



# Web Services and SOAP Fundamentals

---

.NET CORE

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

[HTTPS://WWW.GURU99.COM/WEB-SERVICE-ARCHITECTURE.HTML#1](https://www.guru99.com/web-service-architecture.html#1)

# Web Services - Overview

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

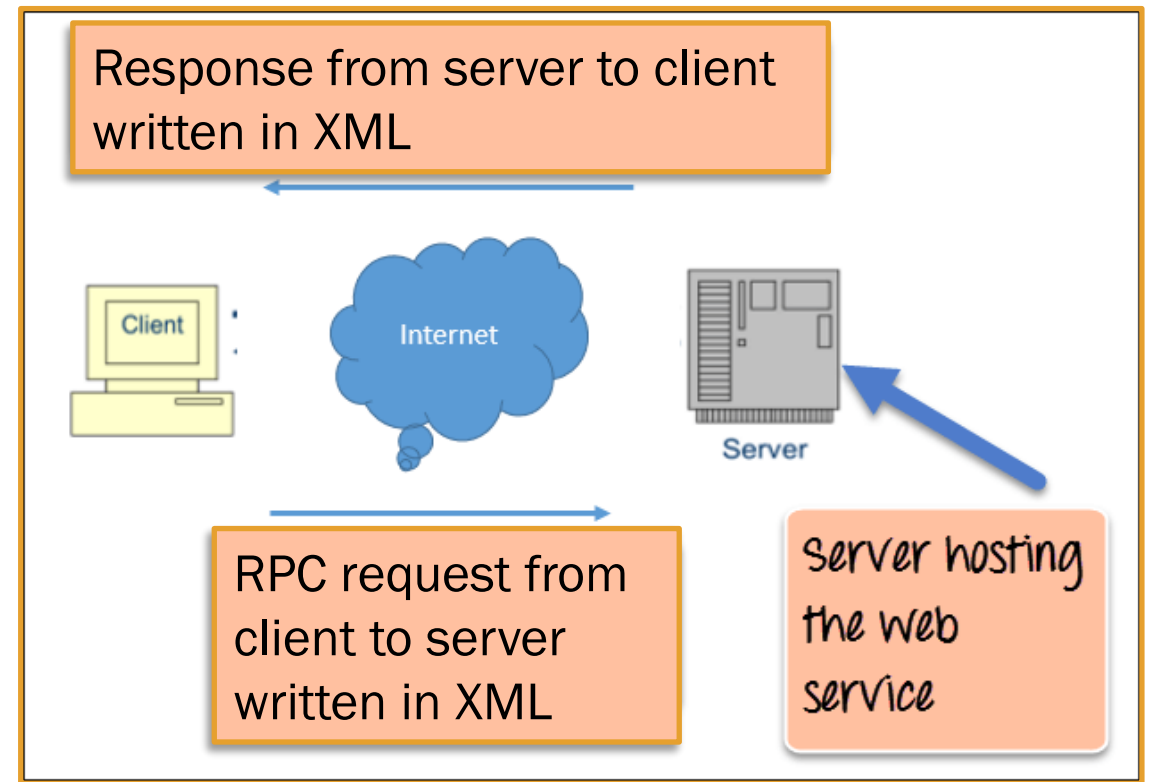
<https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf#features-of-wcf>

---

A **Web Service** is an application with a standardized communication method. This standardization allows communication between web client and server applications.

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)**.

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



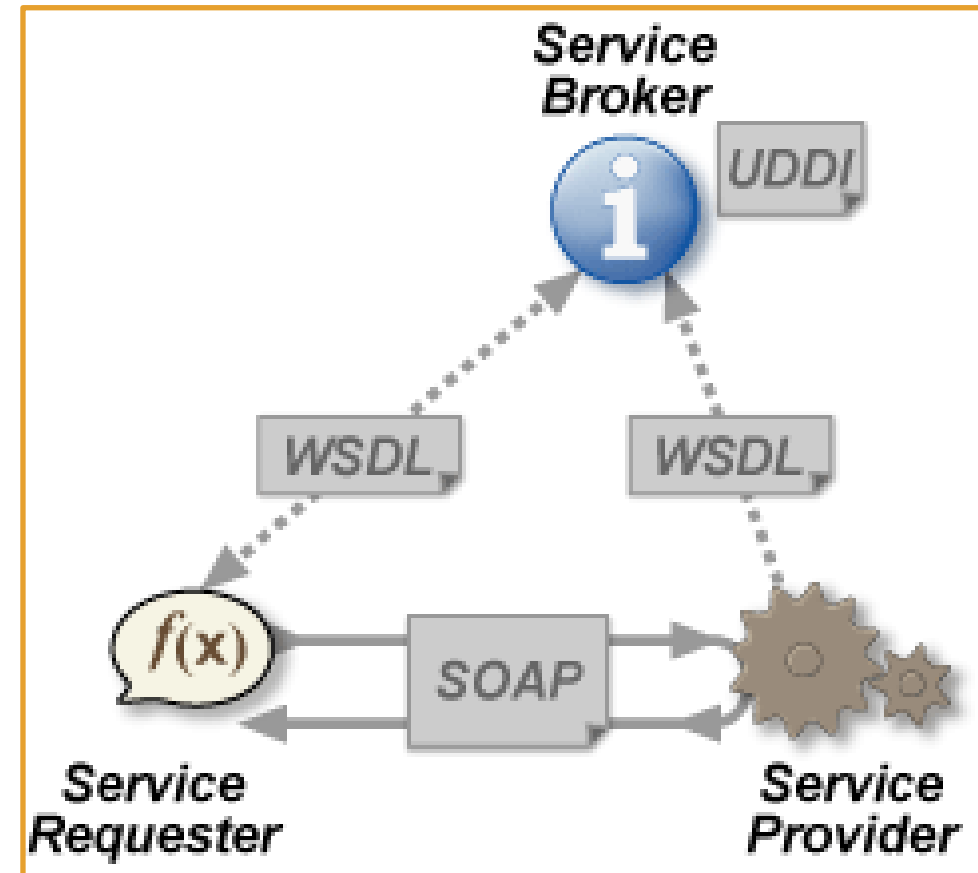
# Web Services – Requirements

[https://en.wikipedia.org/wiki/Web\\_service](https://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\\_what\\_is.asp](https://www.w3schools.com/xml/xml_what_is.asp)

[https://www.w3schools.com/xml/xml\\_schema.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 <b>W3C</b> .	Can serve as a contract for how to communicate.
<i>XML</i> tags are not predefined. The tag is the <b>Key</b> of a <b>key-value</b> pair.	

```
<?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>
```

```
<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, what is requires from the client, and what it gives 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 it.

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, Messages, PortType, Bindings, Service

## definition

- type
  - message
- porttype
  - operation
  - input
  - output

## binding

## service

- port



# WSDL Tag Elements

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

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 <b>&lt;message&gt;</b> tags in each <b>WSDL</b> . One for input parameters and one for output parameters.
<portType>	Used to define a complete input/output operation provided by the <b>Web Service</b> . Gives the names of the input and output <b>&lt;message&gt;</b> .
<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 <b>Web Service</b> 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>

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 <b>&lt;message&gt;</b> tags in each <b>WSDL</b> . One for input parameters and one for output parameters.
<portType>	Used to define a complete input/output operation provided by the <b>Web Service</b> . Gives the names of the input and output <b>&lt;message&gt;</b> .
<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 <b>Web Service</b> itself. This is a web address used for confirmation that the service exists.

```
<?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/http"/>
    <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

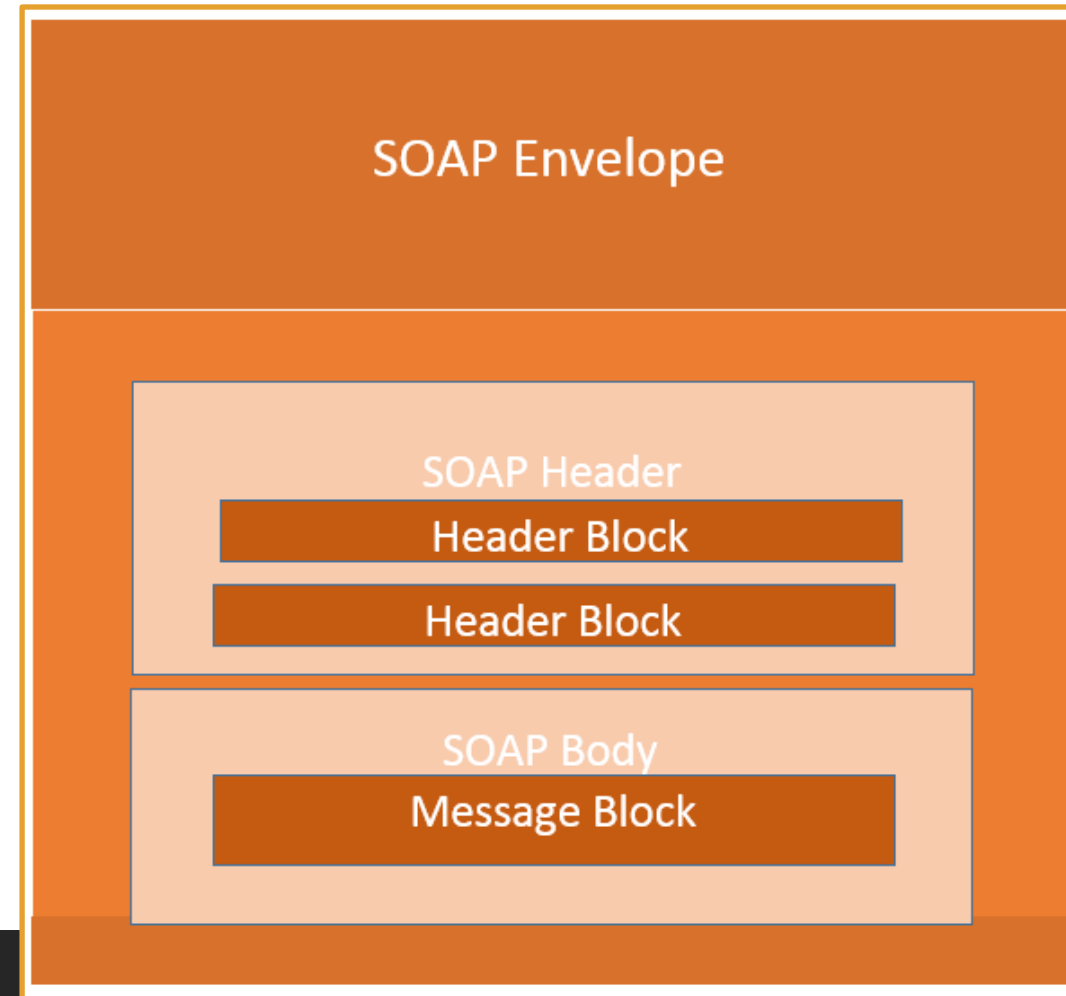
<https://www.guru99.com/soap-simple-object-access-protocol.html>

**SOAP** (*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. **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 an **XML** document that is sent between the **Web Service** and the client application.

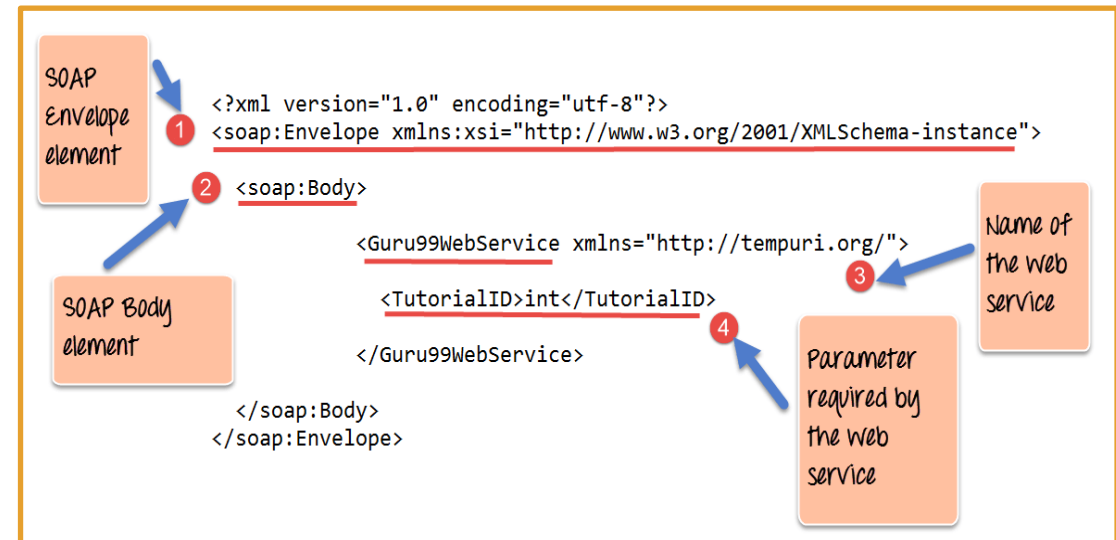


# SOAP Message Elements

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

- Envelope – This element is mandatory and identifies the message as a **SOAP** message. 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.

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](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.
- The **Envelope** may only contain one header element.
- The **SOAP Envelope** must have exactly one **Body** element.
- The **SOAP Envelope Header** must be the first child of the **Envelope**.
- The **SOAP Envelope** changes when versions change. A **Fault** is generated when **Envelope** versions are mismatched.

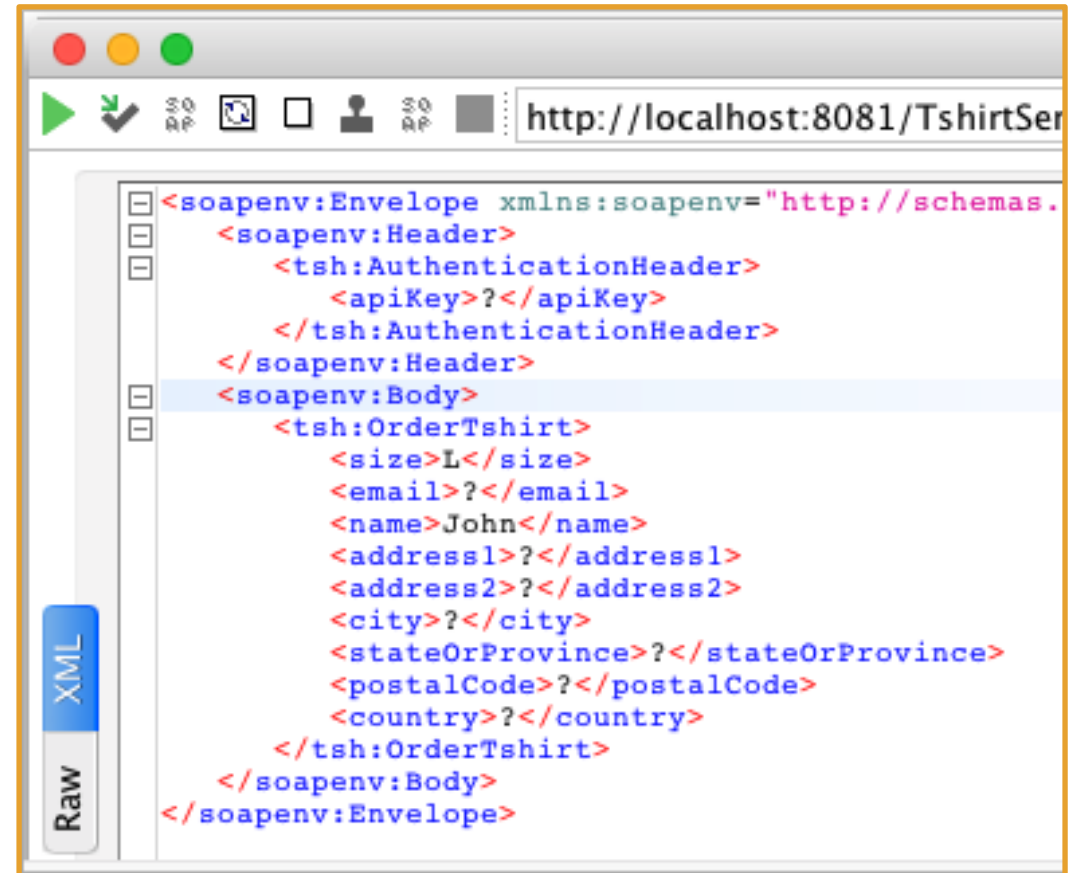
```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</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>Áke 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

[https://www.ibm.com/support/knowledgecenter/SSGMCP\\_5.4.0/fundamentals/web-services/dfhws\\_header.html](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.



The screenshot shows a web browser window with the address bar displaying `http://localhost:8081/TshirtSer`. The main content area displays an XML document representing a SOAP message. The XML structure is as follows:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <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>
```

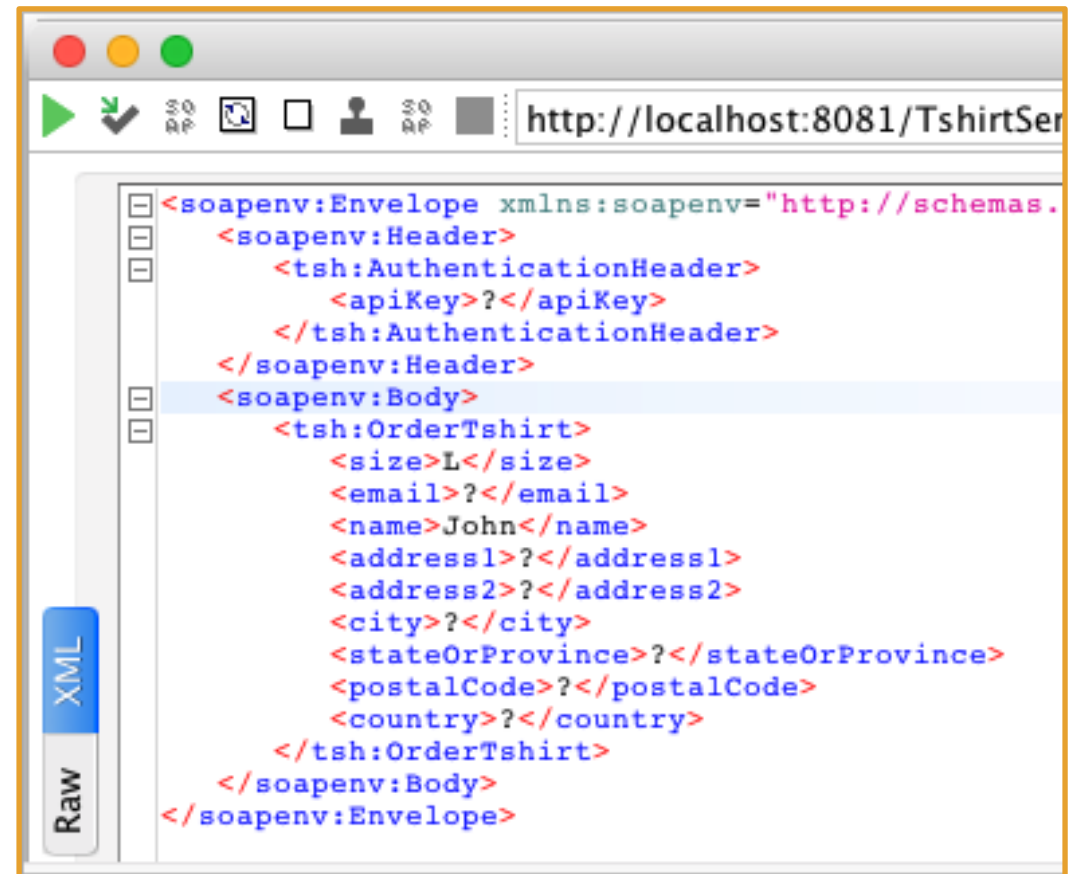
On the left side of the XML viewer, there are two tabs: "XML" (which is selected and highlighted in blue) and "Raw".

# SOAP Message - Header

[https://www.ibm.com/support/knowledgecenter/SSGMCP\\_5.4.0/fundamentals/web-services/dfhws\\_header.html](https://www.ibm.com/support/knowledgecenter/SSGMCP_5.4.0/fundamentals/web-services/dfhws_header.html)

**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..



The screenshot shows a web browser window with the address bar displaying `http://localhost:8081/TshirtSer`. The main content area displays an XML document representing a SOAP message. The XML structure is as follows:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <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>
```

On the left side of the XML viewer, there are two tabs: "XML" (which is selected and highlighted in blue) and "Raw".

# 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 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 Star Wars character data., etc

**WCF** is designed to offer a manageable, easy approach to creating **Web services** and **Web service** clients.

# WCF Features

<https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf#features-of-wcf>

---

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 and other formats.
Data Contracts	You can use C# classes to represent data and the .NET Framework automatically creates the metadata that allows clients to comply with the data types you designed.
Durable Messages	Never loose messages because they 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

[https://www.ibm.com/support/knowledgecenter/SSMQ79\\_9.5.1/com.ibm.epl.pg.doc/topics/pegl\\_serv\\_overview.html](https://www.ibm.com/support/knowledgecenter/SSMQ79_9.5.1/com.ibm.epl.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](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)

