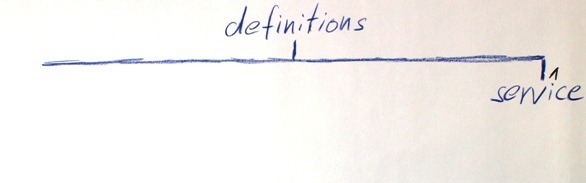
**WSDL Reading, a Beginner's Guide**

**By:** Thomas Bayer  
**Date:** 08/24/2009

*This article explains how to read a* [*WSDL document*](http://www.thomas-bayer.com/axis2/services/BLZService?wsdl) *by analyzing the Web Services description of a* [*public sample Service*](http://thomas-bayer.com/soap/blz-web-service.htm)*. During the article a tree diagram is developed from the content of the WSDL document. The tree illustrates the structure of WSDL. The reader will get an understanding of the WSDL elements and their relationships.*

The root element of a WSDL document is *definitions*. So we start the WSDL tree with a *definitions* node as root. See figure 1: 

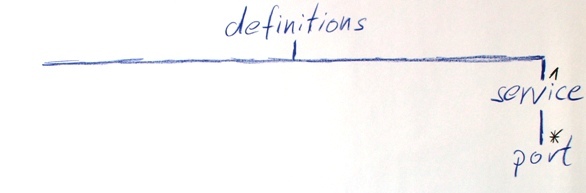
**Figure 1:** Definition and Service

To analyse a WSDL document it is recommended to read it from the buttom upwards. At the bottom of the BLZService's WSDL we find a child element of *definitions* named *service*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <wsdl:service name="BLZService">    <wsdl:port name="BLZServiceSOAP11port\_http" binding="tns:BLZServiceSOAP11Binding">      <soap:address location="<http://www.thomas-bayer.com:80/axis2/services/BLZService>"/>    </wsdl:port>    <wsdl:port name="BLZServiceSOAP12port\_http" binding="tns:BLZServiceSOAP12Binding">      <soap12:address location="<http://www.thomas-bayer.com:80/axis2/services/BLZService>"/>    </wsdl:port>    <wsdl:port name="BLZServiceHttpport" binding="tns:BLZServiceHttpBinding">      <http:address location="<http://www.thomas-bayer.com:80/axis2/services/BLZService>"/>    </wsdl:port>  </wsdl:service> |

**Listing 1:** Service Element

The name of the service is *BLZService*. A service can have multiple ports marked in figure 2 with a \* character. Each port describes a way to access the service. In our *BLZService* example there are three ports. One for SOAP 1.1, one for SOAP 1.2 and one for the HTTP binding.



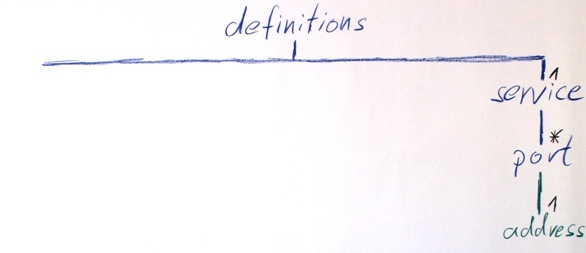
**Figure 2:** Ports of a Service

Let's have a look at the first port in listing 2.

|  |  |
| --- | --- |
| 1  2  3 | <wsdl:port name="BLZServiceSOAP11port\_http" binding="tns:BLZServiceSOAP11Binding">    <soap:address location="<http://www.thomas-bayer.com:80/axis2/services/BLZService>"/>  </wsdl:port> |

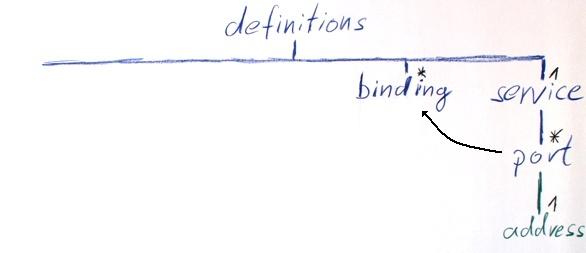
**Listing 2:** Port describing a SOAP 1.1 Endpoint

It's child element *address* has a different XML prefix than the other elements. The prefix *soap* is bound to the SOAP 1.1 binding in this document. Instead of the SOAP binding other bindings for JMS or a file transport can be used. The *address* element has one attribute named *location* pointing to an endpoint address of the service.



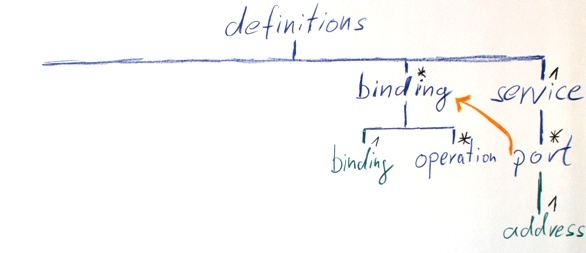
**Figure 3:** Endpoint Address

To move on, we have to look at the *binding* attribute of the port.  
The value "tns:BLZServiceSOAP11Binding" points to a binding further up in the document. Each port is pointing to a different binding in this example. As a consequence the *BLZService* WSDL has three bindings.



**Figure 4:** A Port references a Binding

A binding provides details about a specific transport. The binding in figure 5 has two different types of children.



**Figure 5:** SOAP Binding

First we have a look at the *soap:binding* element in listing 3. The value of the *transport* attribute is an URI that indicates that SOAP messages should be send over HTTP. The value "document" of the *style* attribute gives us a clue about the message style together with the *use* attribute of the *soap:body* elements. In our example we have a *Document/Literal* message style.  
A binding can specify different transport options for each method of a service.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <wsdl:binding name="BLZServiceSOAP11Binding" type="tns:BLZServicePortType">    <soap:binding transport="<http://schemas.xmlsoap.org/soap/http>" style="document"/>    <wsdl:operation name="getBank">      <soap:operation soapAction="" style="document"/>      <wsdl:input>        <soap:body use="literal"/>      </wsdl:input>      <wsdl:output>        <soap:body use="literal"/>      </wsdl:output>    </wsdl:operation>  </wsdl:binding> |

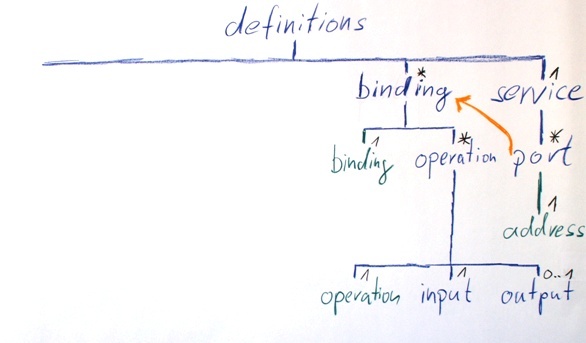
**Listing 3:** SOAP 1.1 Binding

Let's have a look at listing 4. There you can find transport options for the *getBank* operation.  
Inside the *wsdl:operation* element there is a *soap:operation* element at line 2 defining details for the SOAP protocol and its transport. The *soapAction* is a reminiscent from the past. The Basic Profile of the Web Services Interoperability Organization stipulates that the *soapAction* should be used with a fixed value of an empty string.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <wsdl:operation name="getBank">      <soap:operation soapAction="" style="document"/>      <wsdl:input>        <soap:body use="literal"/>      </wsdl:input>      <wsdl:output>        <soap:body use="literal"/>      </wsdl:output>    </wsdl:operation> |

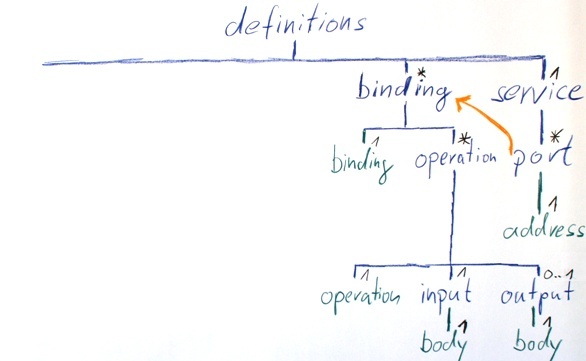
**Listing 4:** Binding Operation

Because Web Services set the focus on messages not parameters, information about the transport of these messages can be found in the *wsdl:input* and *wsdl:output* element. A service may specify one or several faults as an alternative for the output.



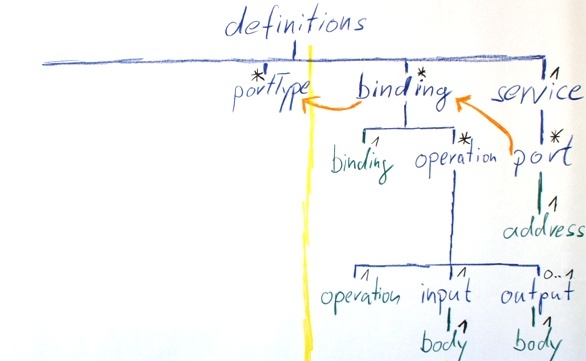
**Figure 6:** Binding Operation

The *soap:body* and *soap:header* elements can describe a message further. In the example the style is always *literal*.



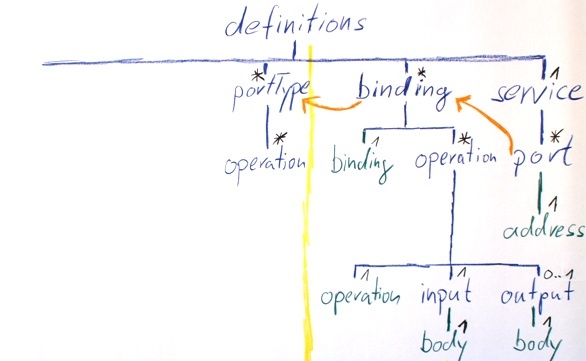
**Figure 7:** soap:body Elements

It is time again to move up in the WSDL. Now we follow the value of the *type* attribute of the binding. It points to a *portType* with the same name further up in the document.



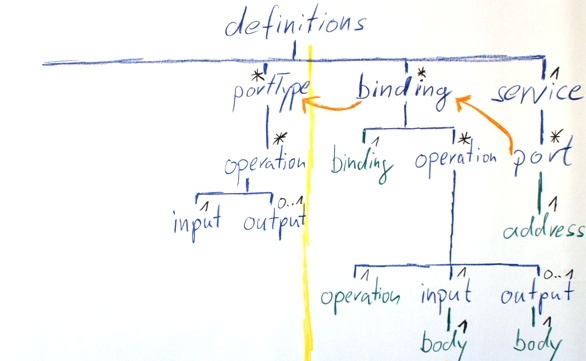
**Figure 8:** Binding referencing a *PortType*

Now we have crossed the border from the concrete details about the transport and location of a service to its pure abstract description of its interface. PortType is in WSDL 1.1 similar to the interface of the Web Service. In WSDL 2.0 the term *portType* is substituted with the term *interface*.  
An interface can have several operations. An operation corresponds to a function in procedural programming.  
The WSDL of the BLZService has only one portType. All of the three bindings refer to the one portType named *BLZServicePortType*.



**Figure 9:** Operations of a *portType*

Inside a *portType* we find operation elements as in the binding. But this time the *input* and *output* describe the structure of the messages not transport specific options.

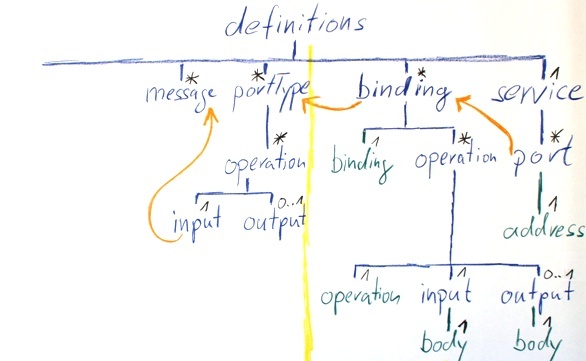


**Figure 10:** Input and Output of an Operation

The *message* attribute of the input refers again up in the WSDL document. It refers to a message named *tns:getBank*. Further up in the document we find a corresponding message with this name.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <wsdl:portType name="BLZServicePortType">    <wsdl:operation name="getBank">      <wsdl:input message="tns:getBank"/>      <wsdl:output message="tns:getBankResponse" wsaw:Action="<http://thomas-bayer.com/blz/BLZService/getBankResponse>"/>    </wsdl:operation>  </wsdl:portType> |

**Listing 5:** The Interface Description of the Service



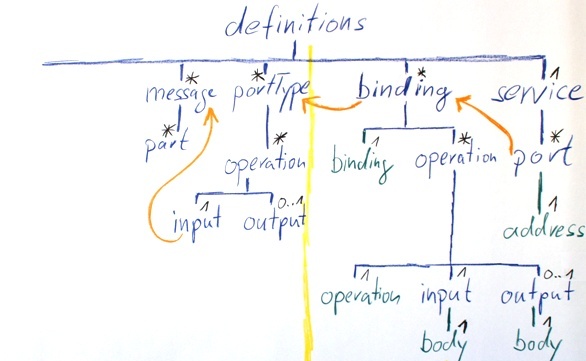
**Figure 11:** Message

The message *getBank* has one *part* element as child. A WSDL specialist will recognize the value of the attribute *name*, "parameters" indicates the wrapper substyle of the *document/literal* style.

|  |  |
| --- | --- |
| 1  2  3 | <wsdl:message name="getBank">    <wsdl:part name="parameters" element="tns:getBank"/>  </wsdl:message> |

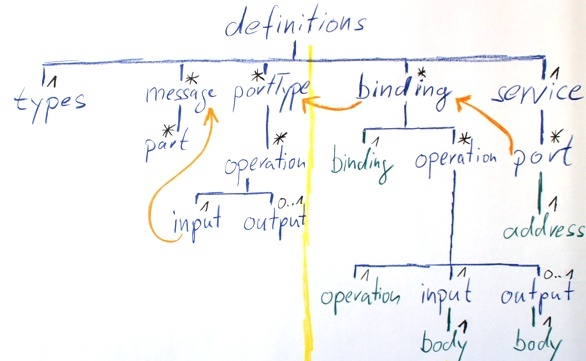
**Listing 6:** getBank Message

The attribute *element* at line 2 points again further up. It refers to an element named *tns:getBank*. We will find this element in a XML Schema.



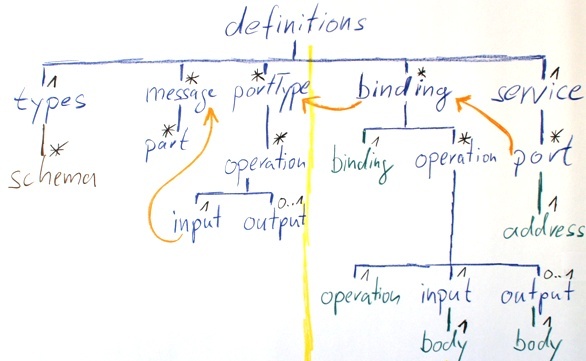
**Figure 12:** A Message and its Parts

The next child of the *definitions* element is *types*.



**Figure 13:** The Types Section

The *types* element can have multiple XML schemas as children.



**Figure 14:** Schemas used in a WSDL

Listing 7 shows the *types* element and an embedded schema.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <wsdl:types>    <xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="<http://thomas-bayer.com/blz/>">      <xsd:element name="getBank" type="tns:getBankType"/>      <xsd:element name="getBankResponse" type="tns:getBankResponseType"/>      <xsd:complexType name="getBankType">        <xsd:sequence>          <xsd:element name="blz" type="xsd:string"/>        </xsd:sequence>      </xsd:complexType>      <xsd:complexType name="getBankResponseType">        <xsd:sequence>          <xsd:element name="details" type="tns:detailsType"/>        </xsd:sequence>      </xsd:complexType>      <xsd:complexType name="detailsType">        <xsd:sequence>          <xsd:element minOccurs="0" name="bezeichnung" type="xsd:string"/>          <xsd:element minOccurs="0" name="bic" type="xsd:string"/>          <xsd:element minOccurs="0" name="ort" type="xsd:string"/>          <xsd:element minOccurs="0" name="plz" type="xsd:string"/>        </xsd:sequence>      </xsd:complexType>    </xsd:schema>  </wsdl:types> |

**Listing 7:** Embedded XML Schema

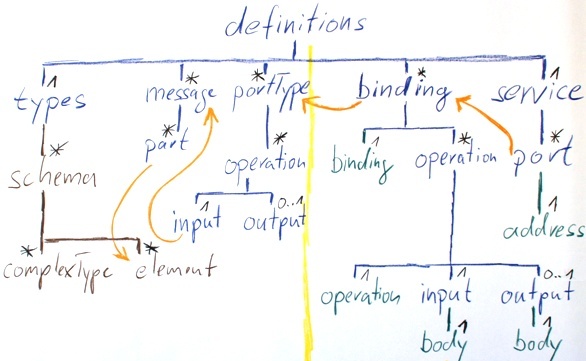
In a schema we can find the definition of:

* complexTypes
* simpleTypes

and the decleration of:

* elements

The XML Schema inside *BLZService* is a typical schema used for Web Services that has only *complexTypes* and elements as toplevel schema components.  
In *document/literal* style all the parts point to elements.



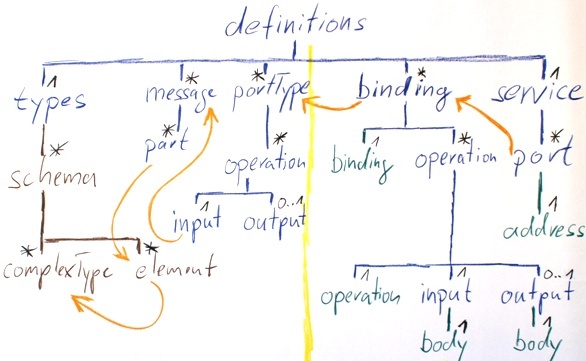
**Figure 15:** A Part can reference an Element

Listing 8 shows the declaration of the *getBank* element.

|  |  |
| --- | --- |
| 1 | <xsd:element name="getBank" type="tns:getBankType"/> |

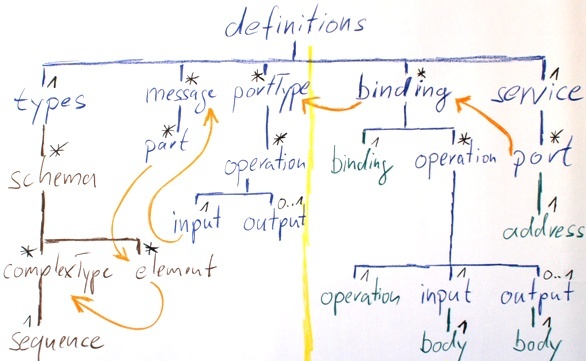
**Listing 8:** Declaration of the Element *getBank*

The type of this element is a complexType named *getBankType* definded somewhere else in the schema.



**Figure 16:** An Element references its Type

The *getBankType* has a sequence as modulgroup containing one element named *blz* of the build-in schema type *string*.



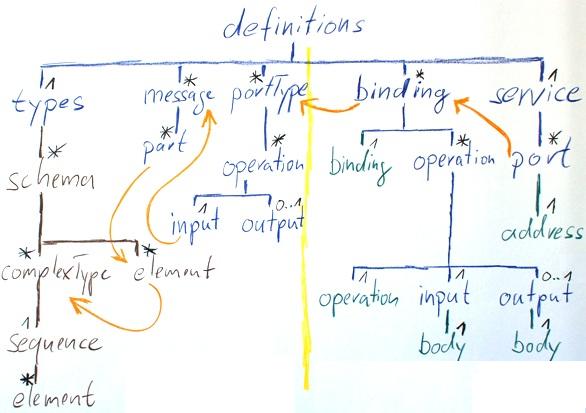
**Figure 17:** A ComplexType with a Sequence as Content

Listing 9 shows the definition of the *getBankType*.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <xsd:complexType name="getBankType">        <xsd:sequence>          <xsd:element name="blz" type="xsd:string"/>        </xsd:sequence>      </xsd:complexType> |

**Listing 9:** The ComplexType *getBankType*

A *sequence* can consist of several elements that describe the order of elements in a SOAP message.



**Figure 18:** The entire Structure of a WSDL Document

Finally we are through the entire WSDL description of the sample service.

All WSDL documents have the same structure as the *BLZService*. To understand a WSDL start reading at the bottom and work your way up by following the right attributes as shown in this article.

I hope this article was helpful to learn how to read a WSDL document.  
  
Thomas Bayer

### WSDL SOAP bindings confusion - RPC vs document

**What is the difference between RPC and document styles in SOAP web services?**

I was asked this question at a job interview and was embarrassed to discover that I didn't know the answer (considering that I listed "SOAP web services" on my resume). I've heard the terms before, but couldn't remember what they meant. The main difference lies in what the body of the SOAP message looks like.  
  
**RPC vs document styles**  
  
The body of an **RPC** (remote procedure call) style SOAP message is constructed in a specific way, which is defined in the SOAP standard. It is built around the assumption that you want to call the web service just like you would call a normal function or method that is part of your application code. The message body contains an XML element for each "parameter" of the method. These parameter elements are wrapped in an XML element which contains the name of the method that is being called. The response returns a single value (encoded in XML), just like a programmatic method. The WSDL code for a RPC-style web service is less complex than that of a document-style web service, but this isn't a big deal since WSDLs aren't meant to be handled by humans.  
  
*A RPC-style request:*

<soap:envelope>

<soap:body>

<multiply> <!-- web method name -->

<a>2.0</a> <!-- first parameter -->

<b>7</b> <!-- second parameter -->

</multiply>

</soap:body>

</soap:envelope>

A **document** style web service, on the other hand, contains no restrictions for how the SOAP body must be constructed. It allows you to include whatever XML data you want and also to include a schema for this XML. This means that the client's and server's application code must do the marshalling and unmarshalling work. This contrasts with RPC in which the marshalling/unmarshalling process is part of the standard, so presumably should be handled by whatever SOAP library you are using. The WSDL code for a document-style web service is much more complex than that of a RPC-style web service, but this isn't a big deal since WSDLs aren't meant to be handled by humans.  
  
*A document-style request:*

<soap:envelope>

<soap:body>

<!-- arbitrary XML -->

<movies xmlns="http://www.myfavoritemovies.com">

<movie>

<title>2001: A Space Odyssey</title>

<released>1968</released>

</movie>

<movie>

<title>Donnie Darko</title>

<released>2001</released>

</movie>

</movies>

</soap:body>

</soap:envelope>

The main downside of the RPC style is that it is *tightly coupled* to the application code (that is, if you decide you want to call these web methods like normal methods--this is not a requirement, but this is what the RPC style was designed for). This means that if you want to change the order of the parmeters or change the types of those parameters, this change will affect the definition of the web service itself (just as it would affect the definition of a normal function or method).  
  
Document style services do not have this issue because they are *loosely couple*d with the application code--the application must handle the marshalling and unmarshalling of the XML data separately. For example, with a document style service, it doesn't matter if the programmer decides to use a "float" instead of an "int" to represent a particular parameter because it's all converted to XML text in the end.  
  
The main downside of the document style is that there is no standard way of determining which method of the web service the request is for. It's easy to get around this limitation, but, however it's done, it must be done manually by the application code. (Note: The "document/literal wrapped" style removes this limitation; read on for more details.)  
  
Another point to note about the document style is that there are no rules for how the SOAP body must be formatted. This can either be seen as a downside or a strength, depending on your perspective. It's a strength if you are looking for the freedom to handle the message the way you want, but a downside if you don't want to have to do the extra marshalling/unmarshalling work that it requires.  
  
**Encoded vs literal encodings**  
  
In addition to the RPC and document styles, there are two types of encodings: "encoded" and "literal".  
  
**Literal** means that the SOAP body follows an XML schema, which is included in the web service's WSDL document. As long as the client has access to the WSDL, it knows exactly how each message is formatted.  
  
**Encoded**, on the other hand, means that the SOAP body does not follow a schema, but still follows a specific format which the client is expected to already know. It is not endorsed by the WS-I standard because there can be slight differences in the way in which various programming languages and web service frameworks interpret these formatting rules, leading to incompatabilities.  
  
This makes for 4 different style/encoding combinations:

**RPC/encoded** - RPC-style message that formats its body according to the rules defined in the SOAP standard (which are not always exact and can lead to incompatabilities).  
**RPC/literal** - RPC-style message that formats its body according to a schema that reflects the rules defined in the SOAP standard. This schema is included in the WSDL.  
**document/encoded** - Document-style message that does not include a schema (nobody uses this in practice).  
**document/literal** - Document-style message that formats its body according to a schema. This schema is included in the WSDL.

There's also a 5th type. It isn't an official standard but it is used a lot in practice. It came into being to compensate for document/literal's main shortcoming of not having a standard way of specifying the web method name:

**document/literal wrapped** - The same as document/literal, but wraps the contents of the body in an element with the same name as the web service method (just like RPC-style messages). This is what web services implemented in Java use by default.

Is my understanding of all this accurate? Which approach do you think is the best? Let me know in the comments.