Web Services and SOAP Fundamentals

.NET CORE

***Web Services** are XML-based, Async capable applications hosted online. They are Loosely-Coupled and can be invoked by any client using the Remote Procedure Calls (RPC).*

HTTPS://WWW.GURU99.COM/WEB-SERVICE-ARCHITECTURE.HTML#1

Web Services - Overview

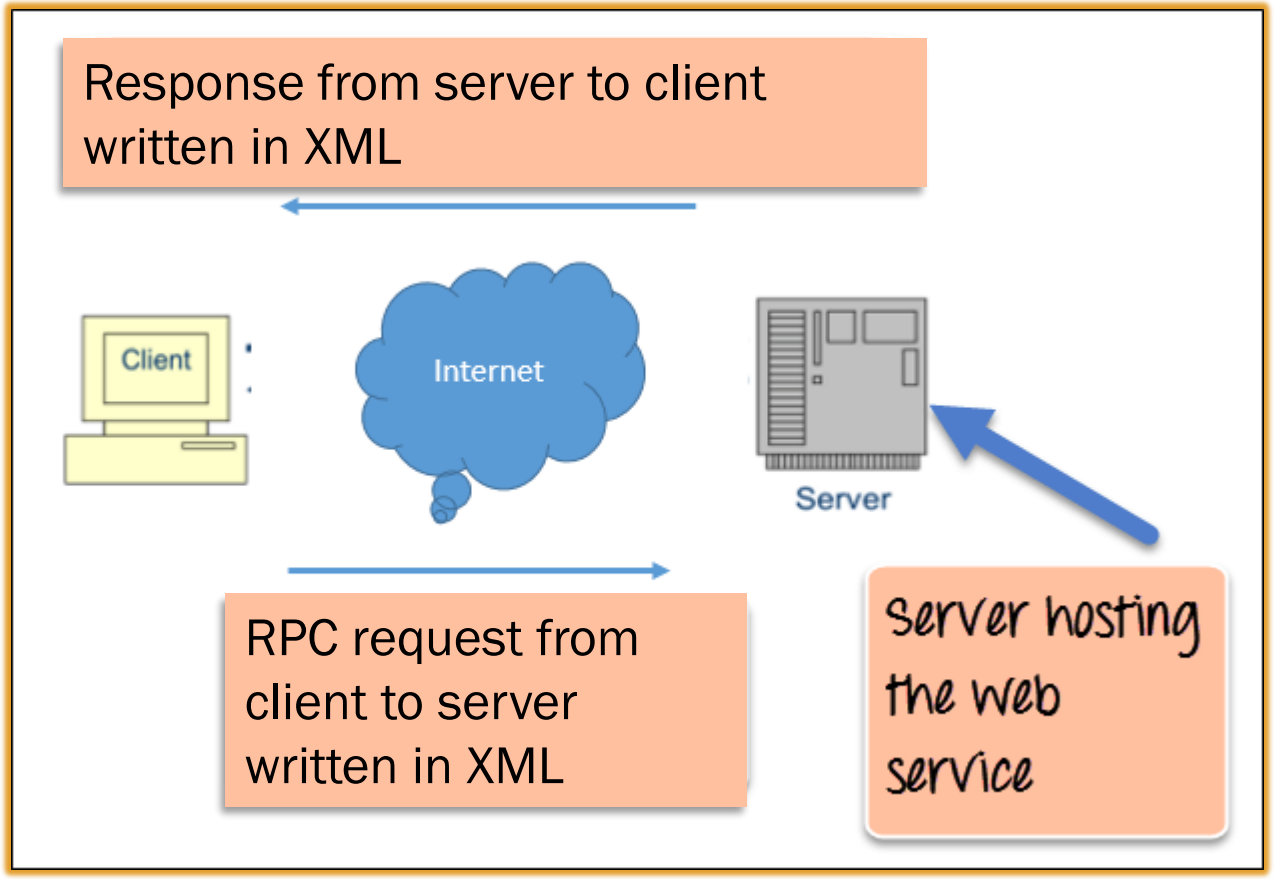
https://www.guru99.com/web-service-architecture.html

A ***Web service*** is an application with a standardized communication method. This standardization allows communication between any client and any server application, written in any language, on the Web.

The methods of a Web Service are searched for and invoked, over a network using ***Remote Procedure Calls (RPC)***. ***RPC***'s are written in ***XML (Extensible Markup Language)***.

When invoked, the Web Service is be able to provide functionality to a client invoking that Web Service.

Because ***RPC***'s are standardized to ***XML***, client and server applications can be written in any language.

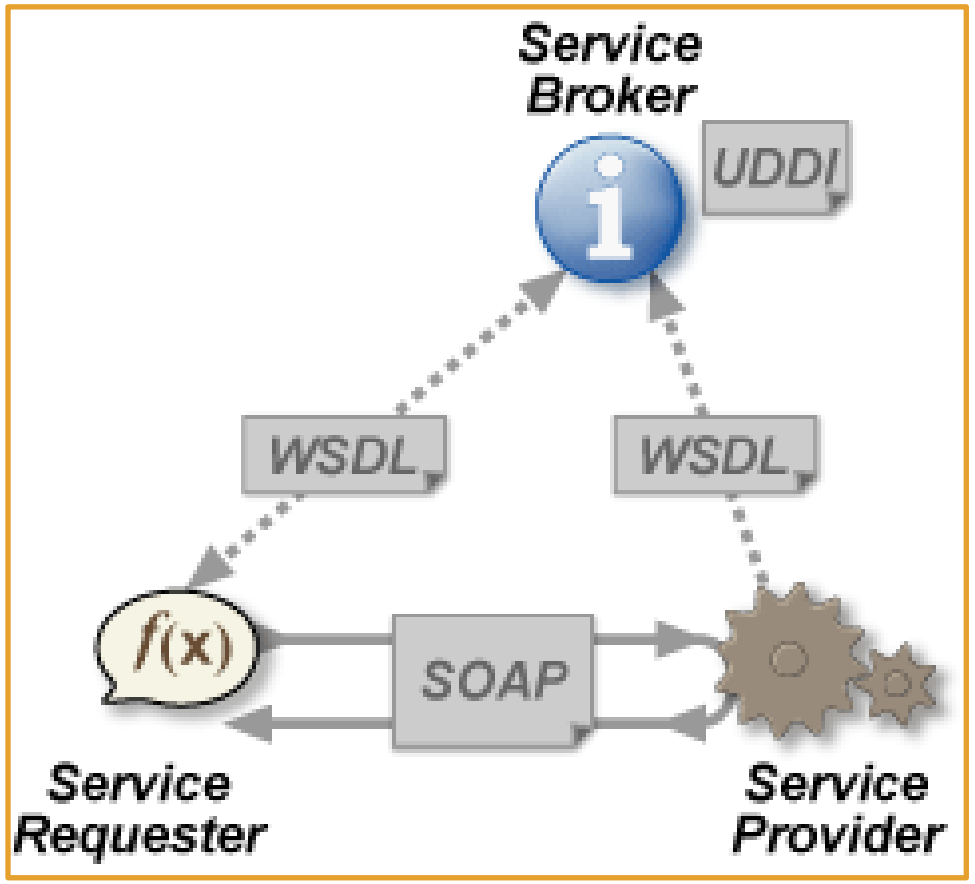


Web Services – Requirements

https://en.wikipedia.org/wiki/Web_service
https://www.guru99.com/web-service-architecture.html#1

Every Web Service must have certain foundational characteristics to function.

- SOAP messaging – ***Simple Object Access Protocol*** messages contain an XML document which has a standard structure. ALL SOAP messages are sent over HTTP.
- WSDL – Web Services Description Language is an XML based file telling the client what exactly the web service does and how to communicate with it.
- UDDI – ***Universal Description, Discover, and Integration*** is a repository created by OASIS where WSDL files can be published by a Web Service (provider). Any client (requester) has access to the UDDI.



XML and XML Schema

https://www.w3schools.com/xml/xml_whatis.asp
https://www.w3schools.com/xml/xml_schema.asp

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

https://www.guru99.com/wsdl-web-services-description-language.html

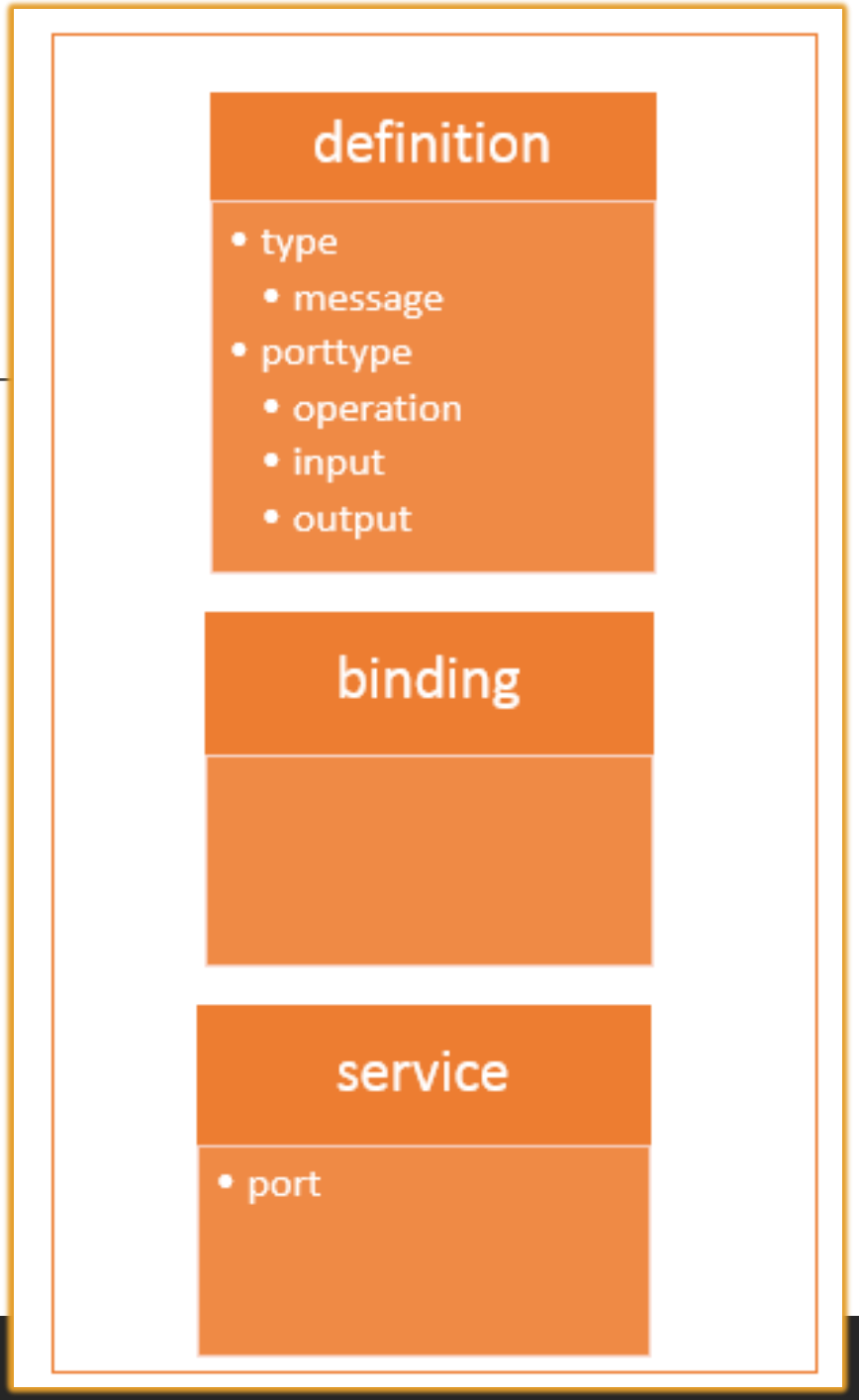
The ***WSDL*** describes what the ***Web Service*** does and gives to the client.

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

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

The WSDL contains these elements:

- Definition, TargetNamespace, DataType, Messages, PortType, Bindings, Service



WSDL Tag Elements

https://www.guru99.com/wsdl-web-services-description-language.html

Tag Name	Purpose
<types>	Defines all complex dataTypes 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.

WSDL File Example

https://www.guru99.com/wsdl-web-services-description-language.html

```
<?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>
 <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 Web Services – Overview

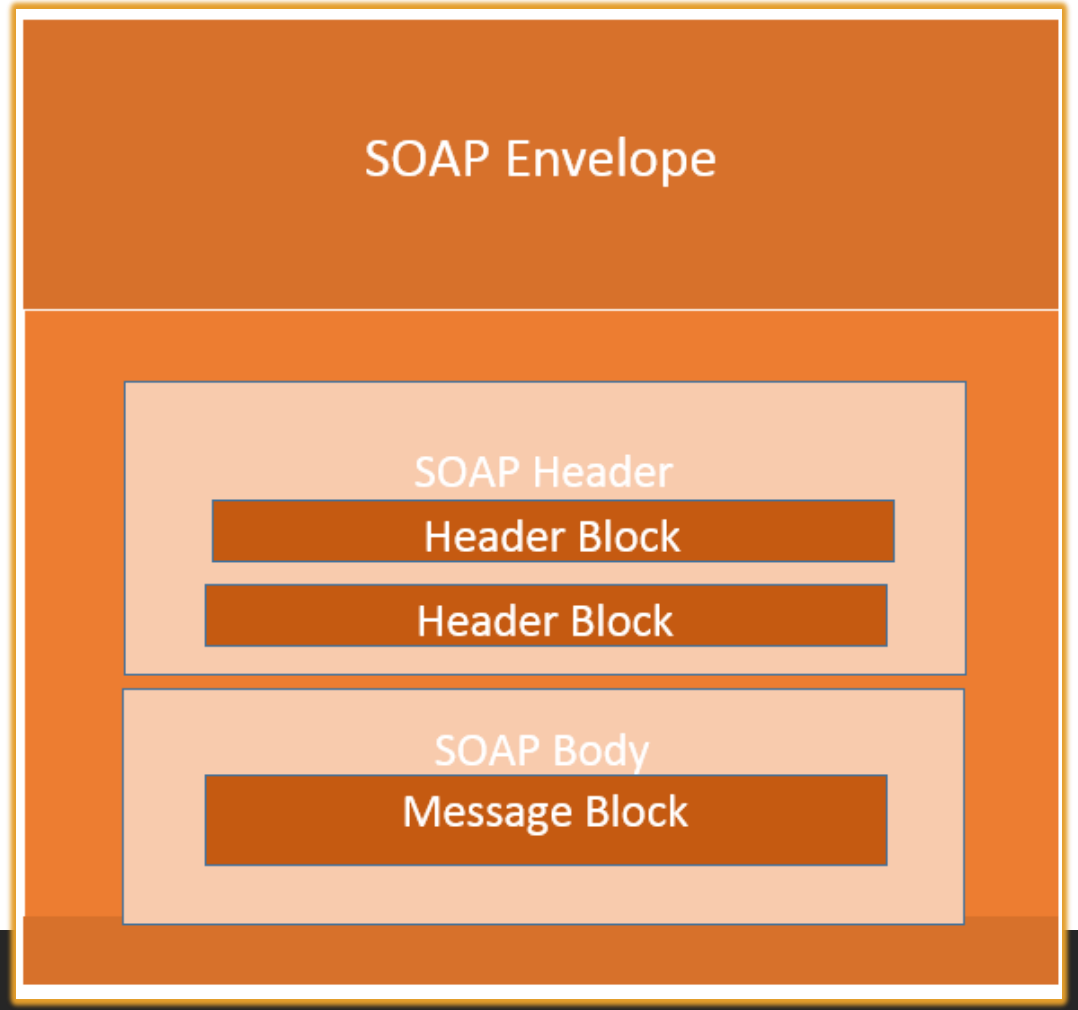
https://www.guru99.com/soap-simple-object-access-protocol.html

SOAP is an intermediate markup language that defines how Web Services talk to each other or talk to client applications that invoke them.

Every programming language can understand the *XML* markup language. Hence, *XML* is used as the underlying medium 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 merely and XML document which is sent between the *Web Service* and the client application.



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. This is the Root element and contains all other parts.
- Header – Thi optional element can contain authentication credentials and complex data type definitions.
- Body – contains the data being sent.

This message contains a web service which has the name of "Guru99WebService".
The "Guru99Webservice" accepts a parameter of the type 'int' and has the name TutorialID.

SOAP Message – Envelope

https://www.guru99.com/soap-simple-object-access-protocol.html
https://www.ibm.com/support/knowledgecenter/SSGMCP_5.4.0/fundamentals/web-services/dfhws_header.html

- The ***SOAP Envelope*** is mandatory
- The ***SOAP Envelope*** encapsulates all the other elements.
- The ***Envelope*** header is optional and may only contain one header element.
- The ***SOAP Envelope*** must have at least 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 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 - Fault

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

A SOAP response can be either a “successful” response or an “error” response. “Success” means a SOAP message will be returned. Failure means a “HTTP 500” is sent. The Fault message contains these 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

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

```
<?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.w
3.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>
```

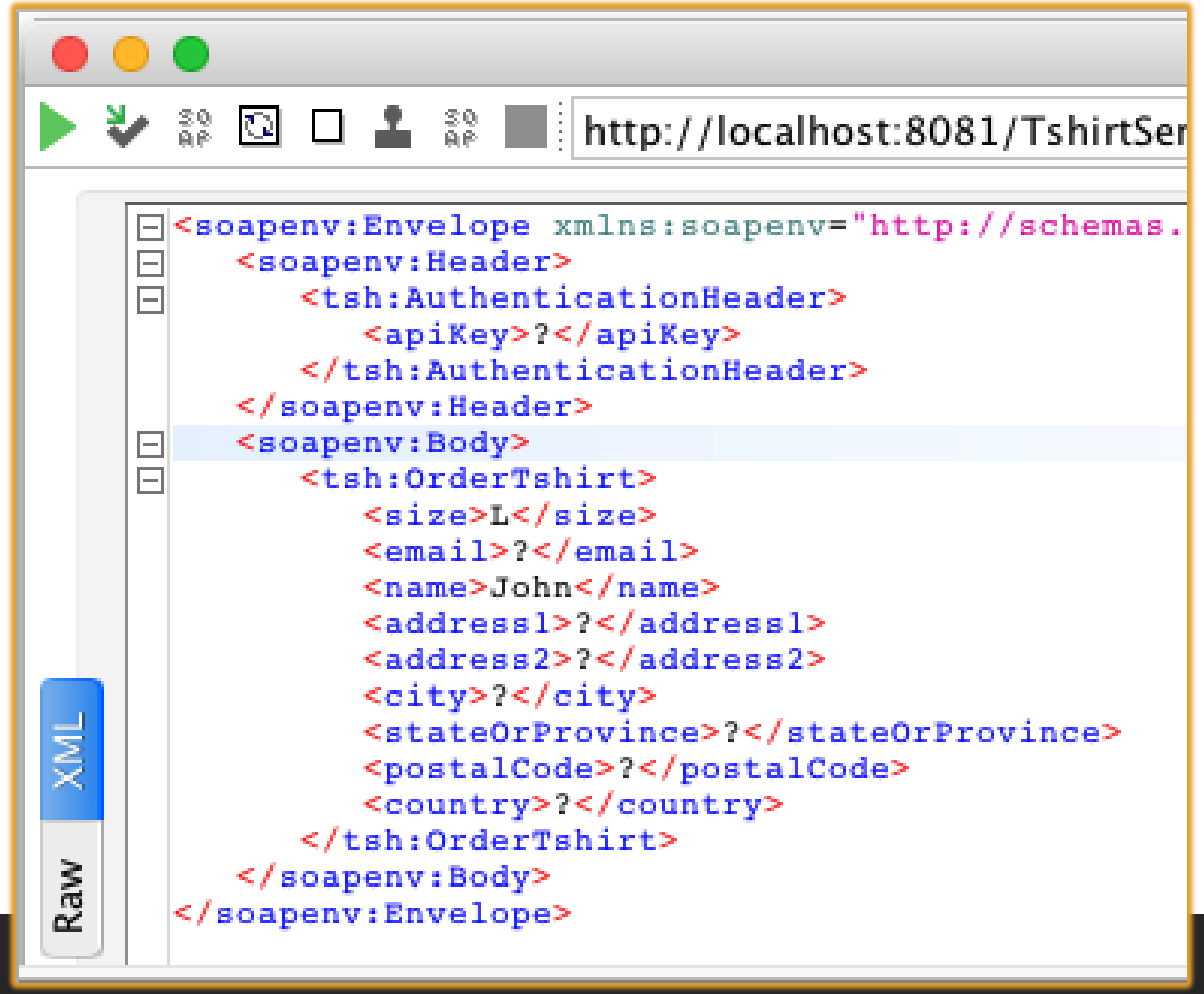
SOAP Message - Header

https://www.ibm.com/support/knowledgecenter/SSGMCP_5.4.0/fundamentals/web-services/dfhws_header.html
https://docs.mulesoft.com/apikit/4.x/apikit-4-get-header-task

---

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.

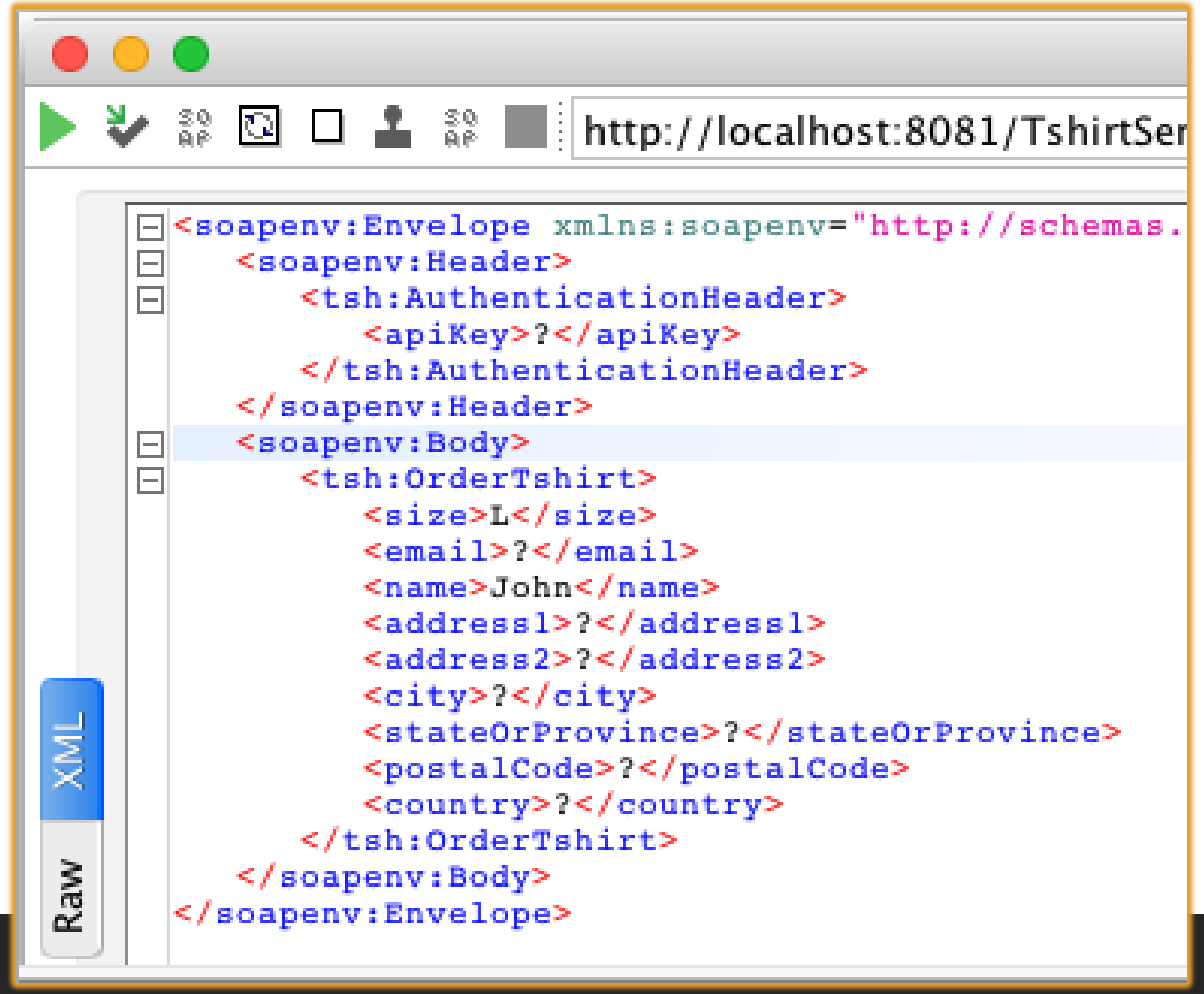


SOAP Message - Header

https://www.ibm.com/support/knowledgecenter/SSGMCP_5.4.0/fundamentals/web-services/dfhws_header.html

Although the 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 specified with a value of ***true***, the node retains the unprocessed header block in the message if it otherwise would have discarded it..



SOAP in the .NET World

In .NET Framework, you use WCF (Windows Communication Foundation).

A Practical SOAP Example

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

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

SOAP

https://www.ibm.com/support/knowledgecenter/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html
https://www.ibm.com/support/knowledgecenter/SSAW57_9.0.5/com.ibm.websphere.nd.multiplatform.doc/ae/cwbs_soap.html

SOAP in theory – protocol neutral. SOAP doesn't care how you send SOAP itself. Doesn't enforce node roles. Message formatted in XML

SOAP in practice – you use 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 – we use WCF(Windows Communication Foundation)(Microsoft proprietary)