[SOAP](http://en.wikipedia.org/wiki/SOAP) uses XML to marshal data that is transported to a software application.

Since SOAP’s introduction, three SOAP encoding styles have become popular and are reliably implemented across software vendors and technology providers:

* **SOAP**[**Remote Procedure Call**](http://en.wikipedia.org/wiki/Remote_procedure_call)**(RPC) encoding*,*** also known as *Section 5 encoding,* which is defined by the SOAP 1.1 specification
* **SOAP Remote Procedure Call Literal encoding (SOAP RPC-literal)***,* which uses RPC methods to make calls but uses an XML do-it-yourself method for marshalling the data
* **SOAP document-style encoding***,* which is also known as *message-style* or *document-literal* encoding.

There are other encoding styles, but software developers have not widely adopted them, mostly because their promoters disagree on a standard. For example, Microsoft is promoting **Direct Internet Message Exchange (**[**DIME**](http://en.wikipedia.org/wiki/Direct_Internet_Message_Encapsulation)**)** to encode binary file data, while the rest of the world is promoting **SOAP with Attachments**. SOAP RPC encoding, RPC-literal, and document-style SOAP encoding have emerged as the encoding styles that a software developer can count on.

**SOAP RPC** is the encoding style that offers you the most simplicity. You make a call to a remote object, passing along any necessary parameters. The SOAP stack serializes the parameters into XML, moves the data to the destination using transports such as [HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) and [SMTP](http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol), receives the response, deserializes the response back into objects, and returns the results to the calling method. Whew! SOAP RPC handles all the encoding and decoding, even for very complex data types, and binds to the remote object automatically.

Now, imagine that you have some data already in XML format. SOAP RPC also allows literal encoding of the XML data as a single field that is serialized and sent to the [Web service](http://en.wikipedia.org/wiki/Web_service) host. This is what’s referred to as *RPC-literal* encoding. Since there is only one parameter — the [XML tree](http://en.wikipedia.org/wiki/XML_tree) — the SOAP stack only needs to serialize one value. The SOAP stack still deals with the transport issues to get the request to the remote object. The stack binds the request to the remote object and handles the response.

Lastly, in a *SOAP document-style* call, the SOAP stack sends an entire [XML document](http://en.wikipedia.org/wiki/XML) to a server without even requiring a return value. The message can contain any sort of XML data that is appropriate to the remote service. In SOAP document-style encoding, the developer handles everything, including determining the transport (e.g., HTTP, MQ, SMTP), marshaling and unmarshaling the body of the SOAP envelope, and [parsing](http://en.wikipedia.org/wiki/Parsing) the XML in the request and response to find the needed data.

The three encoding systems are compared here:

SOAP RPC encoding is easiest for the software developer; however, all that ease comes with a scalability and performance penalty. In SOAP RPC-literal encoding, you are more involved with handling XML parsing, but it requires there to be overhead for the SOAP stack to deal with. SOAP document-literal encoding is most difficult for the software developer, but consequently requires little SOAP overhead.

**Why is SOAP RPC easier?** With this encoding style, you only need to define the public object method in your code once; the SOAP stack unmarshals the request parameters into objects and passes them directly into the method call of your object. Otherwise, you are stuck with the task of parsing through the XML tree to find the data elements you need before you get to make the call to the public method.

There is an argument for parsing the XML data yourself: since you know the data in the XML tree best, your code will parse that data more efficiently than generalized SOAP stack code. You will find this when measuring scalability and performance in SOAP encoding styles.

**References:**

1. Discover SOAP encoding’s impact on Web service performance (<http://www.ibm.com/developerworks/webservices/library/ws-soapenc/>)

There are two communication style models that are used to translate a WSDL binding to a SOAP message body. They are: **Document & RPC**

The **advantage of using a Document style model** is that you can structure the SOAP body any way you want it as long as the content of the SOAP message body is any arbitrary XML instance. The Document style is also referred to as ***Message-Oriented style***.

However, with an **RPC style model**, the structure of the SOAP request body must contain both the operation name and the set of method parameters. The RPC style model assumes a specific structure to the ***XML instance*** contained in the message body.

Furthermore, there are two encoding use models that are used to translate a WSDL binding to a SOAP message. They are: **literal, and encoded**

When using a **literal use model**, the body contents should conform to a user-defined **XML-schema(XSD) structure**. The advantage is two-fold. For one, you can validate the message body with the user-defined XML-schema, moreover, you can also transform the message using a transformation language like XSLT.

With a (SOAP) **encoded use model**, the message has to use XSD datatypes, but the structure of the message need not conform to any user-defined XML schema. This makes it difficult to validate the message body or use XSLT based transformations on the message body.

The combination of the different style and use models give us four different ways to translate a WSDL binding to a SOAP message.

Document/literal

Document/encoded

RPC/literal

RPC/encoded

I would recommend that you read this article entitled [Which style of WSDL should I use?](http://www.ibm.com/developerworks/library/ws-whichwsdl/) by Russell Butek which has a nice discussion of the different style and use models to translate a WSDL binding to a SOAP message, and their relative strengths and weaknesses.

Once the artifacts are received, in both styles of communication, I invoke the method on the port. Now, this does not differ in RPC style and Document style. So what is the difference and where is that difference visible?

The place where you can find the difference is the "RESPONSE"!

**RPC Style:**

package com.sample;

import java.util.ArrayList;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

@WebService

@SOAPBinding(style=Style.RPC)

public interface StockPrice {

public String getStockPrice(String stockName);

public ArrayList getStockPriceList(ArrayList stockNameList);

}

The SOAP message for second operation will have empty output and will look like:

**RPC Style Response:**

<ns2:getStockPriceListResponse

xmlns:ns2="http://sample.com/">

<return/>

</ns2:getStockPriceListResponse>

</S:Body>

</S:Envelope>

**Document Style:**

package com.sample;

import java.util.ArrayList;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

@WebService

@SOAPBinding(style=Style.DOCUMENT)

public interface StockPrice {

public String getStockPrice(String stockName);

public ArrayList getStockPriceList(ArrayList stockNameList);

}

If we run the client for the above SEI, the output is:

123 [123, 456]

This output shows that ArrayList elements are getting exchanged between the web service and client. This change has been done only by the changing the style attribute of SOAPBinding annotation. The SOAP message for the second method with richer data type is shown below for reference:

**Document Style Response:**

<ns2:getStockPriceListResponse

xmlns:ns2="http://sample.com/">

<return xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xs="http://www.w3.org/2001/XMLSchema"

xsi:type="xs:string">123</return>

<return xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xs="http://www.w3.org/2001/XMLSchema"

xsi:type="xs:string">456</return>

</ns2:getStockPriceListResponse>

</S:Body>

</S:Envelope>

**Conclusion**

* As you would have noticed in the two SOAP response messages that it is possible to validate the SOAP response message in case of DOCUMENT style but not in RPC style web services.
* The basic **disadvantage of using RPC style** is that it doesn’t support richer data types and that of using Document style is that it brings some complexity in the form of XSD for defining the richer data types.
* The choice of using one out of these depends upon the operation/method requirements and the expected clients.

Similarly, in what way SOAP over HTTP differ from XML over HTTP? After all SOAP is also XML document with SOAP namespace. So what is the difference here?

Why do we need a standard like SOAP? By exchanging XML documents over HTTP, two programs can exchange rich, structured information without the introduction of an additional standard such as SOAP to explicitly describe a message envelope format and a way to encode structured content.

SOAP provides a standard so that developers do not have to invent a custom XML message format for every service they want to make available. Given the signature of the service method to be invoked, the SOAP specification prescribes an unambiguous XML message format. Any developer familiar with the SOAP specification, working in any programming language, can formulate a correct SOAP XML request for a particular service and understand the response from the service by obtaining the following service details.

* Service name
* Method names implemented by the service
* Method signature of each method
* Address of the service implementation (expressed as a URI)

Using SOAP streamlines the process for exposing an existing software component as a Web service since the method signature of the service identifies the XML document structure used for both the request and the response.

A WSDL document (WSDL = Web Service Description Language) describes a web service. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. This gives four style/use models:

* RPC/encoded.
* RPC/literal.
* Document/encoded (not used in practice).
* Document/literal.

WSDL distinguishes between two message styles: document and RPC. The message style affects the contents of the SOAP Body:

* **Document style:** The SOAP Body contains one or more child elements called parts. There are no SOAP formatting rules for what the body contains; it contains whatever the sender and the receiver agrees upon.
* **RPC style:** RPC implies that SOAP body contains an element with the name of the method or operation being invoked. This element in turn contains an element for each parameter of that method/operation.

For applications that use serialization/deserialization to abstract away the data wire format, there is one more choice to be made: the serialization format. There are two popular serialization formats:

* **SOAP Encoding:** SOAP encoding is a set of serialization. The rules specify how objects, structures, arrays, and object graphs should be serialized. Generally speaking, an application using SOAP encoding is focused on remote procedure calls and will likely use RPC message style. When SOAP encoding is used, the SOAP message contains data type information within the SOAP message. This makes serialization (data translation) easier since the data type of each parameter is denoted with the parameter.
* **Literal:** Data is serialized according to a schema. In practice, this schema is usually expressed using W3C XML Schema. The SOAP message does not directly contain any data type information, just a reference (namespace) to the schema that is used. To perform proper serialization (data translation) both, the sender and the receiver, must know the schema and must use the same rules for translating data.