SOAP Fundamentals

.NET

Simple Object Access Protocol is based on transferring XML data as SOAP Messages. Each message is an XML document. The structure of the XML document follows a specific pattern, but not the content. In SOAP, messages are all sent via HTTP.
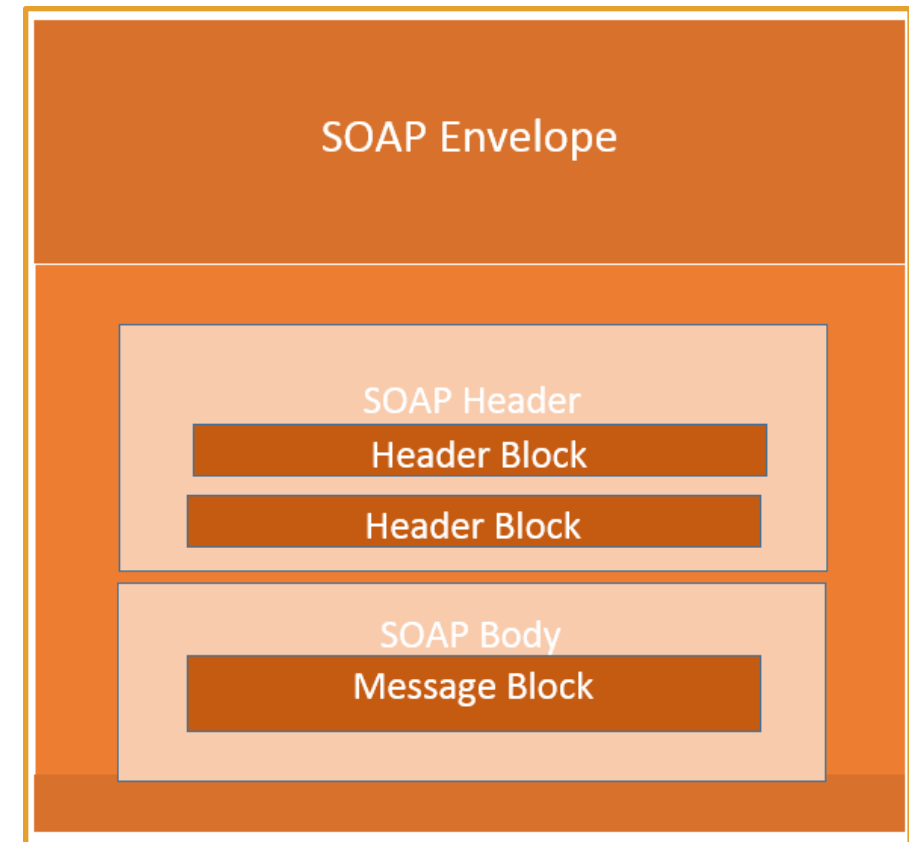
SOAP – Overview

SOAP (Simple Object Access Protocol) is a messaging protocol specification for exchanging structured information in the implementation of web services.

SOAP has three major characteristics:

1. *extensibility* (security and WS-Addressing are under development)
2. *neutrality* (SOAP can operate over any protocol such as HTTP, SMTP, TCP, UDP)
3. *independence* (SOAP allows for any programming model)
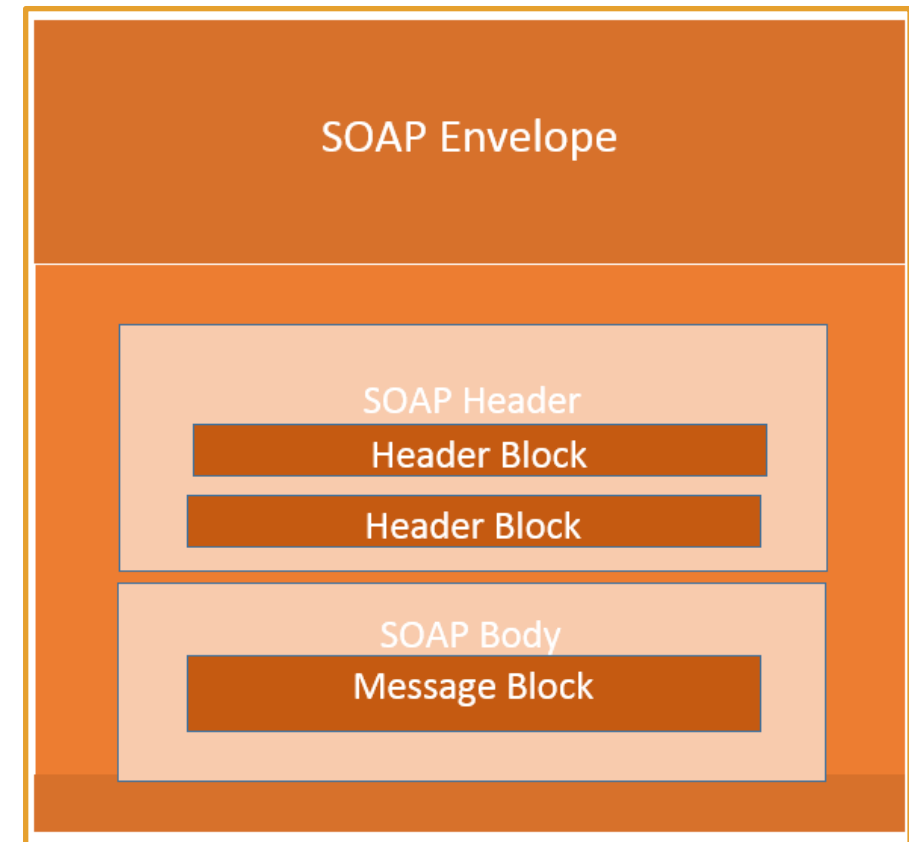
SOAP Web Services – Overview

SOAP (Simple Object Access Protocol) is a protocol that defines how *Web Services* communicate with each other or with client applications that invoke them.

Every programming language can understand the *XML* markup language. SOAP uses *XML* for data exchange.

Because there are no standard specifications on use of *XML* across all programming languages for data exchange, *SOAP* was designed to work with *XML* over *HTTP* and have some sort of specification which could be used across all applications. *W3C* recommends *SOAP* as the medium of exchange between client and *Web Service*.

The *SOAP* specification defines something known as a "*SOAP message*" which is an *XML* document sent between the *Web Service* and the client application.

SOAP Web Service Requirements

Every **SOAP Web Service** must have certain foundational characteristics to function.

- SOAP messaging – *Simple Object Access Protocol* messages contain an XML document with a standardized structure. SOAP messages are most often sent over HTTP but <u>can</u> use any protocol.

- WSDL – *Web Services Description Language* is an XML based file which tells the client what a *Web Service* does and how to communicate with it.

- UDDI – *Universal Description, Discover, and Integration* is a repository where WSDL files can be published by a *Web Service* provider. Any client has access to the *UDDI*. UDDI was created by <u>OASIS</u>.
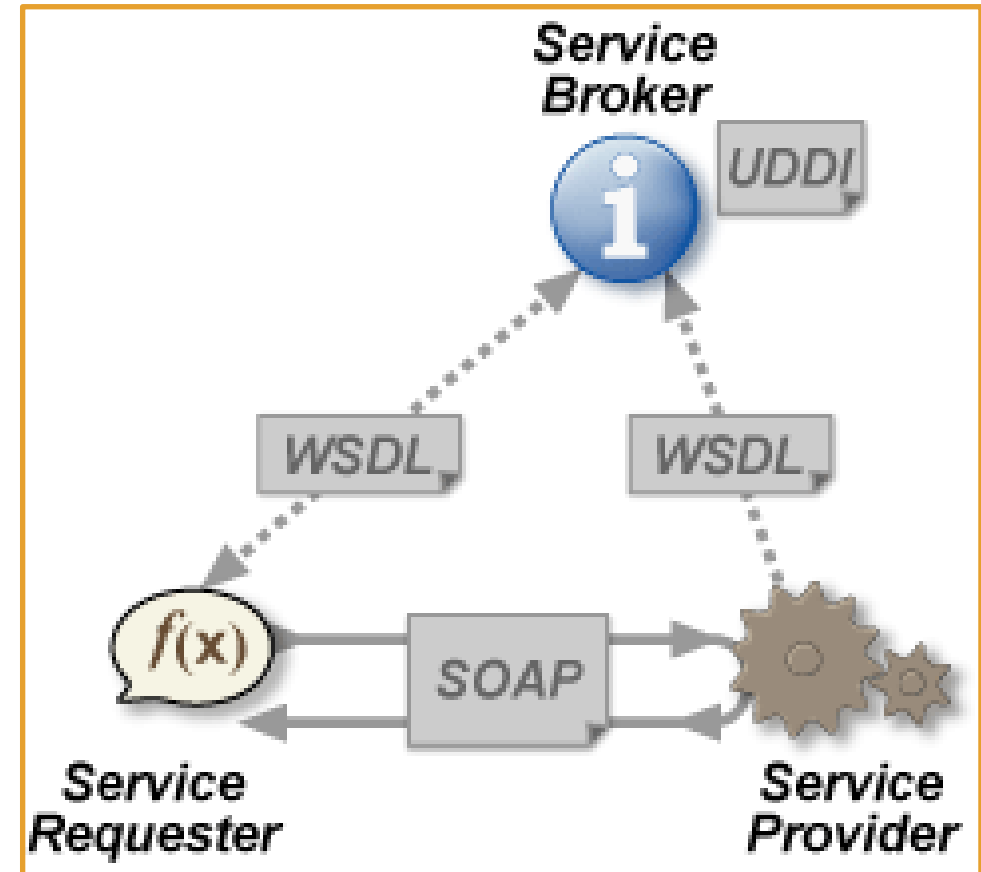
XML and XML Schema

XML(Extensible Markup Language)	XML Schema
Software and Hardware independent.	Describes the structure of an accompanying XML doc.
Stores and transports data in plain text format.	Supports Namespaces and Datatypes.
Is self-descriptive.	Written in XML.
Is recommended by *W3C*.	Can serve as a contract for how client and provider communicate.
XML tags are not predefined. The tag is the *Key* of the *key-value* pair.	

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```xml
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```
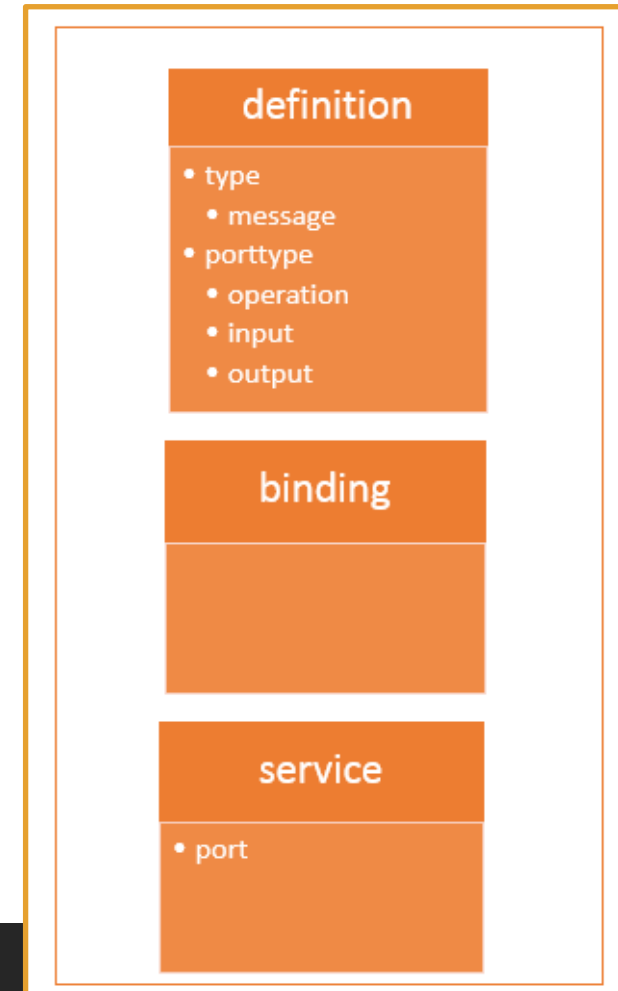
WSDL – Example and Explanation

The **WSDL** describes what the **Web Service** does, what it requires from the client, and what it provides to the client. A **WSDL** contains:

- the location of the **Web Service**,
- all the information required to connect to the **Web Service**
- all the methods and functionality provided the Web Service.

The definitions and **types** of **SOAP** messages passed by the **SOAP** protocol is defined in the **WSDL** document.

A **WSDL** contains these elements:

- Definition, TargetNamespace, DataType, Message, PortType, Binding, Service

WSDL Tag Elements

Tag Name	Purpose
<types>	Defines all complex data types in the messages
<message>	Defines the message which to be exchanged between the client application and the web server. There are two <message> tags in each *WSDL*. One for input parameters and one for output parameters.
<portType>	Used to define a complete input/output operation provided by the *Web Service*. Gives the names of the input and output <message>.
<binding>	Used to define how the messages will be transferred (*HTTP* or other). Port types act like interfaces, so the client must call the relevant port to ask for a particular functionality.
<service>	Declares the name given to the *Web Service* itself. This is a web address used for confirmation that the service exists.

WSDL File Example

Tag Name	Purpose
`<types>`	Defines all complex data types in the messages
`<message>`	Defines the message which is exchanged between the client application and the web server. There are two *<message>* tags in each *WSDL*. One for input parameters and one for output parameters.
`<portType>`	Used to define a complete input/output operation provided by the *Web Service*. Gives the names of the input and output *<message>*.
`<binding>`	Used to define how the messages will be transferred (HTTP or other). Port types act like interfaces, so the client must call the relevant port to ask for a particular functionality.
`<service>`	The name given to the *Web Service* itself. This is a web address used for confirmation that the service exists.

```xml
<?xml version="1.0"?>
<definitions name="Tutorial"
            targetNamespace=http://Guru99.com/Tutorial.wsdl
        xmlns:tns=http://Guru99.com/Tutorial.wsdl
        xmlns:xsd1=http://Guru99.com/Tutorial.xsd
        xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
            <schema targetNamespace=http://Guru99.com/Tutorial.xsd
        xmlns="http://www.w3.org/2000/10/XMLSchema">

        <element name="TutorialNameRequest">
                <complexType>
                <all>
                        <element name="TutorialName" type="string"/>
                </all>
                </complexType>
        </element>
        <element name="TutorialIDRequest">
                <complexType>
                <all>
                        <element name="TutorialID" type="number"/>
                </all>
                </complexType>
        </element>
        </schema>
  </types>
</types>
<message name="GetTutorialNameInput">
        <part name="body" element="xsd1:TutorialIDRequest"/>
</message>
<message name="GetTutorialNameOutput">
        <part name="body" element="xsd1:TutorialNameRequest"/>
</message>
<portType name="TutorialPortType">
        <operation name="GetTutorialName">
        <input message="tns:GetTutorialNameInput"/>
        <output message="tns:GetTutorialNameOutput"/>
    </operation>
</portType>
<binding name="TutorialSoapBinding" type="tns:TutorialPortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/htt
p"/>
        <operation name="GetTutorialName">
                <soap:operation soapAction="http://Guru99.com/GetTutorialName"/
>
                <input>
                <soap:body use="literal"/>
            </input>
            <output>
        <soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="TutorialService">
        <documentation>TutorialService</documentation>
        <port name="TutorialPort" binding="tns:TutorialSoapBinding">
            <soap:address location="http://Guru99.com/Tutorial"/>
    </port>
</service>
</definitions>
```
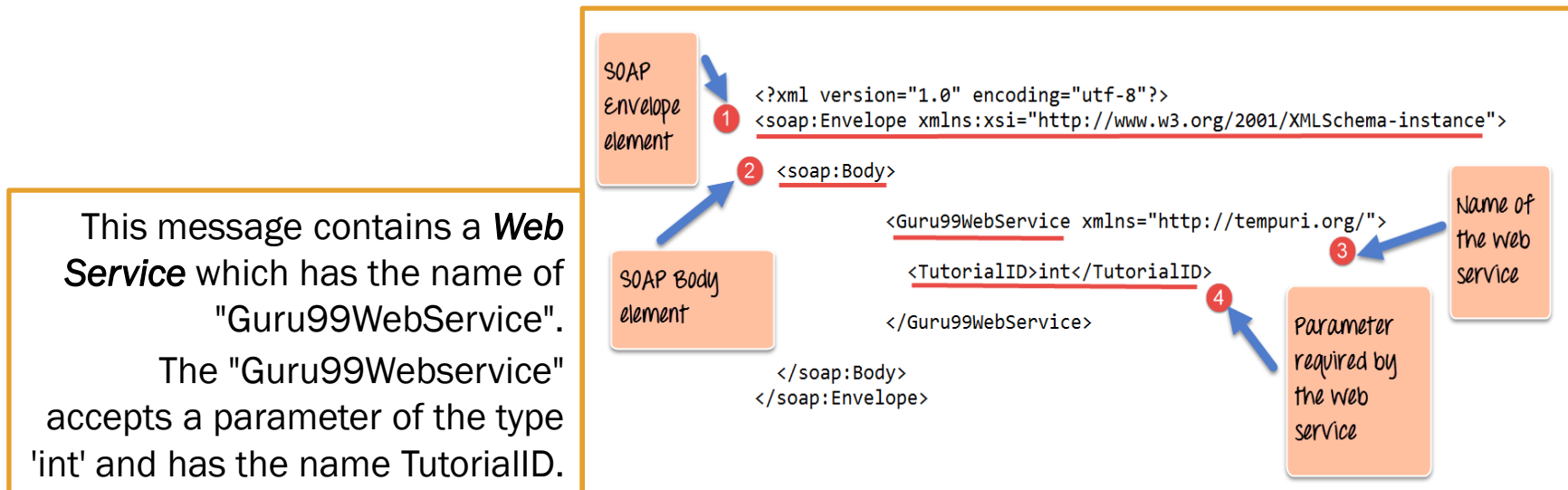
SOAP Message Elements

https://www.guru99.com/soap-simple-object-access-protocol.html

- Envelope – This element is mandatory and identifies the message as a **SOAP** message. The **Envelope** is the **Root** element and contains all other parts.
- Header – This optional element can contain authentication credentials and complex data type definitions.
- Body – There is one Body. It is mandatory and it contains the data being sent.

SOAP Message – Envelope

- The *SOAP Envelope* is mandatory

- The *SOAP Envelope* encapsulates all the other elements.

- The *Envelope* header is optional.

- The *Envelope* may only contain one header element.

- The *SOAP Envelope* must have exactly one *Body* element.

- The *SOAP Envelope Header* must the first child of the *Envelope*.

- The *SOAP Envelope* changes when versions change. A *Fault* is generated when *Envelope* versions are mismatched.

```xml
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header>
<m:reservation xmlns:m="http://travelcompany.example.org/reservation"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
<m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
<m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
</m:reservation>
<n:passenger xmlns:n="http://mycompany.example.com/employees"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
<n:name>Åke Jógvan Øyvind</n:name>
</n:passenger>
</env:Header>
<env:Body>
<p:itinerary
 xmlns:p="http://travelcompany.example.org/reservation/travel">
<p:departure>
 <p:departing>New York</p:departing>
 <p:arriving>Los Angeles</p:arriving>
 <p:departureDate>2001-12-14</p:departureDate>
 <p:departureTime>late afternoon</p:departureTime>
 <p:seatPreference>aisle</p:seatPreference>
</p:departure>
<p:return>
 <p:departing>Los Angeles</p:departing>
 <p:arriving>New York</p:arriving>
 <p:departureDate>2001-12-20</p:departureDate>
 <p:departureTime>mid-morning</p:departureTime>
 <p:seatPreference/>
</p:return>
</p:itinerary>
<q:lodging
 xmlns:q="http://travelcompany.example.org/reservation/hotels">
 <q:preference>none</q:preference>
</q:lodging>
</env:Body>
</env:Envelope>
```
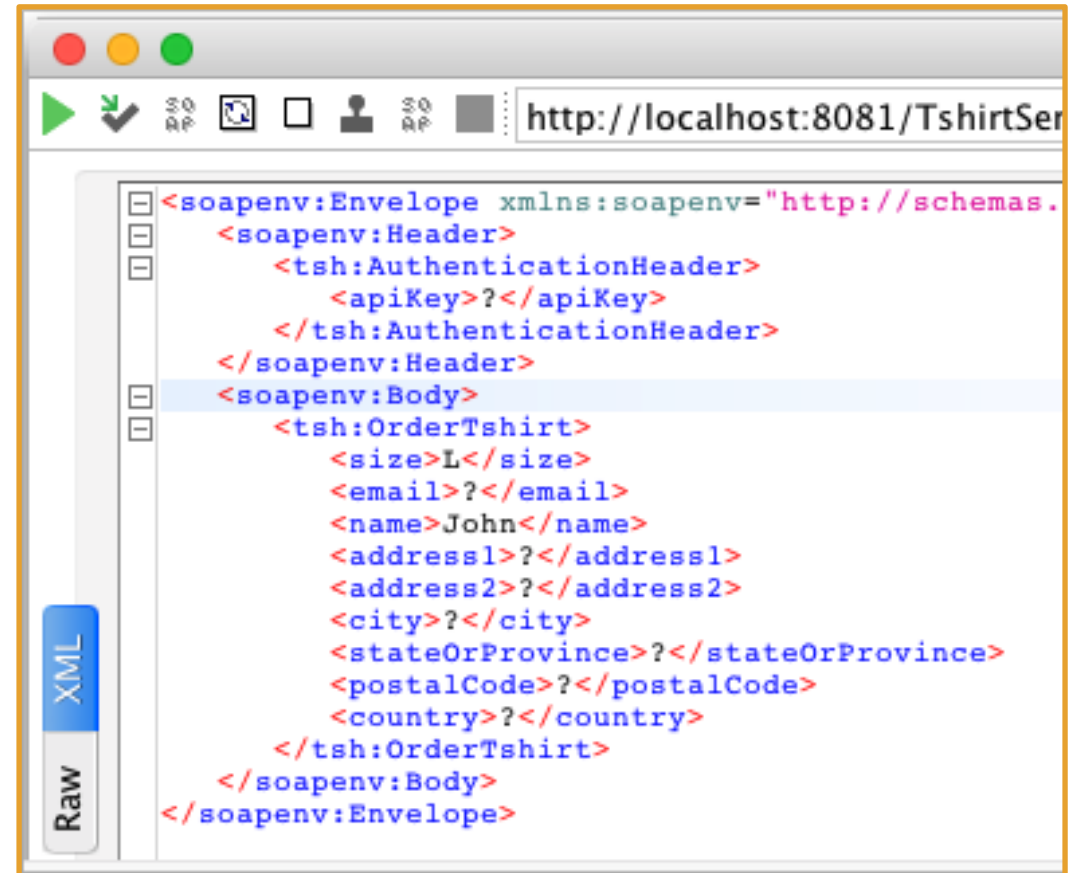
SOAP Message - Header

The **SOAP** <Header> element is optional in a **SOAP** message. It is used to pass application-related information that is to be processed by **SOAP** nodes along the message path.

The immediate child elements of the <Header> element are called *header blocks*. A *header block* is an application-defined *XML* element. *Header Blocks* can be targeted by child elements in the body of the **SOAP** message.

SOAP Message - Header

Header blocks are application-defined. *SOAP-defined* attributes on the *header* blocks indicate how the *header* blocks are to be processed by the *SOAP* nodes.

- encodingStyle - Indicates the rules used to encode the parts of a *SOAP* message.
- role/actor – <role> and <actor> can be assigned to a message in the header. If a <body> node has a matching assignment, the node is processed.
- mustUnderstand – Used to ensure that *SOAP* nodes do not ignore important <header> blocks
- Relay - when <relay> is specified with a value of true, the node retains the unprocessed <header> block in the *message* if it otherwise would have discarded it.

http://localhost:8081/TshirtSer

```
<soapenv:Envelope xmlns:soapenv="http://schemas.
    <soapenv:Header>
        <tsh:AuthenticationHeader>
            <apiKey>?</apiKey>
        </tsh:AuthenticationHeader>
    </soapenv:Header>
    <soapenv:Body>
        <tsh:OrderTshirt>
            <size>L</size>
            <email>?</email>
            <name>John</name>
            <address1>?</address1>
            <address2>?</address2>
            <city>?</city>
            <stateOrProvince>?</stateOrProvince>
            <postalCode>?</postalCode>
            <country>?</country>
        </tsh:OrderTshirt>
    </soapenv:Body>
</soapenv:Envelope>
```
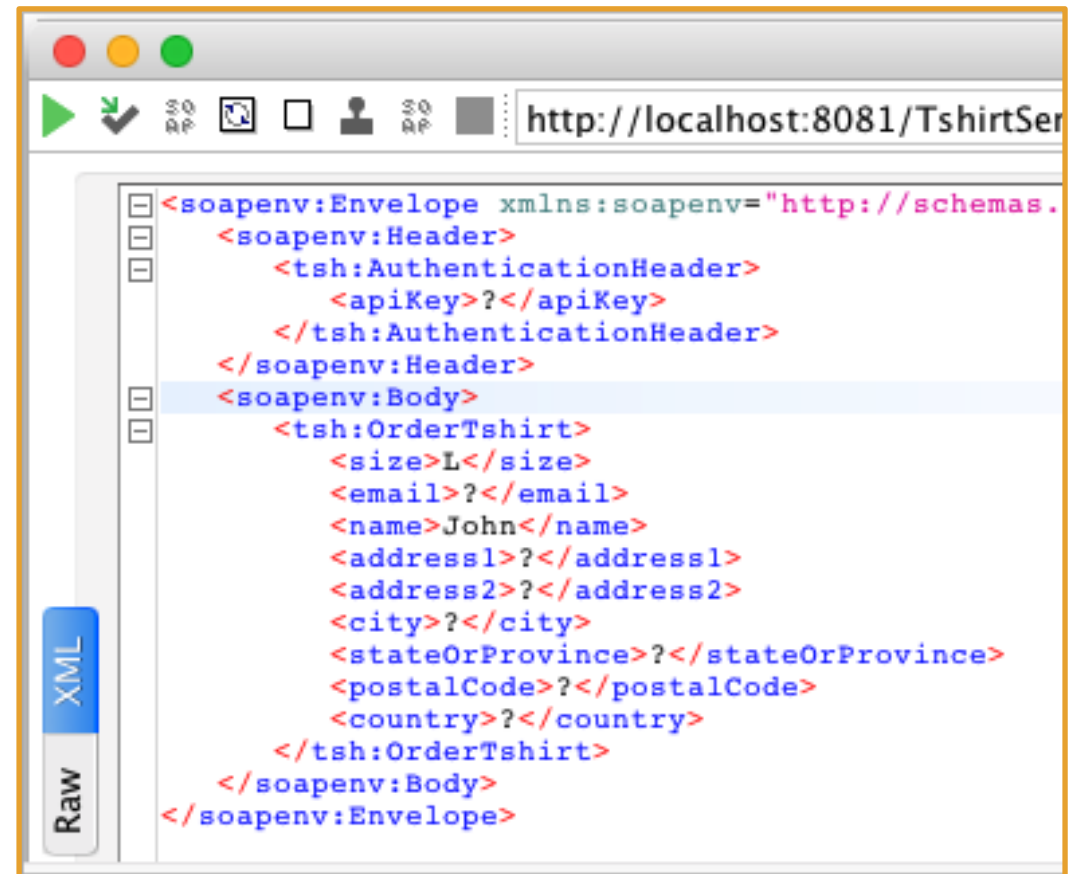
XML

Raw

SOAP Message - Fault

A **SOAP** response can be either "successful" or "error". "Success" means a **SOAP** message will be returned. Failure means a "HTTP 500" is sent.

The **Fault** message contains the following elements.

Fault Element	Meaning
<faultCode>	Gives the code of the error. Possible values are 'VersionMismatch', 'MustUnderstand', 'Client', 'Server'
<faultString>	A text message which gives a detailed description of the error.
<faultActor>	(Optional)A text string telling who caused the fault
<detail>	(Optional) Gives application-specific error messages.

SOAP Fault Message

```xml
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <SOAP-ENV:Body>
          <SOAP-ENV:Fault>
          <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
          <faultstring xsi:type="xsd:string">
              Failed to locate method (GetTutorialID) in class (GetTutorial)
          </faultstring>
          </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP in the .NET World

https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf

In .NET Framework, you use *WCF (Windows Communication Foundation)* when you want to use *SOAP*.

WCF is a framework for building service-oriented applications with a *Service Oriented Access Protocol* (another meaning of *SOAP*). Using *WCF*, you can send data as asynchronous messages from one service endpoint (a URL) to another.

These service endpoints can:

- Process transactions
- Send chat messages
- Supply data like Star Wars character data or Chuck Norris Jokes.

WCF is designed to offer a manageable, easy approach to creating *Web services* and *Web service* clients <u>(with SOAP)</u>.

WCF Features

Service	Details
Service Orientation	Create loosely-coupled services so that any client can connect to any service.
Service Metadata	WCF supports publishing service metadata (like WSDL*).
Data Contracts	You can use C# classes to represent data. The .NET Framework automatically creates the metadata that allows clients to comply with the data types you designed.
Durable Messages	Messages are saved to a DB.
AJAX and REST support	WCF can be configured to process "plain" XML data that is not wrapped in a SOAP envelope and also be extended to support specific XML formats

*And many others

A Practical SOAP Example

https://docs.microsoft.com/en-us/dotnet/framework/wcf/how-to-define-a-wcf-service-contract
https://docs.microsoft.com/en-us/dotnet/framework/wcf/getting-started-tutorial

1. Make sure you have WCF installed. => VS Installer =>Modify => Install Windows Communication Foundation
2. Open VS as an admin => new Project => C# => search WCF => WCF Service Library
3. Tutorial here.

The next task for creating a WCF application is to create a *client* by retrieving metadata from a WCF service. You use Visual Studio to add a service reference, which gets the metadata from the service's MEX endpoint. Visual Studio then generates a managed source code file for a client proxy in the language you've chosen. It also creates a client configuration file (*App.config*). This file enables the client application to connect to the service at an endpoint.
Tutorial: Use A Client

https://docs.microsoft.com/en-us/dotnet/framework/wcf/how-to-use-a-wcf-client

Q/A - The service keeps data as long as it is still running. So a List<> will persist till you shut the service down.

SOAP extra.

SOAP is protocol neutral. SOAP doesn't care how you send SOAP messages. SOAP doesn't enforce node roles. Messages are formatted in XML.

SOAP uses HTTP in practice.
- Typically sent over HTTP using POST.
- Client-server, distinction, request/response cycle
- The contract between the *SOAP* server and the client is the *WSDL* doc(contract).

SOAP in the .NET world:

.NET uses WCF(Windows Communication Foundation)

WCF is Microsoft proprietary

SOAP in the .NET world

SOAP as usually practiced

typically sent over HTTP using POST

SOAP itself

client-server distinction, request/response cycle

transport protocol-neutral

the contract between SOAP server & client is a WSDL document

doesn't enforce node roles

message formatted in XML

framework called WCF (Windows Communication Foundation)

WCF is only for .NET Framework

(.NET Core we can do SOAP clients, but not SOAP servers, at least not with WCF)

in WCF...

we have "endpoints" basically a service each endpoint has:

⊹ address
the URL you need to send requests to

- binding
what transport protocol to use

- contract
all the rules for data format, what operations the service exposes