Windows .NET Web Service and WSDL

VICTORIAN PARTNERSHIP FOR ADVANCED COMPUTING

Table of Contents

1. Document Purpose

An XML web service stub and its client stubs can be generated from a web service's WSDL file. This WSDL file can be remotely extracted from the web service. A web service's wsdl file can therefore be used to clone a web service, especially on another platform. (*This only clones the web interface and interface classes, not the underlying code though*.) It can also be used to generate proxy clients for the service on computing platforms different to that of the web service. The wsdl file can therefore be the centrepiece of interopabililty.

A key issue is change management when the original web service changes. How to update the clients and other web services based upon it needs careful resolution.

This document discusses how to get the WSDL file and generate web client and service components from it. It also considers how to manage change in components generated form it when the WSDL file changes.

2. Terminology

- WSDL: Web Services Description Language

WSDL provides an extensible mechanism for defining the base messaging description and metadata for a Web service

Author: David Jones

Date printed: 12/6/2010
Rev: 1.0

Victorian Partnership for Advanced Computing Ltd
110 Victoria St, Carlton Sth, VIC 3053
PO Box 201, Carlton Sth, VIC 3053
ABN: 59 093 732 426

3. About WSDL

"**Web Services Description Language (WSDL**) *is an XML grammar that defines the functionality offered by a Web service and the format of messages sent and received by a Web service. A Web service's WSDL document defines what services are available in its Web service. The WSDL document also defines the methods, parameter names, parameter data types, and return data types for the Web service. An application that uses a Web service relies on the Web service's WSDL document to access the Web service's features. An application that uses a Web service can be built on any platform that supports Web services."*

[Link 1]

4. WDSL Details

A web service exists as specific URL on a web server. On a Windows IIS web site, it exists in .NET as a .asmx file. Messages are generated by the client as XML soap packets and sent from client applications using http POST to the web service URL and are the response is returned as SOAP packets. The client then consumes the response as an XML document. Responses can be either synchronous or asynchronous.

5. Interopability

Interoperability is an on-going challenge between SOAP implementations. If you want your service to work with other platforms and implementations, you do need to understand the issues. At issue is serialisation and deserialization of custom classes across a web service on one platform and a web service client on another platform.

When a data type is sent a soap packet via a web service interface, if it is one of the basic types such as integer, boolean, byte or string etc. it is simply serialized as an equivalent XSD type and then deserialised upon reception into the equivalent data type at that end of the pipe. The soap packet and wsdl does not need to indicate how to map that data type. XSD .NET and Java mapping are shown in the next two tables. For example a .NET DateTime type would serialize as xsd.datetime. Upon reception by a Java web service component it would be deserialized as java.util.Calendar.

XSD type	.NET type
xsd:int	Int32
xsd:unsignedByte	Byte
xsd:short	Int16
xsd:double	Double
xsd:decimal if length <= 28	Decimal
xsd:string if length > 28	String
xsd:string	String
xsd:dateTime	DateTime
xsd:int if lenrth <= 9	Int32
xsd:long if length > 9 and <= 19	long
xsd:string if length > 19	String
xsd:base64Binary	Byte[]
xsd:string	String

xsd:base64Binary	Byte[]

XSD type	Java type
xsd:base64Binary	byte[]
xsd:boolean	boolean
xsd:byte	byte
xsd:dateTime	java.util.Calendar
xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:QName	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

Its also possible to use a web service interface in an untyped manner and cast the object passed, at either end. Alternatively, it is possible to use metadata in the soap packet to specify the data type and use this upon reception. It is better though to "strongly" type the web service interface by using a WSDL file.

Suffice to say, with the right tools, this is all seamless (automatically handled by the generating software and compilers).

6. Getting a WSDL File from a Server

This can be obtained directly from the web service if security restrictions on the site don't block the required post to the site. Note that any custom classes used a web method parameters or return types are included in the wsdl file.

6.1. WSDL from a .NET XML web service.

This is obtained from the web service using http GET from its URL..

For example if http://time.world.com/time.asmx a .NET XML web service: http://time.world.com/time.asmx?wsdl posted in a web browser will return the web service's wsdl as XML. This can then be saved as a .wsdl file.

6.2. WSDL from a Linux web service.

As above except the web service URL doesn't have the file extension.

For example if http://time.world.com/time is an XML web service:

http://time.world.com/time?wsdl posted in a web browser will return the web service's wdsl as XML. This can then be saved as a .wsdl file.

7. The Windows .NET WSDL Tool

- **wsdl.exe**
- Do a search for it under Program Files directory.
- Typically in SDKs such as : C:\Program Files\\Microsoft SDKs\Windows\v7.0A\Bin
- I add it to my path.
- Ref: [Link 2]

The Web Services Description Language tool generates code for XML Web services and XML Web service clients from WSDL contract files, XSD schemas, and .discomap discovery documents

This tool is used to both generate web services and to generate web service clients. In both cases it creates the web service as an abstract class. It can be coded in C# or VB.

`wsdl [options] {URL | path}`
Options primarily specify whether a client or server is required.
The original web service URL can be specified or a path the saved wsdl can be used. I assume that Visual Studio uses this tool (or similar functionality) when you add a web reference to a project.

8. Generating a .NET Web Service from a WSDL File: Windows

8.1. Get the proxy class

wsdl **/server /l:cs c:\tem\webserice.wsdl**

- If the web service in the file is called custom, then it will generate custom.cs
- Credentials (username and password) may be required [See Link 2]
- /l:vb instead of /l:cs would generate a VB file
- /out:websvrc.cs instead of /l:cs could also be used, or /out:websvc.vb

wsdl **/server /l:cs http://webhost/webservice/webserice.asmx?wsdl**

- Would generate the .cs file as above from the original web service.

8.2. Generate the web service

You then create a new web service in Visual Studio that implements those stubs:

- Create a new .ASPX project.
- Add the custom.cs to the project
- Add a new web service file customSvc.asmx
- Simply generate the class code form the abstract source:

- Remove any code inside the customSvc class.
- Base the new webs service on the custom class:
  - Change the class specification from something like:

```
public abstract partial class custom : System.Web.Services.WebService
```

  - To:

```
public class customSvc : custom
```

  - Right click on custom and choose Implement Abstract Class
  - For each implemented method add the [WebMethod] directive above the method. Eg:

```
[WebMethod]
public override string RunJob(cwsLoginInfo infoLogin)
```

  - Note that the required web method stubs now have:

```
throw new NotImplementedException()
```

    inside.
- Each method can now be coded.
  - Make sure that the abstract class and the implemented web service are in the same namespace.
  - Any custom classes can be directly used in the implementation as they are in the abstract class file.
- Build the service and resolve any issues, eg using references

- Test: http://localhost/website/customSvc.asmx
  - Also http://localhost/website/customSvc.asmx?wsdl

9. Using a Web Service: .NET Windows

The simplest way is to add a web reference to the web service in a .NET application or .NET class.

Alternatively the wsdl file can be used to generate a proxy class:

1. This example produces a C# web service client proxy class directly from the web service:
wsdl /out:myProxyClass.cs http://hostServer/WebserviceRoot/WebServiceName.asmx?WSDL

2. This example outputs a C# web service cllent proxy class from a local wsdl file
wsdl /l:cs c:\temp\service.wsdl
File name depends upon the web service class in the wsdl file

   - After creating the proxy class file, add it to a .NET project.
   - Add a reference in the project to System.Web.Services

Having got a .NET application or class with a web reference using either method, just create a new instance of the web service and use its methods in code.

10. Change Management

If a web service changes, either

(i) Methods are added or deleted

(ii) Method parameters change
(iii) Method return types change
OR
(iv) Custom types are changed, added or deleted

Q***. When could you avoid code changes in the client if the web service changes?***
A. Only in case (i) when only unused methods are removed or new methods are not needed.

10.1. What's changed

The changes in the wsdl may be able to be determined upon inspection. Alternatively you could use:

- A text file command line comparison application
- A Visual Text File comparison application
- A Visual XML file comparison application
- A code comparison application.
- Repository version comparison tools

<Do a search for these tools: To do Add some info about these>

Whist non XML comparison tool can be used for wsdl XML files, a visual XML comparison tool is better because differences are noted at an XML element level rather than line by line.

11. Updating an XML Web Service

When the wsdl changes, regenerate the proxy class file as 8.1.

Replace the old proxy class file with the new one. Then:

- Rebuild the web service and make required changes to resolve the introduced errors

  OR
- Update the web methods by:

  Right click on the web service class and choose Implement Abstract Class

  Resolve any issues

Or if the changes can be identified as minor

- Manually make the changes to the web service files (including the abstract class).

12. Updating the Web Client

12.1. Updating the client: Visual Studio & Web Reference

If using a web reference in Visual Studio, right click on the web reference and update it,
For each of the web service changes the corresponding changes to the client would be:

(i) *Method/s Added*: No action if the new method/s are not to be used.
If to be used, then add code that uses the web method.
Method/s Deleted: Remove/replace those references in the client.
(ii) Parameters changed: Adjust the parameters in the code that calls the web method.
(iii) Return types changed: Adjust the return type usage in the code that calls the web method.
(iv) *Custom classes changed, added or deleted:*
Change the custom class usage in the code.

12.2. Updating the client: Visual Studio & wsdl generated proxy class

- Regenerate the proxy class using wsdl tool and update the proxy class code in the project (or replace the file).
- Make any code changes as immediately above

13. References

1. WSDL Overview
   - http://msdn.microsoft.com/en-us/library/bb126207.aspx
2. WSDL Tool, MSDN
   - http://msdn.microsoft.com/en-us/library/7h3ystb6.aspx
3. Axis User's Guide (Linux)
   - http://ws.apache.org/axis/java/user-guide.html#InstallingAxisAndUsingThisGuide