Deploying web services with WSDL: Part 1

Introduction to web services and WSDL

Bilal Siddiqui

November 01, 2001

In the Deploying web services with WSDL series, Bilal will explore all major technical aspects of creating, deploying, and publishing web services -- from Web Services Markup Language (WSDL), to Simple Object Access Protocol (SOAP), and Universal Description, Discovery, and Integration (UDDI) registries. Part 1 focuses on WSDL authoring: You will learn how to manually create a WSDL interface, and then compare your effort with the output of a WSDL authoring tool.

The idea of having interoperable web-based distributed applications is not new. As just one example, the requirements of the Electronic Data Interchange (EDI) market emerged well before B2B on-line e-commerce gained any significant presence -- and with the popularity of the B2B marketplace, interoperability has become the most compelling EDI requirement.

Take any online electronic Marketplace as an example. There are a lot of businesses, each offering its own services (let's call them web services). In present-day e-commerce, there is no mechanism that allows one business to discover automatically the services that its prospective partners offer. The so-called next generation dotcom will offer a mechanism for exactly this kind of automated discovery.

What is WSDL?

This new breed of dotcom needs a solution that can describe the services -- the web services -- it offers. Specifically, this means that you need a format or some type of grammar, with which you can describe the answers to the following questions:

- What are the services offered in your online business?
- How can you invoke your business services?
- What information do your business services need from the user when he or she invokes your service?
- How will the user provide the required information?
- In which format will the services send information back to the user?

Happily, WSDL provides the mechanism for doing all of these jobs.
WSDL and SOAP

To better understand how WSDL works, I will first describe how SOAP and HTTP work with WSDL. The purpose of WSDL is to "describe" your web services. Businesses will exchange WSDL files to understand the other's services. SOAP comes in once you know your partners' services and wish to invoke them. You can think of services as objects which are accessed by SOAP.

Most likely you will be communicating with potential partners via the Internet or through e-mail. The Internet, of course, uses HTTP and e-mail works on SMTP, making HTTP and SMTP the favored candidates for acting as "transport service providers" to SOAP.

WSDL Authoring

Now I'll look at the process of writing WSDL for the web service. The goal is to expose the existing web services. Your own situation may be any one of the following:

- You have an existing service (for instance, a web site) and you want to expose its functionality.
- You have a WSDL and you want to implement web server-side logic according to what you have already decided to expose. (Some people may think this an unlikely scenario, but UDDI's idea of fingerprints makes it quite probable; I will discuss UDDI in the fourth part of this series of articles).
- You are starting from scratch and have neither a web site nor a WSDL interface.

The information covered in this article allows for any or all of these possibilities.

Four steps to WSDL authoring

I will divide WSDL authoring in four simple steps. Follow each step, and your web service will be ready for deployment.

Step 1: The service interface

As a sample project, you will build the service interface of a mobile phone retail company (I will call this service MobilePhoneService). This company sells mobile phones in different models, so back-end data storage of this company's web service will contain a single table with two columns, model number and price. (I am keeping this simple in order to maintain the focus on WSDL itself). You will have two methods on your service which you will expose using WSDL:

- getListOfModels ()
- getPrice (modelNumber)

The getListOfModels method provides an array of strings, where each string represents the model number of a mobile phone. getPrice takes a model number and returns its price. WSDL calls these methods as operations. Now you'll start building the WSDL Interface file.

The root element of every WSDL file is <definitions>, in which you must provide a complete description of services. First of all, you have to provide various namespace declarations in the
<definitions> element. The three external namespace declarations you have to make are WSDL, SOAP, and XSD (XML Schema Definition). There is another namespace, TNS, that refers to your MobilePhoneService (this means TNS -- which is short for targetNamespace -- will contain all names of elements and attributes that will be defined specifically for the MobilePhoneService). But WSDL is the primary namespace that you'll use in most of your WSDL authoring. I will mention the utility of other namespaces as they are used in this series of articles.

Just a note about namespaces: WSDL uses the concept of namespaces extensively. I encourage you to visit W3C's official web site to learn more about namespaces (see Related topics). WSDL is an implementation of this idea, since namespaces provide an infinite degree of flexibility and this is exactly what is required in a portable format for electronic data interchange.

The <definitions> element contains one or more <portType> elements, each of which is actually a set of operations that you want to expose. Alternatively, you can think of a single portType element as a logical grouping of methods into classes. For example, if your supply chain management solution requires interaction with both customers and suppliers, you will most probably define functionality for interaction with them separately; that is, you will define one portType for customers and one for suppliers. You should call each portType a service, so that your complete WSDL file will become a collection of services.

You have to provide a name for each service. In this case, you have only one service (so one <portType> element). You need to use the name attribute of this portType element to assign a name to your mobile phone sales service.

Within each service you may have several methods, or operations, which WSDL refers to via <operation> elements. The sample application has two methods to expose, namely getListOfModels and getPrice. Therefore, you need to provide two <operation> elements, each having a name. I have used the name attribute of the <operation> element to name each operation.

At this point the WSDL file looks like Listing 1.

**Listing 1: Defining operations**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="MobilePhoneService"
    targetNamespace="http://www.mobilephoneservice.com/MobilePhoneService-interface"
    xmlns="http://schemas.xmlsoap.org/wsdl/
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:tns="http://www.mobilephoneservice.com/MobilePhoneService"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">

    <portType name="MobilePhoneService_port">
        <operation name="getListOfModels ">
            .......
            .......
        </operation>

        <operation name="getPrice">
            .......
            .......
        </operation>
    </portType>
</definitions>
```
Step 2: Making parameters

Having defined the operations (or methods), you now need to specify the parameters that you will send to them and the parameters that they will return. In WSDL terms, all parameters are called "messages." It is useful to think that you are sending in messages and as a result getting back return messages. Method calls are the operations that take place to prepare return "messages" in response to incoming messages.

Recall from the first step that you have two operations to expose. The first operation, `getListOfModels`, does not take any parameter and returns an array of strings, where each string represents the model number of a mobile phone. Therefore, you have to define a `<message>` element that contains an array of strings.

Have a look at the various `<message>` elements in Listing 2. The first of these has a name attribute equal to `ListOfPhoneModels` (a logical name for this message), and a single `<part>` element with the name models, which means `ListOfPhoneModels` is a one-part message, where the name of the only part present is "models." You can have any number of parts in a message -- so long as you remember to give them different names for unique identification.

I have included another attribute of the `<part>` element, which is `type`. Think of this "type" attribute as data types in C++ or Java. I have specified the data type of `models` as tns:Vector. (Recall that I specified a few namespaces in the root `<definitions>` element, one of which was `tns`). This refers to the `MobilePhoneService` namespace. What this means is that you can create your own namespace while authoring WSDL. You may now be asking two logical questions: Why? And how?

To answer the why, let's take the array of strings returned by the `getListOfModels` operation as an example. WSDL uses a few primitive data types that XML Schema Definition (XSD) defines (like int, float, long, short, byte, string, Boolean, etc.) and allows you to either use them directly or to build complex data types based on these primitive ones, before using them in messages. This is why you need to define your own namespace when referring to complex data types. In this case, you need to build a complex data type for an array of strings.

Now coming to the how question, you will use XSD to create your namespace. For this purpose I have used the xsd:complexType element within the `<types>` element to define data type named `Vector`. `Vector` uses two primitive data types -- string (element data) and Integer (element count). Hence `Vector` becomes part of the namespace and can be referred to by the alias `tns`.

In a similar manner, I have defined the other two messages, `PhoneModel` and `PhoneModelPrice`, in Listing 2. These two messages use only string primitive data types of the xsd namespace and therefore you do not need to define any more complex data types in order to use them.

You may have noticed that while creating the `<message>` elements, you did not specify whether these messages are incoming parameters or return values. This is a job you will take care of in the `<operation>` element within the `<portType>` element. Therefore, as you can see in Listing 2, I have added the `<input>` and `<output>` elements to each of the two operations. Each input
element refers to a message by its name and treats it as a parameter that the user will provide when invoking this operation. Each <output> element similarly refers to a message; it treats the message as the return value of the operation call.

Listing 2 fits the discussion so far nicely into one frame.

**Step 3: Messaging and transport**

I have defined operations and messages in an abstract way, without worrying about the details of implementation. In fact, WSDL's job is to define or describe web services and then to provide a reference to an external framework to define how the WSDL user will reach the implementation of these services. You can think of this framework as a binding between WSDL's abstract definitions and their implementation.

Currently, the most popular binding technique is to use the Simple Object Access Protocol (SOAP). WSDL will specify a SOAP server that has access to the actual implementation of your web service, and from there it is entirely SOAP's job to take the user from the WSDL file to its implementation. SOAP is the topic of next installment in this series of articles, so for the time being I will avoid SOAP details and keep focused on WSDL authoring.

The third step in WSDL authoring is to describe the process of SOAP binding with a WSDL file. You will include a <binding> element within the <definitions> element. This binding element should have a name and a type. The name will identify this binding and type will identify the portType (set of operations) that you want to associate with this binding. In Listing 3, you will find that the name of the <portType> element matches the type attribute value of the <binding> element.

The WSDL binding element contains a declaration of which external technologies you will use for binding purposes. Since you are using SOAP, you will use SOAP's namespace here. In WSDL terminology, the use of an external namespace is called the extensibility element. In Listing 3, you will see an empty <soap:binding/> element. The purpose of this element is to declare that you are going to use SOAP as a binding and transport service.

The <soap:binding> element has two attributes: style and transport. Style is an optional attribute that describes the nature of operations within this binding. The transport attribute specifies HTTP as the lower-level transport service that this binding will use.

A SOAP client will read the SOAP structure from your WSDL file and coordinate with a SOAP server on the other end, so you must be very concerned with interoperability. I intend to cover this in detail in the third part of this series of articles.

After the empty <soap:binding/> element, you have two WSDL <operation> elements, one for each of your operations from Step 1. Each <operation> element provides binding details for individual operations. Therefore, I have provided another extensibility element, namely <soap:operation/> (again an empty element that relates to the operation in which it occurs). This <soap:operation/> element has a soapAction attribute that a SOAP client will use to make a SOAP request.
Recall from Step 2 that the \texttt{getListOfModels} operation only has an output and does not have any input. Therefore, you have to provide an \texttt{<output>} element for this operation. This output contains a \texttt{<soap:body/>} element (again an empty element that relates to the output in which it occurs). The SOAP client needs this information to create SOAP requests. The value of the namespace attribute of \texttt{<soap:body/>} should correspond to the name of the service that you will deploy on your SOAP server in the next part of this series of articles.

You are nearly finished with Step 3. Just copy the next operation after this one and you will come up with \texttt{Listing 3}.

**Step 4: Summing it up**

You have produced a WSDL file that completely describes the interface of your service. WSDL now requires the additional step of creating a summary of the WSDL file. WSDL calls this a an implementation file, which you will use while publishing your web service at a UDDI registry in the fourth part of this series of articles. Have a look at \texttt{Listing 4}, a WSDL implementation file. Its main features are the following:

- The root \texttt{<definitions>} element is exactly the same as in \texttt{Listing 3} (a WSDL interface file), except that \texttt{Listing 4} (the implementation file) refers to a different \texttt{targetNamespace}, which refers to your implementation file.
- There is an \texttt{<import>} element that refers to the interface file of \texttt{Listing 3} (file name MobilePhoneService-interface.wsdl) and its namespace.
- There is a \texttt{<service>} tag with a logical \texttt{name} for this service. Within the service element is a port element that refers to the SOAP binding that you created in \texttt{Listing 3}.

**Using IBM's Web Services ToolKit (WSTK) for WSDL authoring**

The web service is now completely ready for deployment. I have shown how to create these files manually (using a simple text editor like emacs). These same files can be generated using web services authoring tools like IBM's WSTK (see Related topics for links to the toolkit, and other resources mentioned in this article).

WSTK can generate these files using a wizard-assisted process. Users can generate WSDL files for the same two methods that I demonstrated in the above tutorial and compare WSTK files with the WSDL files of \texttt{Listings 3 and 4}.

You will notice following differences:

- WSTK creates all name attributes according to a logical formula; in the example, I used names of my own convenience.
- WSTK generates at least one input tag for each operation, even if that operation does not take any input. The \texttt{listAllPhoneModels} operation did not have any input element, but if you generate the same file with WSTK, it will contain an empty input element for this method.
- WSTK produces a third file in addition to the two files that were produced. This third file is a SOAP deployment descriptor that the SOAP engine uses for service deployment. I will discuss service deployment in the article of this series.
In this installment I have demonstrated manual WSDL authoring to create interface and implementation files, and compared the files with those that IBM's Web Services ToolKit produces. In the next part of this series, I will discuss deployment of this WSDL service on a SOAP server.
Related topics

- Visit W3C's official web site to find the Web Services Description Language (WSDL) 1.1 specification and all other XML-related official specifications including technical documentation for XSD and namespaces.
- Visit IBM's alphaWorks web site to download the Web Services ToolKit (WSTK) used in this article.
- Download Apache's SOAP toolkit from Apache.org.
- Building Web Services: Making Sense of XML, SOAP, WSDL, and UDDI is a new book from Steve Graham, Simeon Simeonov, Toufic Boubez, Glen Daniels, Doug Davis, Yuichi Nakamura, Ryo Neyama -- a group of authors from all corners of the web services technology sector. (Sams publishing, 2001).
- Read this article on developerWorks that describes how to map WSDL elements to a UDDI registry.

© Copyright IBM Corporation 2001
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)