

Overview and examples

SOAP

Simple Object Access Protocol

By Hamid M. Perazi

1	About this document.....	3
2	What is SOAP?	3
3	SOAP and XML.....	3
3.1	XML messaging.....	3
3.1.1	RPC and EDI.....	3
3.1.2	Several ways to express same data via XML.....	4
3.1.3	SOAP messages	4
3.1.3.1	Example: a document-style SOAP.....	5
3.1.3.2	Envelopes.....	5
3.1.4	RPC Messages	5
3.1.4.1	Example: Simple RPC-style SOAP message.....	6
3.1.5	Encoding Styles	7
3.1.5.1	Example the encodingStyle attribute	7
3.1.6	Versioning.....	8
3.1.6.1	Example: versioning, the Upgrade header	8
3.1.7	SOAP Faults.....	9
3.1.7.1	The fault code	9
3.1.7.2	The fault string.....	9
3.1.7.3	The fault actor.....	9
3.1.7.4	The fault details.....	9
3.1.7.5	Standard SOAP Fault Codes.....	10
3.1.8	Version Mismatch.....	10
3.1.8.1	MustUnderstand.....	10
3.1.9	Server.....	10
3.1.10	Client.....	10
3.1.11	Custom Faults	10
3.1.11.1	Example: Custom Faults.....	10
3.1.12	Message paths and Actors.....	11
3.1.13	Targeting.....	11
3.1.14	WS-Routing	11
3.1.14.1	Example: WS-Routing message	11
3.1.15	Using SOAP for RPC-style Web Services.....	12
3.1.15.1	Invoking methods.....	12
3.1.15.2	Returning responses	13
3.1.15.3	Reporting Errors.....	13
3.1.16	SOAP's Data Encoding.....	13
3.1.16.1	Used terminology.....	14
3.1.16.2	XML Schemas and xsi:type.....	15

1 About this document

Following document is a summary about SOAP. This document contains material and text from different sources¹ and my own work experience.

2 What is SOAP?

SOAP is a very common packaging format, built on XML. SOAP encodes messages and data values. SOAP's place in the web services technology stack is as a standardized packaging protocol for the messages shared by applications. The specifications defines nothing more than a simple XML-based envelope for the information being transferred, and a set of rules for translating application and platform-specific data types into XML representations. SOAP's design makes it suitable for a wide variety of application messaging and integration patterns.

3 SOAP and XML

SOAP is in XML format. SOAP is an application of the XML specification. It relies heavily on XML standards like XML Schema and XML namespaces for its definition and function.

3.1 XML messaging

XML messaging is where applications exchange information using XML documents. It provides a flexible way for applications to communicate, and forms the basis of SOAP. A message can be anything, a product, a listing flight to a certain city or pieces of information that may be relevant for a particular application.

Since XML isn't tied to a particular application, programming language or operative system it can be used in all environments. A window Java program can create a XML document and send it to a UNIX based Perl program. The fundamental idea is that two applications regardless of their environment can send messages to each other. SOAP provides such standard way to structure XML messages.

3.1.1 RPC and EDI

SOAP has two related applications:

- RPC: Remote Procedure Call
- EDI: Electronic Document Interchange

RPC is the basis of distributed computing, the way for one program to make a procedure call on another, passing arguments and receiving return values.

EDI is basis of automated business transactions, defining a standard format and interpretation of financial and commercial documents and messages.

¹ Among others material from O'Reilly is used

3.1.2 Several ways to express same data via XML

There is several ways to express same data via XML. Below is some of these ways:

```
<phoneNumber>(604) 683-1770</phoneNumner>
<phonenumber>
  <areaCode>604</areaCode>
  <exchange>683</exchange>
  <number>1770</number>
</phoneNumber>
<phoneNumber area="604" exchange="683" number="1770"/>
<phone area="604">
  <exchange>683</exchange>
  <number>1770</number>
</phone>
```

We need to define:

- The type of information we are exchanging
- How that information is to be expressed as XML
- How to actually go about sending that information

Without agreed conventions, programs cannot know how to decode the information. SOAP provides these conversions.

3.1.3 SOAP messages

A SOAP message consists of an envelope containing an optional header and a required body.

The header contains blocks of information relevant to how the message is to be processed. This includes routing and delivery settings and etc. Anything that can be expressed in XML syntax can go in the body of a message.

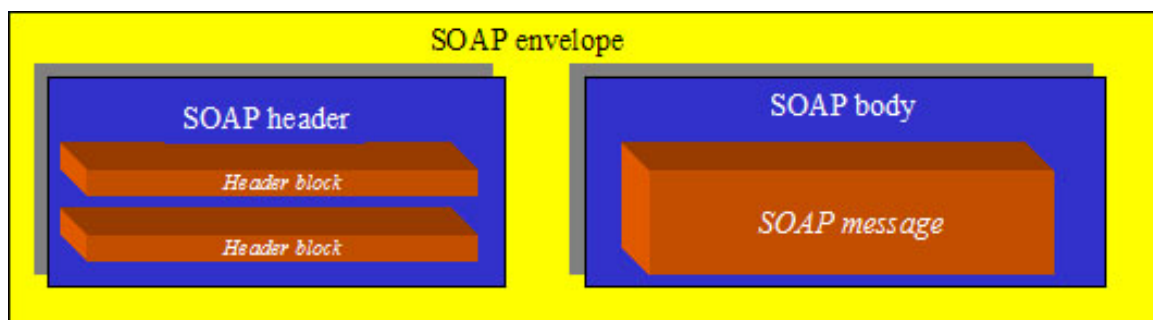


Figure 1 Structure of a SOAP envelope

The XML syntax for expressing a SOAP message is based on the <http://www.w3.org/2001/06/soap-envelope> namespace. This XML message identifier points to an XML Schema that defines the structure of what a SOAP message looks like.

3.1.3.1 Example: a document-style SOAP

```
<s:Envelope
xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Header>
    <m:transaction xmlns:m="soap-transaction" s:mustUnderstand="true">
<transactionID>1234</transactionID>
</m:transaction>
</s:Header>
<s:Body>
  <n:purchaseOrder xmlns:n="urn:OrderSverice">
    <from><person> Oliver Porasl</person>
      <dept> XXX</dept></from>
    <to><person> John Smith</person>
      <dept> YYYY</dept></to>
<order><quantity>1</quantity>
  <item> Table</item></order>
</n:purchaseOreder>
</s:Body>
</s:Envelope>
```

3.1.3.2 Envelopes

Every *Envelope* element must contain exactly one *Body* element. The *Body* element may contain as many child nodes as are required. The contents of the *Body* elements are the message. The *Body* element is defined in such a way that it can contain any valid, well-formed XML that has been namespace qualified and doesn't contain any processing instructions or Document Type Definition (DTD) references.

Each element contained by the *Header* is called a *header block*. The purpose of a header block is to communicate contextual information relevant to the processing of a SOAP message. An example might be a header block a header block that contains authentication credentials or message routing information.

3.1.4 RPC Messages

Typically messages come in pairs

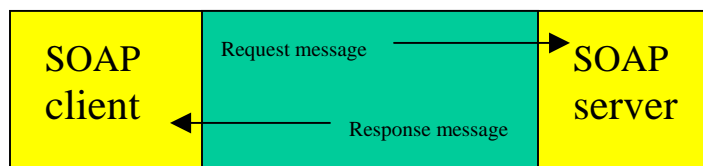


Figure 2 Basic RPC messaging architecture

SOAP doesn't require every request to have a response, or vice versa, but it is common to see the request-response pairing. Imagine the server offers this function which returns a stuck's price, as aSOAP sevice:

```
Public Float getQuote(String symbol);
```

3.1.4.1 Example: Simple RPC-style SOAP message

This example illustrates a simple RPC-style SOAP message that represents a request for IBM's current stock price. Header block indicates a transaction ID of "1234".

```
<s:Envelope
xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Header>
    <m:transacrction xmlns:m="soap-transaction" s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:getQuote xmlns:n="urn:QuoteService">
      <symbol xsi:type="xsd:string">
        IBM
      </symbol>
    </n:getQuote>
  </s:Body>
</s:Envelope>
```

Following example is a possible response to requested stock quote value:

```
<s:Envelope
xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Body>
    <n:getQuoteResponse
      xmlns:n="urn:QuoteService">
      <value xsi:type="xsd:float">
        100.03
      </value>
    </n:getQuoteResponse>
  </s:Body>
</s:Envelope>
```

3.1.4.1.1 The mustUnderstand Attribute

When a SOAP message is sent from one application to another, there is an implicit requirement that must understand how to process that message. If the recipient does not understand the message, the recipient must reject the message must reject the message and explain the problem to the sender.

Header blocks are different. A recipient may or may not understand how to deal with a particular header block but still be able to process the primary message properly. If the sender of the message wants to require that recipient understand a particular block, it may add a *mustUnderstand="true"* attribute to the header block. If this flag is present, and the recipient does not understand the block to which it is attached, the recipient must reject the entire message. This guarantees that the recipient understands transactions.

The SOAP fault structure is not allowed to express any information about which headers were not understood. The *details* element would be the only place to put this information and it is reserved.

```
<s:Envelope xmlns:s="..">
  <s:Header>
    <f:Misunderstood qname="abc:transaction" xmlns:="soap-transactions" />
  </s:Header>
  <s:Body>
    <s:Fault>
      <faultcode>MustUnderstand</faultcode>
      <faultstring>
        Header(s) not understood
      </faultstring>
      <faultactor>http://porasl.com</faultactor>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

3.1.5 Encoding Styles

SOAP standard introduces a concept known as *encoding styles*. An encoding style is a set of rules that define exactly how native application and platform data types are to be encoded into a common XML syntax. These are for use with RPC-style SOAP.

The encoding style for a particular set of XML element is defined through the use of the *encodingStyle* attribute, which can be placed anywhere in the document and applies to all subordinate children of the element on which it is located.

3.1.5.1 Example the encodingStyle attribute

```
<s:Envelope
  xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Body>
    <n:getQuote xmlns:n="urn:QuoteService"
      s:encodingStyle="http://www.w3.org/2001/06/soap-encoding">
      <symbol xsi:type="xsd:string">IBM</symbol>
    </n:getQuote>
  </s:Body>
</s:Envelope>
```

Even though the SOAP specification defines an encoding style. It has been explicitly declared that no single style is the default serialization scheme. Encoding styles are how applications on different platforms share information, even though they may not have common data types or representations.

3.1.6 Versioning

There have been several versions of the SOAP specification. The most recent working draft, SOAP version 1.2, represents the first fruits of the World Wide Web Consortium's (W3C) effort to standardize an XML-based packaging protocol for web service. The W3C chose SOAP as the basis for that effort.

The version of a SOAP message can be determined by checking the namespace defined for the SOAP envelope. Version 1.1 uses the namespace <http://schemas.xmlsoap.org/soap/envelope> whereas version 1.3 uses the namespace <http://www.w3.org/2001/06/soap-envelope>

Exampel, Distinguished between SOAP 1.1 and SOAP 1.2

```
<!--Version 1.1. SOAP Envelope -->
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope">
....
</s:Envelope>

<!-- Version 1.2 SOAP Envelope -->
<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope">
...
</s:Envelope>
```

When applications report a version mismatch error back to the sender of the message, it may optionally include an *Upgrade* header block that tells the sender which version of SOAP it supports.

3.1.6.1 Example: versioning, the Upgrade header

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope">
  <s:header>
    <V:Upgrade xmlns:V="http://www.w3.org/2001/06/soap-upgrade">
      <envelope gname="ns1:Envelope" xmlns:ns1="http://www.w3.org/2001/06/soap-
envelope" />
    </V:Upgrade>
  </s:Header>
  <s:Body>
    <s:Fault>
      <faultcode>s:VersionMismatch</faultcode>
      <faultstring>Version Mismatch</faultstring>
    </s:Body>
  </s:Envelope>
```

3.1.7 SOAP Faults

A SOAP fault is a special type of message specifically targeted at communicating information about errors that may have occurred during the processing of a SOAP message.

```
<s:Envelope xmlns:s="....">
  <s:Body>
    <s:Fault>
      <faultcode> Client.Authentication</faultcode>
      <faultstring>
        Invalid credentials
      </faultstring>
      <faultactor>http://porasl.com</faultactor>
      <details>
        <!-- application specific details -->
      </details>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

SOAP fault demonstrates how this extensibility works. The `Client.Authentication` fault code is a more granular derivative of the *Client* fault type. The “.” Notation indicates that the piece to the left of the period is more generic than the piece that is to the right of the period.

The information communicated in the SOAP fault is as follows:

3.1.7.1 The fault code

An algorithmically generated value for identifying the type of error that occurred. The value must be an XML Qualified Name meaning that the name of the code only has meaning within a defined XML namespace.

3.1.7.2 The fault string

A human-readable explanation of the error.

3.1.7.3 The fault actor

The unique identifier of the message processing node at which the error occurred.

3.1.7.4 The fault details

Used to express application-specific details about the error that occurred. This must be present if the error that occurred is directly related to some problem with the body of the

message. It must not be used, to express information about errors that occur in relation to any aspect of the message process.

3.1.7.5 Standard SOAP Fault Codes

SOAP defines four standard types of faults that belong to the <http://www.w3.org/2001/06/soap-envelope> namespace.

3.1.8 Version Mismatch

The SOAP envelope is using an invalid namespace for the SOAP Envelope element.

3.1.8.1 MustUnderstand

A Header block contained a *mustUnderstand*=”true” flag that was not understood by the message recipient.

3.1.9 Server

An error occurred that can’t be directly linked to the processing of the message

3.1.10 Client

There is a problem in the message. For example, the message contains invalid authentication credentials, or there is an improper application.

3.1.11 Custom Faults

A web service may define its own custom fault codes that do not derive from the ones defined by SOAP. The only requirement is that these custom faults be namespace qualified.

3.1.11.1 Example: Custom Faults

```
<s:Envelope xmlns:s=”...”>
  <s:Body>
    <s:Fault xmlns:xyz=”urn:myCustomFaults”>
      <faultcode>abc:CustomFault</faultcode>
      <faultstring>
        My Custom fault can be written here!
      </faultstring>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

Custom faults can still be useful in situations where the standard fault codes are too generic or are otherwise inadequate for the expression of what error occurred.

3.1.12 Message paths and Actors

A SOAP intermediary is a web service specially designed to sit between a service consumer and a service provider and add value or functionality to the transaction between the two. The set of intermediaries that the message travels through is called the message path. Every intermediary along that path is known as an actor.

3.1.13 Targeting

The construction of a message path (the definition of which nodes a message passes through) is not covered by the SOAP specification. What SOAP does specify is a mechanism of identifying which parts of the SOAP message are intended for processing by specific actors in its message path. This mechanism is known as “targeting” and can only be used in relation to header blocks (the body of the SOAP envelope cannot be explicitly targeted at a particular node).

3.1.14 WS-Routing

WS-Routing defines a standard SOAP header block for expressing routing information. Its role is to define the exact sequence of intermediaries through which a message is to pass.

3.1.14.1 Example: WS-Routing message

```
<s:Envelope xmlns:s="...">
  <s:header>
    <m:path xmlns:m="http://schemas.xmlsoap.org/rp" s:mustUnderstand="true">
      <m:action>http://www.abc.com/test</m:action>
      <m:to>http://www.telecom.com/someNodes/target</m:to>
      <m:fwd>
        <m:via>http://host1.com</m:via>
        <m:via>http://host2.com</m:via>
      </m:fwd>
      <m:rev>
        <m:via/>
      </m:rev>
      <m:from>mailto:info@porasl.com</m:from>
      <m:id>
        uuid:84rewrew-rwe-3r3dwedw-r4wd-we3dd
      </m:id>
    </m:path>
  </s:Header>
</s:body>
```

```
....  
</s:body>  
</s:Envelope>
```

In this example, we see the SOAP message is intended to be delivered to a recipient located at <http://www.telecom.com/someNodes/target/> but that it must first go through both the <http://host1.com> and <http://host2.com> intermediaries.

To ensure that the message path defined by the WS-Routing header block is properly followed, and because WS-Routing is a third-party extension to SOAP that not every SOAP processor will understand, the *mustUnderstand="true"* flag can be set on the *path* header block.

3.1.15 Using SOAP for RPC-style Web Services

RPC is the most common application of SOAP at the moment. The following sections show how method calls and return values are encoded in SOAP message bodies.

3.1.15.1 Invoking methods

The rules for packaging an RPC request in a SOAP envelope are simple:

- The method call is represented as a single structure with each in-out parameter modeled as a field in that structure.
- The names and physical order of the parameters must correspond to the names and physical order of the parameters in the methods being invoked.

This means that a Java Method with following signature:

```
String checkStatus(String orderCode, String customerID);
```

Can be invoked with these arguments:

```
result=checkStatus("BookName","Store number 2")
```

using the following SOAP envelope:

```
<s:Envelope xmlns:s="...">  
  <s:Body>  
    <checkStatus xmlns="..."  
      s:encodingStyle=http://www.w3.org/2001/06/soap-encoding>  
      <orderCode xsi:type="string">BookName</orderCode>  
      <customerID xsi:type="string">Store number 2</customerID>  
    </checkStatus>  
  </s:body>  
</s:Envelope>
```

3.1.15.2 Returning responses

Method responses are similar to method calls in that the structure of the response is modeled as a single structure with a field for each in-out or parameter in the method signature. If the *checkStatus* method we called earlier returned the string *new*, the SOAP response might be like below:

```
<SOAP:Envelope xmlns:s="...">
  <SOAP:Body>
    <checkStatusResponse
      SOAP:encodingStyle=http://www.w3.org/2001/06/soap-encoding>
      <return xsi:type="xsd:string">new</return>
    </checkStatusResponse>
  </SOAP:body>
</SOAP:Envelope>
```

The name of the message response structure (*CheckStatusResponse*) element isn't important, but the convention is to name it after the method, with *Response* appended. Similarly, the name of the return element is arbitrary- the first field in the message response structure is assumed to be the return value.

3.1.15.3 Reporting Errors

The SOAP RPC conventions make use of the SOAP fault as the standard method of returning error responses to RPC clients. As with standard SOAP messages, the SOAP fault is used to convey the exact nature of the error that has occurred and can be extended to provide additional information through the use of the detail element. There's little point in customizing error messages in SOAP faults when you are doing RPC, as most SOAP RPC implementations will not know how to deal with the custom error information.

3.1.16 SOAP's Data Encoding

The first part of the SOAP specification outlines a standard envelope format for packaging data. The second part of the specification outlines one possible method of serializing the data intended for packaging. These rules outline in specific detail how basic application data types are to be mapped and encoded into XML format when embedded into a SOAP Envelope.

Encoding styles are optional, and in many situations not useful. SOAP envelopes are designed to carry any arbitrary XML document no matter what the body of the message looks like, or whether it conforms to any specific set of data encoding rule. Encoding rules are offered to allow applications to dynamically exchange information without priori knowledge of the types of information to be exchanged.

3.1.16.1 Used terminology

A value represents either a single data unit or combination of data units. This could be a person's name, the score of a football game, or the current temperature. An accessor presents an element that contains or allows access to a value. In the following, bookname is an accessor, and myBookname is a Value:

```
<bookname>myBookname</bookname>
```

A compound value represents a combination of two or more accessors grouped as children of a single accessor, as below:

```
<name>
  <firstname>Hamid</firstnamre>
  <lastname> Porasl</lastname>
</name>
```

There are two types of compound values, structs and arrays. A struct is a compound value in which each accessor has a different name. An array is a compound value in which the accessors have the same name, i.e. Values are identified by their positions in the array. Bellow examples of an struct and an array are shown:

<!-- A struct -- >

```
<person>
  <firstname>Hamid</firstnamre>
  <lastname> Porasl</lastname>
</person>
```

<!-- an array -- >

```
<people>
  <person name='Hamid Porasl' />
  <person name='Andrew Porasl' />
</people>
```

Through the use of the special *id* and *href* attributes, SOAP defines that accessors may either be *single-referenced* or *multi-referenced*. A single-referenced accessor doesn't have an *identity* except as a child of its parent element. In example below the <address> element is a single-referenced accessor.

```
<people>
  <person name='Andrew Porasl'>
    <address>
      <street> 3663 City Edge Place</street>
      <city> Vancouver</city>
      <state>BC</state>
    </address>
  </person>
</people>
```

A multi-referenced accessor uses `id` to give an identity to its value. Other accessors can use the `href` attribute to refer to their values. Each person has the same address because they reference the same multi-referenced `address` accessor.

```
<people>
  <person name='Andrew Porasl'>
    <address href='#address-1'>
  </person>
  <person name='Hamid Porasl'>
    <address href='#address-1'>
  </person>

  <address id='address-1'>
    <street>3663 City Edge Place</street>
    <city> Vancouver</city>
    <state> BC</state>
  </address>
```

This approach can also be used to allow an accessor to reference external information sources that aren't part of the SOAP Envelope, as:

```
<person name='Andrew Porasl'>
  <address href='http://porasl.com/Andrew/data.xml#Andrew_porasl' />
</person>
```

3.1.16.2 XML Schemas and `xsi:type`