Table of Contents

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

Chapter 2. The SOAP Message

All SOAP messages are packaged in an XML document called an *envelope*, which is a structured container that holds one SOAP message. The metaphor is appropriate because you stuff everything you need to perform an operation into an envelope and send it to a recipient, who opens the envelope and reconstructs the original contents so that it can perform the operation you requested. The contents of the SOAP envelope conform to the SOAP specification,[1] allowing the sender and the recipient to exchange messages in a language-neutral way: for example, the sender can be written in Python and the recipient can be written in Java or C#. Neither side cares how the other side is implemented because they agree on how to interpret the envelope. In this chapter we'll get inside the SOAP envelope.

[1] The spec can be found at http://www.w3.org/TR/SOAP. The SOAP 1.1 specification is not a W3C standard, but the SOAP 1.2 spec currently under development will be.

2.1. The HTTP Binding

The SOAP specification requires a SOAP implementation to support the transporting of SOAP XML payloads using HTTP, so it's no coincidence that the existing SOAP implementations all support HTTP. Of course, implementations are free to support any other transports as well, even though the spec doesn't describe them. There's nothing whatsoever about the SOAP payload that prohibits transporting messages over transports like SMTP, FTP, JMS, or even proprietary schemes; in fact, alternative transports are frequently discussed, and a few have been implemented. Nevertheless, since HTTP is the most prevalent SOAP transport to date, that's where we'll concentrate. Once you have a grasp of how SOAP binds to HTTP, you should be able to easily migrate your understanding to other transport mechanisms.

SOAP can certainly be used for request/response message exchanges like RPC, as well as inherently one-way exchanges like SMTP. The majority of Java-based SOAP implementations to date have implemented RPC-style messages, so that's where we'll spend most of our time; HTTP is a natural for an RPC-style exchange because it allows the request and response to occur as integral parts of a single transaction. However, one-way messaging shouldn't be overlooked, and nothing about HTTP prevents such an exchange. We'll look at one-way messaging in Chapter 8.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

2.2. HTTP Request

The first SOAP message we'll look at is an RPC request. Although it's rather simple, it contains all of the elements required for a fully compliant SOAP message using an HTTP transport. The XML payload of the message is contained in an HTTP POST request. Take a quick look, but don't get too caught up in figuring out the details just yet. The following message asks the server to return the current temperature in degrees Celsius at the server's location:

```
POST /LocalWeather HTTP/1.0
Host: www.mindstrm.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 328
SOAPAction: "WeatherStation"

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
    <m:GetCurrentTemperature xmlns:m="WeatherStation">
         <m:scale>Celsius</m:scale>
    </m:GetCurrentTemperature>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.3. HTTP Response

An RPC-style request message usually results in a corresponding HTTP response. Of course, if the server can't get past the information in the HTTP headers, it can reply with an HTTP error of some kind. But assuming that the headers are processed correctly, the system is expected to respond with a SOAP response. Here's the HTTP response to the RPC-style request from the previous example:

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 359

<SOAP-ENV:Envelope
     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
     SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
     <m:GetCurrentTemperatureResponse xmlns:m="WeatherStation">
          <m:temperature>26.6</m:temperature>
     </m:GetCurrentTemperatureResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.4. The SOAP Envelope

The SOAP envelope represents the entirety of the XML for a SOAP request or response.[3] The `Envelope` is the highest-level XML element in the message, and it must be present for the message to be considered valid. So in essence, the `Envelope` represents the XML document that contains the SOAP message. The `Envelope` can contain an optional `Header` element that, if present, has to be the first subelement of the `Envelope`. The `Envelope` must contain a `Body` element.[4] If the `Envelope` contains a `Header` element,

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

then the Body element has to come right after the Header; otherwise the Body has to be the first subelement of the Envelope.

[3] An exception to this occurs where an attachment is included. We'll look at that possibility in Chapter 8.

[4] There have been some discussions about providing for a Body-less Envelope in SOAP 1.2.

A SOAP envelope packaged and transported using HTTP is similar to a paper envelope sent using the postal service. The SOAP envelope is the paper envelope; the SOAP header (if present) and the SOAP body are the contents of the paper envelope; and the HTTP headers are the physical address information on the outside of the paper envelope.

2.5. The Envelope Element

The Envelope is the topmost element of the XML document that represents the SOAP message. The Envelope is the only XML element at the top level; the rest of the SOAP message resides within the Envelope, encoded as subelements. A SOAP message must have an Envelope element. The Envelope can have namespace declarations, as shown in the earlier examples, and needs to be qualified as shown earlier, using the http://schemas.xmlsoap.org/soap/envelope namespace. That's why the element name is shown as SOAP-ENV:Envelope. It is also common for the Envelope element to declare the encodingStyle attribute, with the attribute namespace-qualified using the declared namespace identifier SOAP-ENV as well.

All subelements and attributes of the Envelope must themselves be namespace-qualified. These elements and attributes are also qualified by SOAP-ENV, just as the Envelope is qualified. For the remainder of this chapter and the rest of the book, we'll use the SOAP-ENV namespace identifier to mean the http://schemas.xmlsoap.org/soap/envelope namespace. This should make for easier reading. Keep in mind that it's the namespace itself that matters, not the name used for the qualifier.

2.6. The Header Element

The SOAP header is optional. If it is present, it must be named Header and be the first subelement of the Envelope element. The Header element is also namespace-qualified using the SOAP-ENV identifier.

Most commonly, the Header entries are used to encode elements used for transaction processing, authentication, or other information related to the processing or routing of the message. This is useful because, as we'll see, the Body element is used for encoding the information that represents an RPC (or other) payload. The Header is an extension mechanism that allows any kind of information that lies outside the semantics of the message in the Body, but is nevertheless useful or even necessary for processing the message properly.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

2.7. The actor Attribute

A SOAP message often passes through one or more intermediaries before being processed. For example, a SOAP proxy service may stand between a client application and the target SOAP service. We'll see an interesting example of this in Chapter 10, where we'll develop a SOAP application service that proxies another service on behalf of the client application, which actually specifies the ultimate service address. Therefore, the header may contain information intended solely for the intermediary as well as information intended for the ultimate destination. The `actor` attribute identifies (either implicitly or explicitly) the intended recipient of a `Header` element.

It's important to understand the requirements that SOAP puts on an intermediary. Essentially, it requires that any SOAP `Header` elements intended for use by an intermediary are not passed on to the next SOAP processor in the path. `Header` elements represent contracts between the sender and receiver. However, if the information contained in a `Header` element intended for an intermediary is also needed by a downstream server, the intermediary can insert the appropriate `Header` in the message to be sent downstream. In fact, the intermediary is free to add any `Header` elements it deems necessary.

2.8. The mustUnderstand Attribute

SOAP includes the concept of optional and mandatory header elements. This doesn't mean that the inclusion of the elements is mandatory — that is an application issue. Instead, "mandatory" means that the recipient is required to understand and make proper use of the information supplied by a `Header` element. This requirement allows us to accommodate situations in which a recipient of a SOAP message can't perform its job unless it knows what to do with the data provided by a specific `Header` element. In this case the element can include the `mustUnderstand` attribute with an assigned value of 1.

This may be necessary if the sending application is upgraded with a new version, for example. That new version may use some new information that has to be processed by the server in order for the result to be useful. Of course you'd expect that there would be a corresponding upgrade to the server, but maybe that hasn't happened yet. Because of the version mismatch, the older version of the server does not understand the new SOAP header element that it received from the upgraded client application. Let's say, for example, that the `username` header element must be understood by the recipient; if it is not, the message should be rejected. We can include this requirement in the SOAP message by assigning the `mustUnderstand` attribute the value of 1. (The value 0 is essentially equivalent to not supplying the attribute.) Let's modify our previous example to indicate that the recipient must understand the `username` element:

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

2.9. The encodingStyle Attribute

The `encodingStyle` attribute specifies the data encoding rules used to serialize and deserialize data elements. It is important to understand that SOAP does not specify any default rules for data serialization. SOAP does, however, specify a simple data typing scheme commonly supported by SOAP implementations. This is the subject of the next chapter, so we won't get into the details of the encoding rules here. However, it's important to understand how to specify the encoding style to be used for serializing and deserializing the element data in the SOAP message.

The `encodingStyle` attribute is namespace-qualified using the `SOAP-ENV` namespace identifier. In the following example we specify the `encodingStyle` attribute as part of the `Envelope` element; we'll see examples in later chapters that make this declaration in the `Body` element instead. Either way works, as long as you recognize that the `encodingStyle` attribute applies to the element in which it was declared as well as all of its subelements (i.e., it's in-scope).

2.10. Envelope Versioning

The SOAP spec doesn't define a numbering system for declaring which version of the SOAP envelope you're using. In SOAP 1.1, all envelopes must be associated with the `http://schemas.xmlsoap.org/soap/envelope` namespace,[6] which we've used in all of the previous examples. That's it. No other versioning information is used for SOAP envelopes at this time. If a SOAP message is associated with any other namespace, or if no namespace is declared, then the recipient has to respond with a SOAP version mismatch. Here is an example of a SOAP fault response for this situation:

[6] For SOAP 1.2, the namespace is http://www.w3.org/2001/12/soap-envelope.

```
HTTP/1.0 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: 311

<SOAP-ENV:Envelope
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode>SOAP-ENV:VersionMismatch</faultcode>
         <faultstring>SOAP Envelope Version Mismatch</faultstring>
      </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.11. The Body Element

The SOAP `Body` element is mandatory in SOAP 1.1. If there is no SOAP header, the `Body` must be the first subelement of the `Envelope`; otherwise it must directly follow the SOAP `Header` element. The `Body` element is also namespace-qualified using the `SOAP-ENV` identifier.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

The `Body` element contains the SOAP request or response. This is where you might find an RPC-style message that contains the method name and its parameters, or a one-way message and its relevant parts, or a fault and its details. SOAP `Body` elements are not completely defined by the SOAP specification; we'll cover that subject in great detail later. In fact, SOAP defines only one kind of `Body`: the SOAP `Fault`.

2.12. SOAP Faults

The `Fault` is the only `Body` element entry defined by SOAP. It's used to carry error information back to the originator of a SOAP message. The `Fault` element must appear as an immediate subelement of the `Body` element, and it can't appear more than once.

SOAP defines four subelements of the `Fault` element. Not all of these are required, and some are appropriate only under certain conditions. These elements are described in the following sections. You'll probably recognize that we've seen all of these in previous examples, so you may want to flip back and look at them again after you've read these descriptions.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01