

# WSDL Connecting Janus to .Net

2006 Sirius User Group  
Alan Brooks



# Web Services Description Language

- WSDL is a specification defining how to describe web services in a common XML grammar.
- A WSDL document is formatted and distributed as an XML document.



# WSDL: Four Critical Pieces of Data



- **Interface** information describing all publicly available functions
- **Data type** information for all message requests and message responses
- **Binding** information about the transport protocol to be used
- **Address** information for locating the specified service

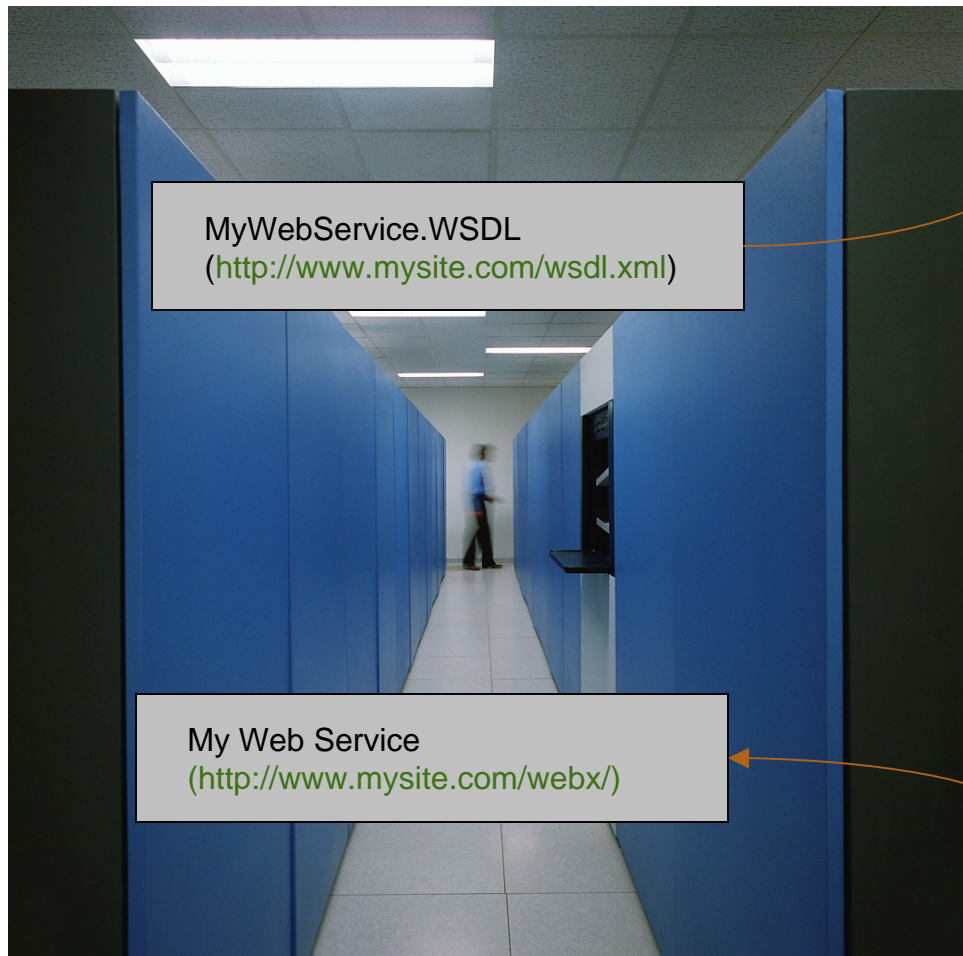


# WSDL in a Nutshell

- A contract between the service requestor and the service provider
- Similar to the way user interface represents a contract between client code and a server object.
- WSDL is platform and language independent and is used primarily (although not exclusively) to describe SOAP services.



# WSDL: What Part Does It Play?



The WSDL is retrieved by a code generator or a browser from it's location on the web

Glue, SOAP:Lite, .Net, Elbow Grease...

The WSDL Operations, message, data types and other specification are used to generate client classes and methods.



.Net, Java, PHP, etc.

The client sends and receives messages from the web service based on the WSDL contract.



# What Is It Good For?

- Provides both a human and machine readable description of your web service.
  - Amazon
  - eBay Price Watcher
  - Google
  - National Weather Service
  - Stock Quote Servers
  - Clients (.Net, Java, Websphere, etc.)



# Who Doesn't Need It?

- Anyone writing both the client and server, in-house, with no need to “explain” the service or data to outside parties.
- Non-Public web services.



# Automation

- Allows a client to locate a web service and invoke any of its publicly available functions.
- Integration of new services with little or no manual code.
- WSDL is a cornerstone of the web service architecture, because it provides a common language for describing services and a platform for automatically integrating those services.



# WSDL -- Six Major Elements

- Definitions -- root WSDL element
  - Types: What data types will be transmitted?
  - Message: What messages will be transmitted?
  - PortType: What operations (functions) will be supported
  - Binding: How will the messages be transmitted on the wire? What SOAP-specific details are there?
  - Service: Where is the service located?



# Hello World: My First WSDL

<Definitions>

```
<?xml version="1.0" encoding="UTF-8"?>  
<definitions name="HelloService"  
targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"  
xmlns="http://schemas.xmlsoap.org/wsdl/"  
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```



# Hello World: My First WSDL

<Types>

This sample uses default SOAP types, so there's no specific section of the document to show them.



# Hello World: My First WSDL

<Message>

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

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



# Hello World: My First WSDL

```
<PortType>
```

```
<portType name="Hello_PortType">  
  <operation name="sayHello">  
    <input message="tns:SayHelloRequest"/>  
    <output message="tns:SayHelloResponse"/>  
  </operation>  
</portType>
```



# Hello World: My First WSDL

<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>
```



# Hello World: My First WSDL

<Binding> {continued}

<output>

<soap:body

encodingStyle=<http://schemas.xmlsoap.org/soap/encoding/>

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

</output>

</operation>

</binding>



# Hello World: My First WSDL

```
<Service>
```

```
  <service name="Hello_Service">
```

```
    <documentation>
```

```
      WSDL File for HelloService
```

```
    </documentation>
```

```
    <port binding="tns:Hello_Binding"
```

```
      name="Hello_Port">
```

```
      <soap:address
```

```
location="http://localhost:8080/soap/servlet/rpcrouter"/>
```

```
    </port>
```

```
  </service>
```

```
</definitions>
```



# Hello World: Summary

- <Definitions>: The HelloService
- <messages>:
  - sayHelloRequest (first name parm)
  - sayHelloResponse (returned message)
- <porttype>: sayHello operation that consists of a request/response service.
- <binding>: instructions to use the SOAP/HTTP transport protocol
- <service>: Service is available at {url}



# WSDL Sample: eBay

```
<?xml version="1.0"?>
<definitions name="eBayWatcherService"
  targetNamespace="http://www.xmethods.net/sd/eBayWatcherService.wsdl"
  xmlns:tns="http://www.xmethods.net/sd/eBayWatcherService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="getCurrentPriceRequest">
    <part name="auction_id" type="xsd:string"/>
  </message>
  <message name="getCurrentPriceResponse">
    <part name="return" type="xsd:float"/>
  </message>
```



# WSDL Sample: eBay (cont.)

```
<portType name="eBayWatcherPortType">
  <operation name="getCurrentPrice">
    <input message="tns:getCurrentPriceRequest" name="getCurrentPrice"/>
    <output message="tns:getCurrentPriceResponse" name="getCurrentPriceResponse"/>
  </operation>
</portType>
<binding name="eBayWatcherBinding" type="tns:eBayWatcherPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getCurrentPrice">
    <soap:operation soapAction=""/>
    <input name="getCurrentPrice">
      <soap:body use="encoded" namespace="urn:xmethods-EbayWatcher"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output name="getCurrentPriceResponse">
      <soap:body use="encoded" namespace="urn:xmethods-EbayWatcher"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
</binding>
```



# WSDL Sample: eBay (cont.)

```
<service name="eBayWatcherService">
  <documentation>
    Checks current high bid for an eBay auction
  </documentation>
  <port name="eBayWatcherPort" binding="tns:eBayWatcherBinding">
    <soap:address location="http://services.xmethods.net:80/soap/servlet/rpcrouter"/>
  </port>
</service>
</definitions>
```



# WSDL Sample: eBay

- **Service:** `ebayWatcherService`
  - Style is `rpc / encoded`
- **Operation:** `getCurrentPrice`
  - **Messages:**
    - `getCurrentPriceRequest`
      - » (sends a string)
    - `getCurrentPriceResponse`
      - » (responds with a float)
- **Binding:** `SOAP/HTTP transport`
- **Address:** `http://services.xmethods.net:80/soap/servlet/rpcrouter`



# But Why Leave Things Simple When You Can Make Them Complicated?

- XML Schema provides the WSDL default data type specification.
- XML Schema most widely used, but types are infinitely complex.
- Messages can be composed of “parts” instead of raw “types”.
- External schema can be imported.
- Web Services themselves come in multiple “flavors”, or dialects, which must be described accurately by WSDL.



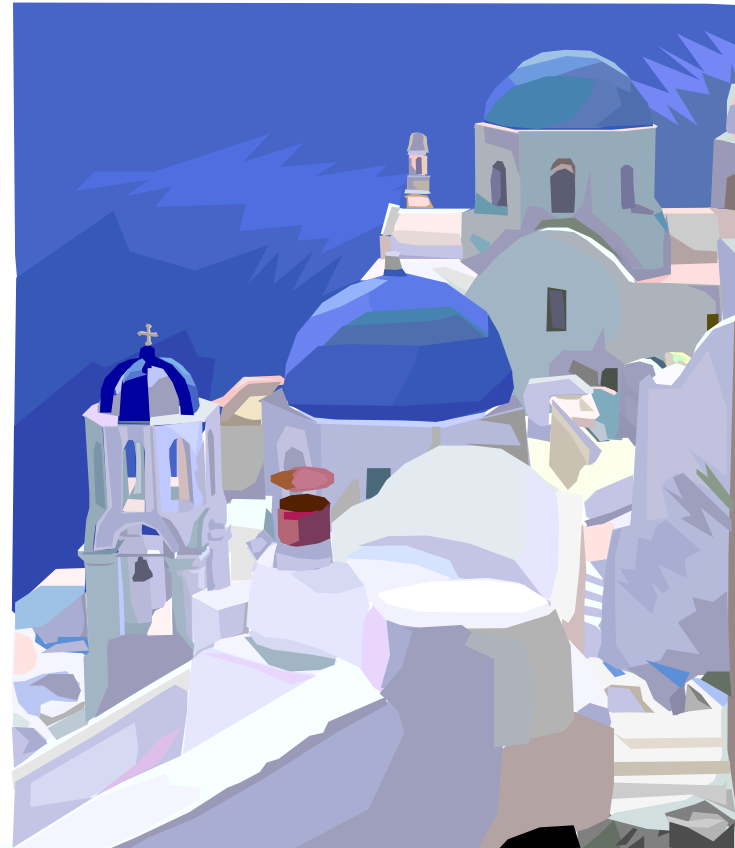
# WSDL and Web Service Dialects

- Binding:
  - RPC or Document.
  - Dictates how to translate a WSDL binding to a SOAP message.
  - Does not imply specific programming model
- Use:
  - Encoded or Literal
  - The terms *encoded* and *literal* are only meaningful for the WSDL-to-SOAP mapping.



# WSDL and Web Service Dialects

- RPC / Encoded
- RPC / Literal
- Document / Encoded
  - (unsupported)
- Document / Literal
- Document / Literal -  
Wrapped
  - (unofficial, but ubiquitous)



# RPC / Encoded (WSDL)

```
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
<binding .../>
<!-- I won't bother with the details, just assume it's RPC/encoded. -->
```



# RPC / Encoded (Message)

```
<soap:envelope>  
  <soap:body>  
    <myMethod>  
      <x xsi:type="xsd:int">5</x>  
      <y xsi:type="xsd:float">5.0</y>  
    </myMethod>  
  </soap:body>  
</soap:envelope>
```

**Strengths:** The WSDL is very straightforward.  
Operation name appears in message so receiver has easy time dispatching message to application implementation of operation.

**Weaknesses:** Type encoding is inside message, so extra info transferred in each message.  
Cannot easily validate message because only <x> and <y> are described by schema.  
Not standards compliant (WS-I).



# RPC / Literal (WSDL)

```
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
<binding .../>
<!-- I won't bother with the details, just assume it's RPC/literal, which is the only
      difference between this and RPC / Encoded -->
```



# RPC / Literal (Messaging)

```
<soap:envelope>  
  <soap:body>  
    <myMethod>  
      <x>5</x>  
      <y>5.0</y>  
    </myMethod>  
  </soap:body>  
</soap:envelope>
```

Strengths: The WSDL is still about as straightforward as it is possible for WSDL to be.  
The operation name still appears in the message.  
The type encoding info is eliminated.  
RPC/literal is WS-I compliant.

Weaknesses: You still cannot easily validate this message since only the `<x ...>5</x>` and `<y ...>5.0</y>` lines contain things defined in a schema; the rest of the `soap:body` contents comes from WSDL definitions.



# Document / Encoded

- Not WS-I compliant: Nobody uses it.



# Document / Literal (WSDL)

```
<types>
  <schema>
    <element name="xElement" type="xsd:int"/>
    <element name="yElement" type="xsd:float"/>
  </schema>
</types>
<message name="myMethodRequest">
  <part name="x" element="xElement"/>
  <part name="y" element="yElement"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>
<binding .../> <!-- I won't bother with the details, just assume it's document/literal. -->
```



# Document / Literal (Messaging)

```
<soap:envelope>  
  <soap:body>  
    <xElement>5</xElement>  
    <yElement>5.0</yElement>  
  </soap:body>  
</soap:envelope>
```

Strengths: There is no type encoding info  
You can finally validate this message with an XML validator.  
Everything within the soap:body is defined in a schema.  
Document/literal is WS-I compliant, but with restrictions (see [below](#)).

Weaknesses: The WSDL is getting a bit more complicated. This is a very minor weakness, however, since WSDL is not meant to be read by humans.  
The operation name in the SOAP message is lost. Without the name, dispatching can be difficult, and sometimes impossible.  
WS-I only allows one child of the soap:body in a SOAP message. As you can see, **this example's soap:body has two children.**



# Document / Literal (Messaging)

The document/literal style seems to have merely rearranged the strengths and weaknesses from the RPC/literal model. You can validate the message, but you lose the operation name.

Is there anything you can do to improve upon this?

Yes. It's called the document/literal-wrapped pattern.



## Document / Literal - Wrapped (WSDL)

```
<types>
  <schema>
    <element name="myMethod">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
    <element name="myMethodResponse">
      <complexType/>
    </element>
  </schema>
</types>
```



## Document / Literal - Wrapped (WSDL cont.)

```
<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
  <part name="parameters" element="myMethodResponse"/>
</message>
<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
    <output message="empty"/>
  </operation>
</portType>

<binding .../> <!-- I won't bother with the details, just assume it's document/literal. -->
```



# Document / Literal - Wrapped (Messaging)

```
<soap:envelope>  
  <soap:body>  
    <myMethod>  
      <x>5</x>  
      <y>5.0</y>  
    </myMethod>  
  </soap:body>  
</soap:envelope>
```

Notice that this SOAP message looks remarkably like the RPC/literal SOAP message in. You might say it looks exactly like the RPC/literal SOAP message, but there's a subtle difference. In the RPC/literal SOAP message, the `<myMethod>` child of `<soap:body>` was the name of the operation. In the document/literal wrapped SOAP message, the `<myMethod>` clause is the name of the wrapper element which the single input message's part refers to. It just so happens that one of the characteristics of the wrapped pattern is that the name of the input element is the same as the name of the operation. This pattern is a sly way of putting the operation name back into the SOAP message.



## Document / Literal - Wrapped (Messaging)

- The input message has a single part.
- The part is an element.
- The element has the same name as the operation.
- The element's complex type has no attributes.



## Document / Literal - Wrapped (Messaging)

### Strengths:

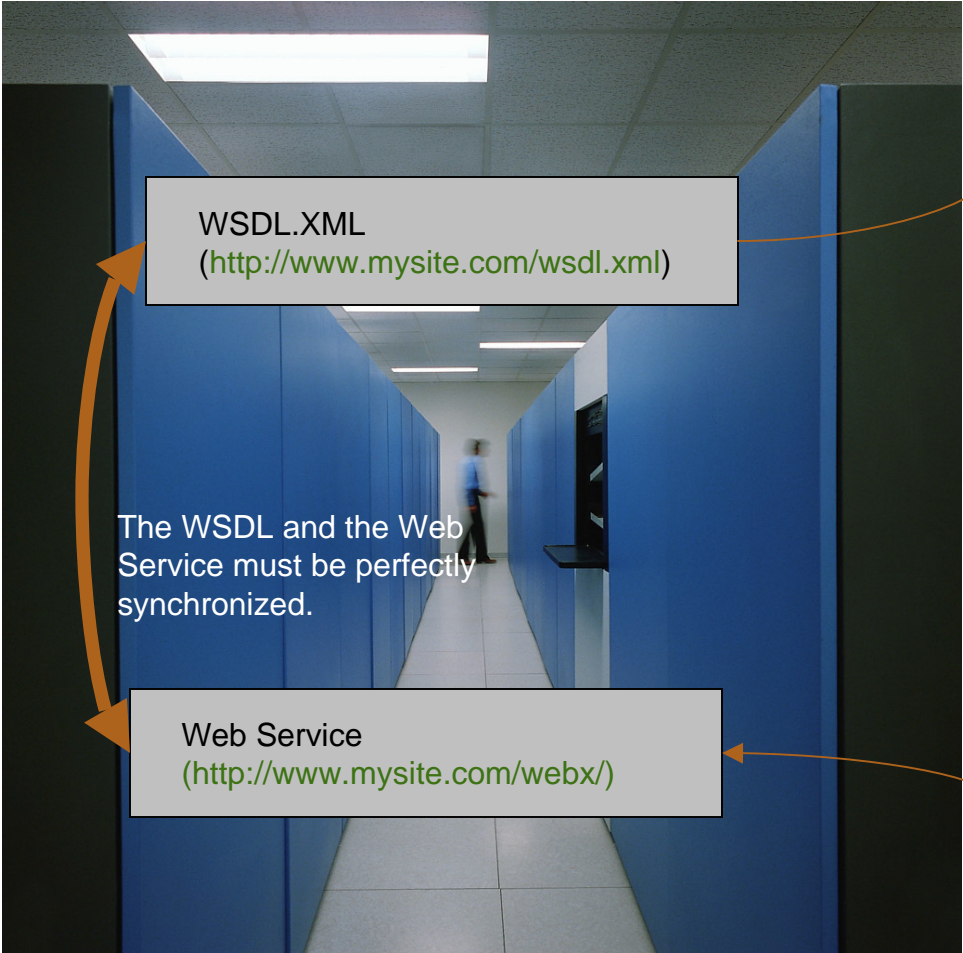
- There is no type encoding info in the message.
- Everything that appears in the `soap:body` is defined by the schema, so you can easily validate this message.
- Once again, you have the method name in the SOAP message.
- Document/literal is WS-I compliant, *and* the wrapped pattern meets the WS SL restriction that the SOAP message's `soap:body` has only one child.

### Weaknesses:

- The WSDL is even more complicated (but who cares?)



# The Missing Piece



The WSDL is retrieved by a code generator or a browser from it's location on the web

Glue, SOAP:Lite, .Net,...

The WSDL Operations,message, data types and other specification are used to generate client classes and methods.



.Net, Java, PHP, etc.

The client sends and receives messages from the web service based on the WSDL contract.

