>` element (see `Defining Services` above). Inside the `<binding>` element you have a WSDL SOAP extension element called `<soap:binding>` which is used to specify the transport protocol you're using (SOAP can be used over HTTP, SMTP, or possibly any other transport) and the style of request (`rpc` and `document` are the two styles). For example:

Visit www.LearnXmlWS.com for more resources

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

Then for each operation that this service exposes, you specify the value of the SOAPAction HTTP header. If you're not familiar with SOAP, SOAPAction is an HTTP header that the client sends when it invokes the service. The SOAP server may use this header to determine the service or use it in any other way it needs to. You specify the SOAPAction like this:

```
<binding name='WeatherSoapBinding' type='wsdl:wsdl:WeatherSoapPort' >
  <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
  <operation name='GetTemperature' >
    <soap:operation
      soapAction='http://tempuri.org/action/Weather.GetTemperature' />
    .....
  </operation>
</binding>
```

Basically, you add an <operation> element with the same name as the operation you defined earlier. Within this <operation> you add a <soap:operation> with the soapAction attribute. Finally, as I mentioned earlier, you must specify how the input and output messages of this operation are encoded, the complete binding looks like this:

```
<binding name='WeatherSoapBinding' type='wsdl:wsdl:WeatherSoapPort' >
  <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
  <operation name='GetTemperature' >
    <soap:operation
      soapAction='http://tempuri.org/action/Weather.GetTemperature' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
</binding>
```

Within the <operation> you add an <input> and an <output> element and use a <soap:body> element within each to specify how the data is encoded. The URI <http://schemas.xmlsoap.org/soap/encoding/> indicates the SOAP encoding style as described in the SOAP 1.1 specification. Now the WSDL file is complete and you can start using it from SOAP clients.

Summary

In this article, I have explained the basics of WSDL and how it is used to describe Web services that use SOAP. As I mentioned at the beginning of the article, you probably won't have to create an entire WSDL file from scratch, however you're likely to tweak the ones that tools generate for you. My next article will build on this one by showing you how to use the Microsoft SOAP Toolkit V2 RC 0 to expose and invoke Web services using SOAP.