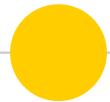


Implementing our first web service





Approaches to building web services

- There are 2 approaches to building web services :

1. Code first (Bottom Up) approach

Allows you to reuse your investment by exposing your existing application.

For example : Credit Card System is an existing Application with Proven business value. Competitive pressure is moving Credit Card System to expose some of this business functionality Like (Credit Card Number Validation) as web service. The implementation class **already exists**, all that is needed is to create a WSDL and expose the class as web service.

2. Contract First (Top Down) Approach

Is the correct way to build new web service from scratch.

This Approach starts with the WSDL (the contract) by defining operation message and so forth. Then you build the endpoint interface and finally the implementation class.

1. Code first (Bottom Up) approach

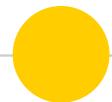
- In bottom-up development, the interface is generated based on an **existing implementation** thereby exposing its functionality.
- Bottom-up development can be a fast way of creating interfaces but gives you less control over the exact definition of the interface.

2. Contract First (Top Down) Approach

- In top-down or contract-first development, the service interface is the starting point for developing services.
- A framework is used to generate a code skeleton based on the service WSDLs and XSDs. The generated implementation can then be completed.
- This can be a good approach when there is no service implementation yet in place.

Code first (Bottom Up) approach





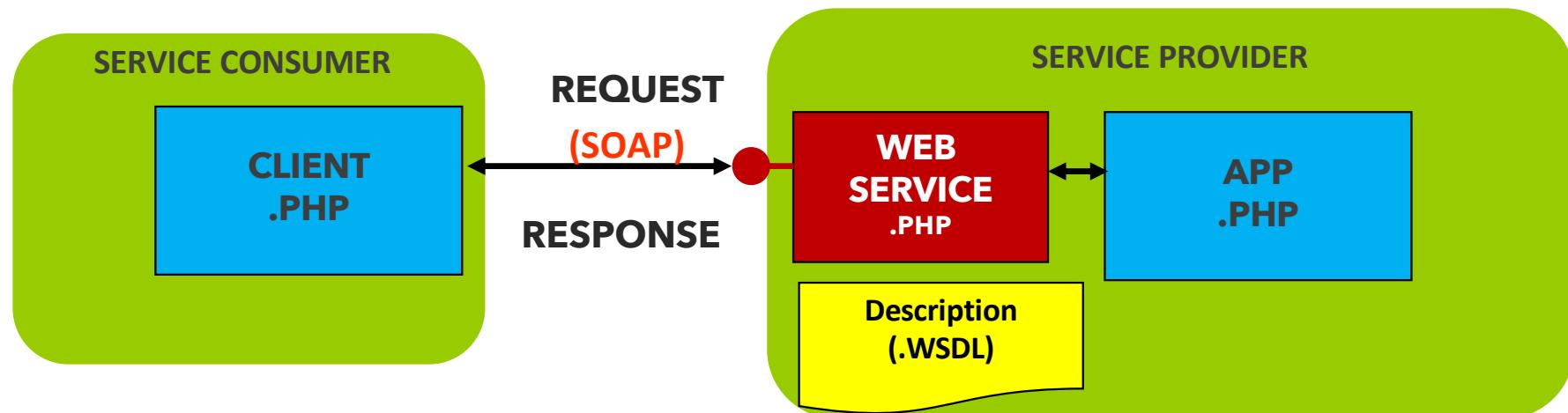
Implementing service

○ We have initially :

- App (.php)

○ We will implement :

- Web service: Webservice (.php)
- Client : Client (.php) or any other programming language.
- Contract : (.WSDL) will be automatically generated by SOAP API.



Example 1

- Create a web service for the following php function :

```
function SayHello($name){  
    $hi = "Hi ".$name;  
    return $hi;  
}
```

Step 1: write down the webservice.php

Step 2: Deploy the WS in an application server (WWW/HTDOCS folder)

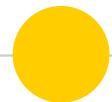


Step 1: write the code

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists files and folders under 'MYWEBSERVICES'. The 'webservice.php' file is selected in the list. The main area displays the PHP code for a web service. The code includes comments explaining the purpose of the script, the use of NuSOAP, and the definition of a 'SayHello' method. It also shows how to register this method with WSDL support and invoke it using the server's service function.

```
1 <?php
2 /**
3  * @Description: Server Side Web Service
4  * Bottom-up approach
5  * This Script creates a web service using NuSOAP php library.
6  * @Author: Neila BEN LAKHAL
7  * @Website: neila.benlakhal@github.io
8 */
9 // Pull in the NuSOAP code
10 require_once('lib/nusoap.php');
11 // Create the server instance
12 $server = new nusoap_server();
13 // Define the method as a PHP function
14 function SayHello($name){
15     $hi = "Hi ".$name;
16     return $hi;
17 // Initialize WSDL support
18 $server->configureWSDL('myname', 'http://www.mynamespace.com');
19 // Register the method to expose
20     $server->register('SayHello',
21         array('name' => 'xsd:string'), //input parameter
22         array('hi' => 'xsd:string'), //output
23         'http://www.mynamespace.com', //namespace
24         'http://www.mynamespace.com#SayHello', //soapaction
25         "say Hi to the caller");// documentation
26     );
27 // Use the request to (try) to invoke the service
28 $server->service(file_get_contents("php://input"));
29 ?>
```

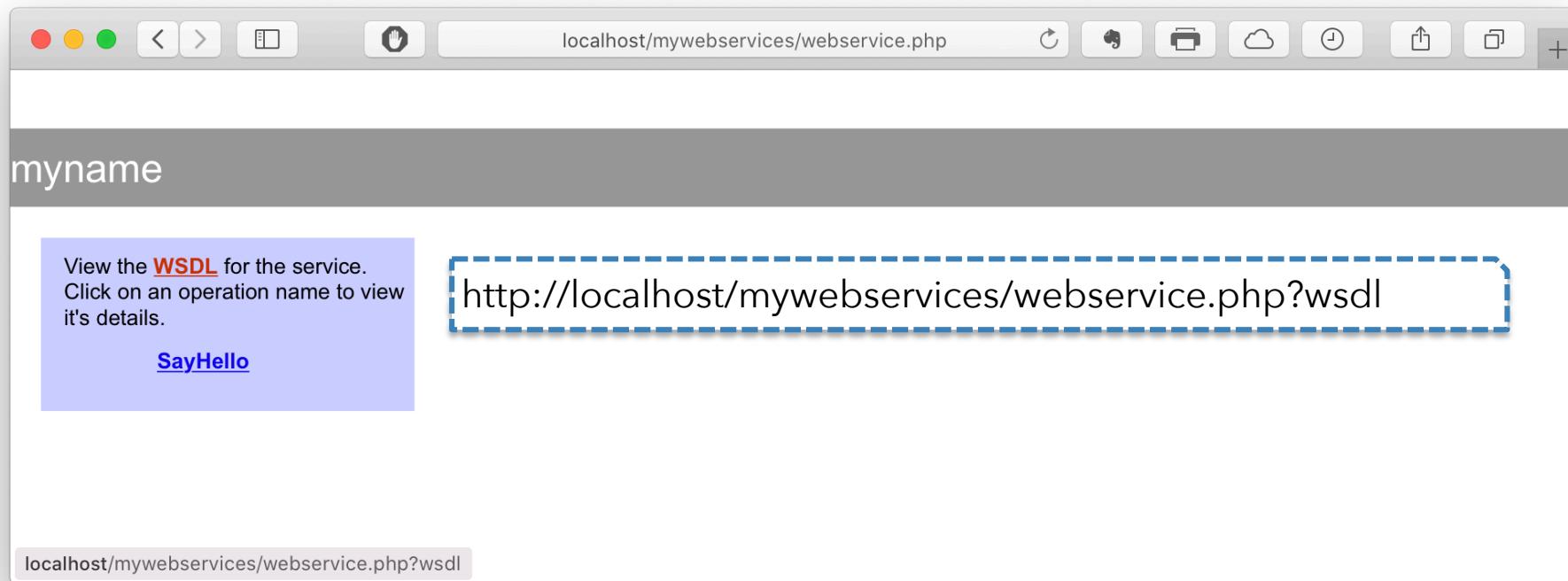
Ln 21, Col 47 Spaces: 2 UTF-8 LF PHP ⚙



Step 2: Deploy

- Add your **webservice** file to **WWW** or **htdocs** folder
- Access the web service :

<http://localhost/mywebservices/webservice.php>



Web service Contract

http://localhost/mywebservices/webservice.php?wsdl

```

<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://www.mynamespace.com" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/">
    <targetNamespace="http://www.mynamespace.com"/>
    <types>
        <xsd:schema targetNamespace="http://www.mynamespace.com">
            <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
            <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
        </xsd:schema>
    </types>
    <message name="SayHelloRequest">
        <part name="name" type="xsd:string"/>
    </message>
    <message name="SayHelloResponse">
        <part name="hi" type="xsd:string"/>
    </message>
    <portType name="mynamePortType">
        <operation name="SayHello">
            <input message="tns:SayHelloRequest"/>
            <output message="tns:SayHelloResponse"/>
        </operation>
    </portType>
    <binding name="mynameBinding" type="tns:mynamePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="SayHello">
            <soap:operation soapAction="http://www.mynamespace.com#SayHello" style="say Hi to the caller"/>
            <input>
                <soap:body use="encoded" namespace="http://www.mynamespace.com" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded" namespace="http://www.mynamespace.com" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>
    </binding>
    <service name="myname">
        <port name="mynamePort" binding="tns:mynameBinding">
            <soap:address location="http://localhost/mywebservices/webservice.php"/>
        </port>
    </service>
</definitions>

```



Web service Test (SOAPUI)

SoapUI 5.6.0

Empty SOAP REST Import Save All Forum Trial Preferences Proxy Endpoint Explorer Search Forum Online Help

Navigator Projects NumberConversion NumberConversion webservice mynameBinding SayHello Request 1

Request Properties

Property	Value
Name	Request 1
Description	
Message Size	449
Encoding	UTF-8
Endpoint	http://localhost/mywebservices/webservice.php
Timeout	
Bind Address	
Follow Redir...	true
Username	
Password	
Domain	
Authenticati...	No Authoriz...
WSS-Passwo...	
WSS TimeTo...	
SSL Keystore	
Skip SOAP A...	false
Enable MTOM	false
Force MTOM	false
Inline Respo...	false
Expand MT...	false
Disable mul...	true
Encode Attrib...	false

Raw XML

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:myn="http://mynamebinding.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <myn:SayHello soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding;">
      <name xsi:type="xsd:string">You</name>
    </myn:SayHello>
  </soapenv:Body>
</soapenv:Envelope>
```

Raw XML

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
  <SOAP-ENV:Body>
    <hi xsi:type="xsd:string">Hi You</hi>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A... Header... Attachmen... W... WS... JMS Hea... JMS Proper... Headers (9) Attachments (0) SSL Info WSS (0) JMS (0) response time: 143ms (434 bytes) 8 : 20

Properties SoapUI log http log jetty log error log wsrm log memory log >



Web service test (client)

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with icons for file operations, search, and Docker. The main area has two tabs: "webservice.php" (active) and "client.php". The "webservice.php" tab contains a PHP script for creating a client using NuSOAP. The "client.php" tab is currently closed.

```
<?php
/*
@Description: Web service client
This Script creates a client using NuSOAP php library.
@Author: Neila BEN LAKHAL
@Website: neila.benlakhal@github.io
*/
// Pull in the NuSOAP code
require_once('lib/nusoap.php');
$error = '';
$result = array();
$guest= " Web";
$wsdl = "http://localhost/mywebservices/webservice.php?wsdl";
if(!$error){
    //create client object
    $client = new nusoap_client($wsdl, true);
    $err = $client->getError();
    if ($err) {
        echo '<h2>Constructor error</h2>' . $err;
        // At this point, you know the call that follows will fail
        exit();
    }
    try {
        // Call the SOAP method
        $result = $client->call('SayHello', array($guest));
        // Display the result
        echo "<h2>Result</h2/>";
        print_r($result);
    } catch (Exception $e) {
        echo 'Caught exception: ', $e->getMessage(), "\n";
    }
}
// Display the request and response (SOAP messages)
echo '<h2>Request</h2>';
echo '<pre>' . htmlspecialchars($client->request, ENT_QUOTES) . '</pre>';
echo '<h2>Response</h2>';
echo '<pre>' . htmlspecialchars($client->response, ENT_QUOTES) . '</pre>';
// Display the debug messages
echo '<h2>Debug</h2>';
echo '<pre>' . htmlspecialchars($client->debug_str, ENT_QUOTES) . '</pre>';
?>
```

Bottom status bar: Ln 27, Col 40 Tab Size: 4 UTF-8 LF PHP ⚙

Example 2

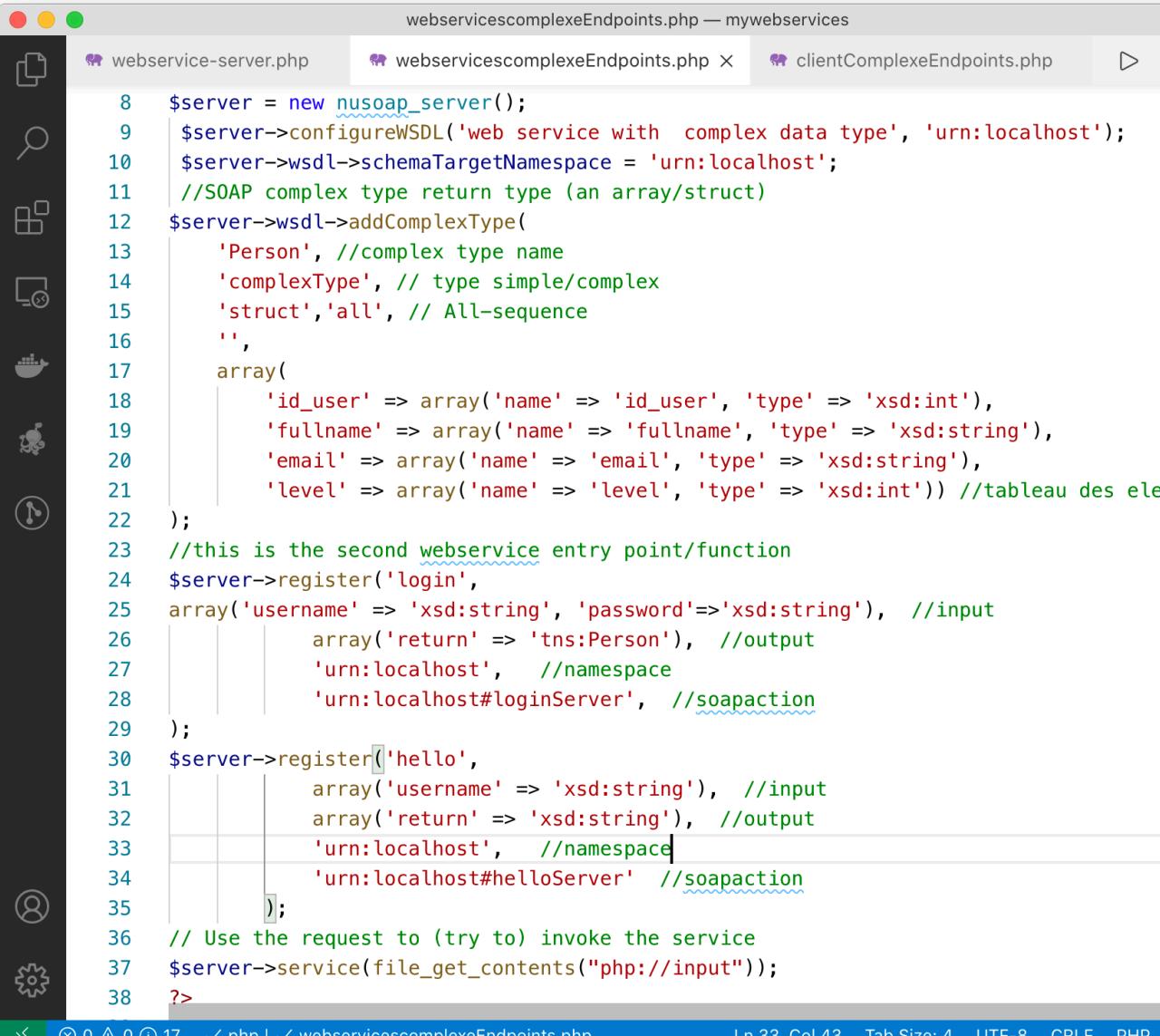
- Web service with complex data type (.xsd)

```
//(first endpoint)
function hello($username) {
return 'Howdy, '.$username.'!';
}
```

```
//(second endpoint)
// function with complex XML (data type)
function login($username, $password){
return array('id_user'=>1,
'fullname'=>'WEB',
'email'=>'web@soa.com', 'level'=>100);
}
```

```
//(first endpoint)
function hello($username) {
return 'Howdy, '.$username.'!';
}
```

```
//(second endpoint)
// function with complex XML (data type)
function login($username, $password){
return array('id_user'=>1,
'fullname'=>'WEB',
'email'=>'web@soa.com', 'level'=>100);
}
```



The screenshot shows a code editor interface with three tabs at the top:

- webservice-server.php
- webservicescomplexEndpoints.php** (highlighted in blue)
- clientComplexEndpoints.php

The code in the webservicescomplexEndpoints.php tab is as follows:

```

8 $server = new nusoap_server();
9 $server->configureWSDL('web service with complex data type', 'urn:localhost');
10 $server->wsdl->schemaTargetNamespace = 'urn:localhost';
11 //SOAP complex type return type (an array/struct)
12 $server->wsdl->addComplexType(
13     'Person', //complex type name
14     'complexType', // type simple/complex
15     'struct', 'all', // All-sequence
16     '',
17     array(
18         'id_user' => array('name' => 'id_user', 'type' => 'xsd:int'),
19         'fullname' => array('name' => 'fullname', 'type' => 'xsd:string'),
20         'email' => array('name' => 'email', 'type' => 'xsd:string'),
21         'level' => array('name' => 'level', 'type' => 'xsd:int')) //tableau des ele
22 );
23 //this is the second webservice entry point/function
24 $server->register('login',
25     array('username' => 'xsd:string', 'password'=>'xsd:string'), //input
26     array('return' => 'tns:Person'), //output
27     'urn:localhost', //namespace
28     'urn:localhost#loginServer', //soapaction
29 );
30 $server->register('hello',
31     array('username' => 'xsd:string'), //input
32     array('return' => 'xsd:string'), //output
33     'urn:localhost', //namespace
34     'urn:localhost#helloServer' //soapaction
35 );
36 // Use the request to (try to) invoke the service
37 $server->service(file_get_contents("php://input"));
38 ?>

```

The status bar at the bottom of the editor shows:

- Line 33, Column 43
- Tab Size: 4
- UTF-8
- CRLF
- PHP

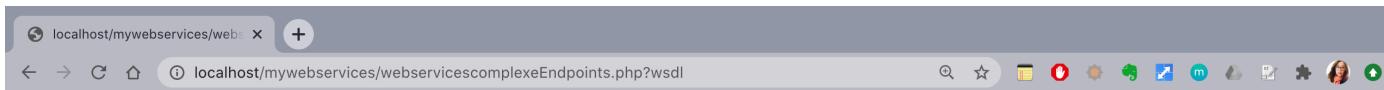
AddComplexType method

The screenshot shows a GitLab repository page for the project 'php-client'. The 'Repository' section is selected in the sidebar. The main content area displays the source code of the 'AddComplexType' method. The code is annotated with PHPDoc comments and uses reflection-style assignment for parameters.

```
* @param name
* @param typeClass (complexType|simpleType|attribute)
* @param phpType: currently supported are array and struct (php assoc array)
* @param compositor (all|sequence|choice)
* @param restrictionBase namespace:name (http://schemas.xmlsoap.org/soap/encoding/:Array)
* @param elements = array ( name = array(name=>'',type=>'') )
* @param attrs = array(
*     array(
*         'ref' => "http://schemas.xmlsoap.org/soap/encoding/:arrayType",
*         "http://schemas.xmlsoap.org/wsdl/:arrayType" => "string[]"
*     )
* )
* @param arrayType: namespace:name (http://www.w3.org/2001/XMLSchema:string)
* @access public
* @see getTypeDef
*/
function addComplexType($name,$typeClass='complexType',$phpType='array',$compositor='', $restrictionBase='', $elements=array(),
    $this->complexTypes[$name] = array(
        'name' => $name,
        'typeClass' => $typeClass,
        'phpType' => $phpType,
        'compositor'=> $compositor,
        'restrictionBase' => $restrictionBase,
        'elements' => $elements,
        'attrs' => $attrs,
        'arrayType' => $arrayType
    );
    $this->xdebug("addComplexType $name:");
    $this->appendDebug($this->varDump($this->complexTypes[$name]));
}
```

<https://www.a-company.it/gitlab/timestampservice.clients/php-client/blob/2ce35014bac0cc9c47192d137cda61fd9ddac31c/nusoap/class.xmlschema.php>

Added complex type in WSDL contract



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="urn:localhost"  
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"  
    xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:localhost">  
  ▼<types>  
    ▼<xsd:schema targetNamespace="urn:localhost">  
      <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />  
      <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />  
      ▼<xsd:complexType name="Person">  
        ▼<xsd:all>  
          <xsd:element name="id_user" type="xsd:int"/>  
          <xsd:element name="fullname" type="xsd:string"/>  
          <xsd:element name="email" type="xsd:string"/>  
          <xsd:element name="level" type="xsd:int"/>  
        </xsd:all>  
      </xsd:complexType>  
    </xsd:schema>  
  </types>
```



Exemple 3

Web service call (Web App)

The screenshot shows a web browser window with the URL `localhost/mywebservices/webservice-client.php`. The page title is "Books Store SOAP Web Service". A red circle highlights the input field labeled "Enter ISBN" and the button "Fetch Book Information". Below this, a table displays "Book Information" for the book "C++ By Example" by John, priced at 500, with ISBN PR-123-A1 and category Programming. A large red oval encloses the entire row of the table. At the bottom, there is an "Add New Book" form with five input fields: "Enter Title", "Enter Author", "Enter Price", "Enter ISBN", and "Enter Category", followed by a "Add New Book" button.

Title	Author	Price	ISBN	Category
C++ By Example	John	500	PR-123-A1	Programming



Exemple 3

- Web service :
- <http://localhost/mywebservices/webservice-server.php>

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost/mywebservices/webservice-server.php
- Title Bar:** booksServer
- Content Area:** A purple box contains the following text:

View the [WSDL](#) for the service. Click on an operation name to view its details.

[fetchBookData](#)
[insertBook](#)
- Bottom Navigation:** A yellow bar with the text "NEILA.BENLAKHALO" on the left and navigation icons (< 169 >) on the right.

More examples

- Currently, platforms like Paypal, ebay.. are offering access to their API and support multiple SOAP-based web services:
- NVP/SOAP API (PAYPAL)
<https://developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAPIArchitecture/>
- EBAY
<https://developer.ebay.com/devzone/shopping/docs/CallRef/GetItemStatus.html#Samples>
- FLIGHT STAT
<https://developer.flightstats.com/api-docs>

<https://developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAPIArchitecture/>



PayPal SOAP API Basics - PayP

developer.paypal.com/docs/nvp-soap-... Log into Dashboard

PayPal Developer Docs APIs Tools Support PayPal.com Search

NVP Basics
SOAP Basics
Credentials
SDKs
NVP Operations
SOAP Operations
Adaptive Platform API
Reference

NVP/SOAP API

PayPal SOAP API Basics

The PayPal SOAP API is based on open standards known collectively as web services, which include the Simple Object Access Protocol (SOAP), Web Services Definition Language (WSDL), and the XML Schema Definition language (XSD). A wide range of development tools on a variety of platforms support web services.

Like many web services, PayPal SOAP is a combination of client-side and server-side schemas, hardware and software servers, and core services.

PayPal SOAP High-level Diagram

The diagram illustrates the architecture of the PayPal SOAP API. It shows a 'SOAP API client application' on the left sending a 'Signed SOAP request over HTTPS' to the 'PayPal API Servers'. The servers contain two environments: 'Test' (with URLs like 'api.sandbox.paypal.com/2.0/' and 'api-3t.sandbox.paypal.com/2.0/') and 'Production' (with URLs like 'api.paypal.com/2.0/' and 'api-3t.paypal.com/2.0/'). The servers return a 'SOAP response over HTTPS' to the client. A dashed arrow points from the servers to the 'PayPal Main Interactive Site', indicating 'Joint processing of requests in service and interactive site'. Below the main site is a box labeled 'PayPal Core Services (Payment Engine Business Logic)'.

PayPal SOAP API Basics - PayP

developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAP... Log into Dashboard

PayPal Developer Docs APIs Tools Support PayPal.com Search

NVP Basics
SOAP Basics
Credentials
SDKs
NVP Operations
SOAP Operations
Adaptive Platform API
Reference

PayPal WSDL/XSD Schema Definitions

The PayPal Web Services schema and its underlying eBay Business Language (eBL) base and core components are required for developing applications with the PayPal Web Services API. The following are the locations of the WSDL and XSD files.

Location of PayPal WSDL and XSD Files

Development and Testing with the PayPal sandbox

PayPal Schema	https://www.sandbox.paypal.com/wsdl/PayPalSvc.wsdl
eBL Base Components and Component Types	https://www.sandbox.paypal.com/wsdl/eBLBaseComponents.xsd https://www.sandbox.paypal.com/wsdl/CoreComponentTypes.xsd

Production with the Live PayPal API

PayPal Schema	https://www.paypal.com/wsdl/PayPalSvc.wsdl
eBL Base Components and Component Types	https://www.paypal.com/wsdl/eBLBaseComponents.xsd https://www.paypal.com/wsdl/CoreComponentTypes.xsd

PayPal SOAP API Definitions

The PayPal SOAP API comprises individual API definitions for specific business functions. As a foundation, the API relies on eBay Business Language (eBL) base and core components. The core eBL structures `AbstractRequestType` and `AbstractResponseType` are the basis of the SOAP request and response of each PayPal API. `AbstractResponseType` is also the framework for error messages common across all PayPal APIs.



PAYPAL Service Access



<https://www.paypalobjects.com/wsdl/paypalsvc.wsdl>

The screenshot shows a web browser window with the title "PayPal SOAP API Basics - PayP". The URL in the address bar is "developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAP...". The page content is titled "PayPal WSDL/XSD Schema Definitions". It explains that the PayPal Web Services schema and its underlying eBay Business Language (eBL) base and core components are required for developing applications with the PayPal Web Services API. It provides links to the locations of the WSDL and XSD files for both the sandbox and live environments.

PayPal WSDL/XSD Schema Definitions

The PayPal Web Services schema and its underlying eBay Business Language (eBL) base and core components are required for developing applications with the PayPal Web Services API. The following are the locations of the WSDL and XSD files.

Location of PayPal WSDL and XSD Files

Development and Testing with the PayPal sandbox

PayPal Schema	https://www.sandbox.paypal.com/wsdl/PayPalSvc.wsdl
eBL Base Components and Component Types	https://www.sandbox.paypal.com/wsdl/eBLBaseComponents.xsd https://www.sandbox.paypal.com/wsdl/CoreComponentTypes.xsd

Production with the Live PayPal API

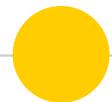
PayPal Schema	https://www.paypal.com/wsdl/PayPalSvc.wsdl
eBL Base Components and Component Types	https://www.paypal.com/wsdl/eBLBaseComponents.xsd https://www.paypal.com/wsdl/CoreComponentTypes.xsd



PayPal API SOAP Request



```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
3.    <SOAP-ENV:Header>
4.      <RequesterCredentials xmlns="urn:ebay:api:PayPalAPI">
5.        <Credentials xmlns="urn:ebay:apis:eBLBaseComponents">
6.          <Username><API-Username></Username>
7.          <Password><API-Password></Password>
8.          <Signature/>
9.          <Subject/>
10.         <Credentials>
11.       </RequesterCredentials>
12.     </SOAP-ENV:Header>
13.     <SOAP-ENV:Body>
14.       <<Specific-API-Name>-Req xmlns="urn:ebay:api:PayPalAPI">
15.         <<Specific-API-Name>-Request>
16.           <Version xmlns="urn:ebay:apis:eBLBaseComponents"><API-Version>
</Version>
17.           <<Required-Or-Optional-Fields> xs:type="<Type>"><Data></Required-
Or-Optional-Fields>>
18.         </<Specific-API-Name>-Request>
19.       </<Specific-API-Name>-Req>
20.     </SOAP-ENV:Body>
21.   </SOAP-ENV:Envelope>
```



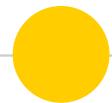
FLIGHT STAT API

⌚ <https://developer.flightstats.com/api-docs>

The screenshot shows a browser window displaying the "Flex API Quick Reference" for the Flight Stats API. The URL in the address bar is <https://developer.flightstats.com/api-docs>. The page is organized into sections for different APIs:

- Aircraft**: Provides extensive information on specific aircraft. It includes a table of entry points and their descriptions, such as `aircraft/` for aircraft details and `availableFields/` for user fields.
- Airlines**: Provides basic reference information about one or more airlines. It includes a table of entry points and their descriptions, such as `active/` for active airlines and `iata/` for airlines by IATA code.
- Airports**: (Partially visible)

Each section includes links to the Base URI, Version, WSDL, XSD, and JSON specifications.



EBAY API



○ <https://developer.ebay.com/devzone/shopping/docs/CallRef/GetItemStatus.html#Samples>

Sample: Basic Call
Retrieves status for one or more specified item listings.

Input

GetItemStatus takes an array of **ItemID** values. The list can include one or more **ItemID** values.

This example retrieves the status for all item listings in the specified list of **ItemID** values.

URL format. See also the [non-wrapped](#) version of this URL. For results in a format other than XML, specify a different value for [responseencoding](#).

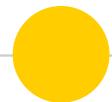
```
https://open.api.ebay.com/shopping?
    callname=GetItemStatus&
    responseencoding=XML&
    appid=yourAppID&
    siteid=0&
    version=967&
    ItemID=180126682091,230139965209,300118131654,300118131865,300118132027
```

Here is the same input in XML format. Note that this does not include [standard values](#).

XML format. Also available is the [SOAP](#) equivalent.

```
<?xml version="1.0" encoding="utf-8"?>
<GetItemStatusRequest xmlns="urn:ebay:apis:eBLBaseComponents">
    <ItemID>180126682091</ItemID>
    <ItemID>230139965209</ItemID>
    <ItemID>300118131654</ItemID>
    <ItemID>300118131865</ItemID>
    <ItemID>300118132027</ItemID>
</GetItemStatusRequest>
```

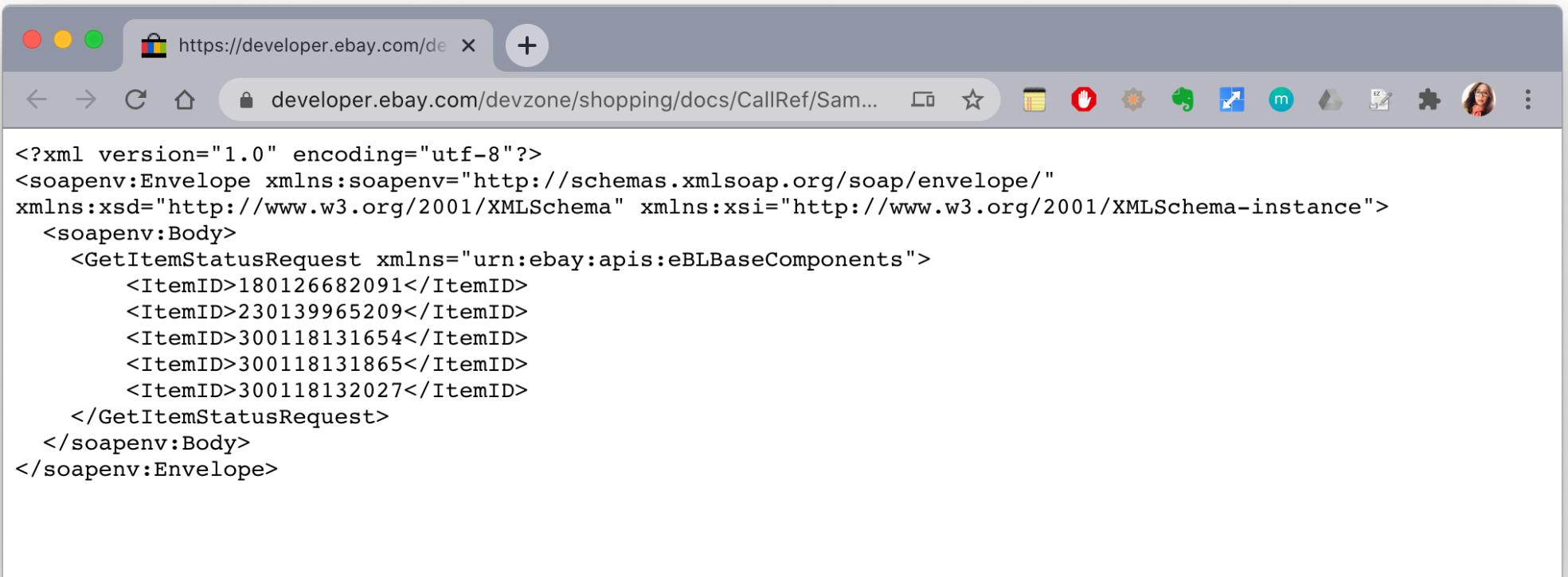
Output



EBAY API



① https://developer.ebay.com/devzone/shopping/docs/callref/samples/getitemstatus_basic_in_soap_soap.txt



The screenshot shows a web browser window with the URL https://developer.ebay.com/devzone/shopping/docs/callref/samples/getitemstatus_basic_in_soap_soap.txt. The page displays an XML code snippet for a GetItemStatusRequest. The XML is as follows:

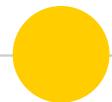
```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <GetItemStatusRequest xmlns="urn:ebay:apis:eBLBaseComponents">
      <ItemID>180126682091</ItemID>
      <ItemID>230139965209</ItemID>
      <ItemID>300118131654</ItemID>
      <ItemID>300118131865</ItemID>
      <ItemID>300118132027</ItemID>
    </GetItemStatusRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Contract first (Top-down) approach



Top-down approach

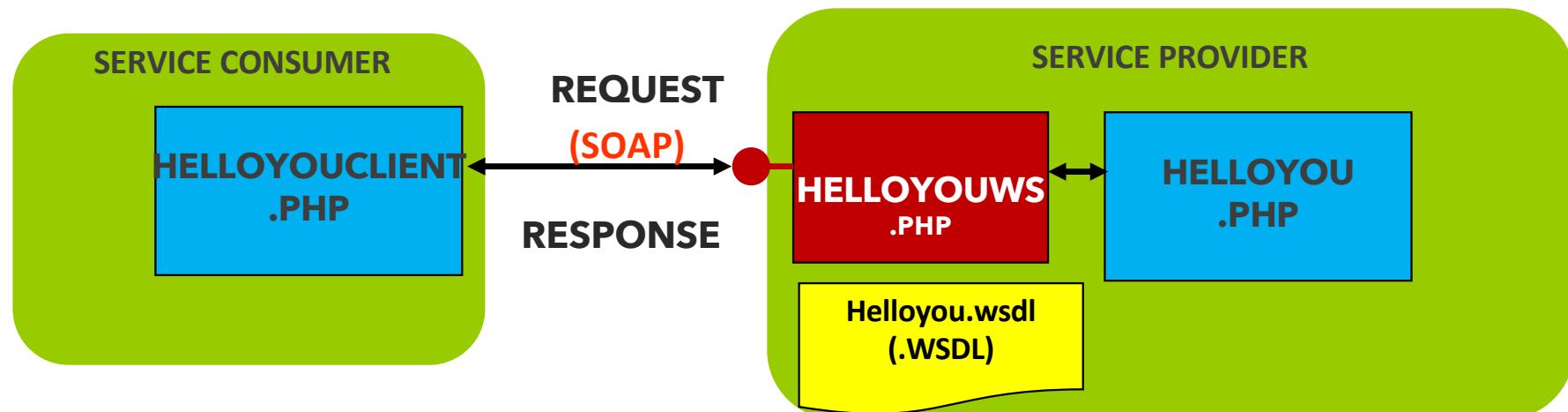
- Unlike the bottom-up approach used to migrate to SOA architecture (already implemented IS), where a service composition approach is followed, the top-down approach is used to implement a business process (BP) through a decomposition of the BP and the identification of the different services that compose it.



Implementing service

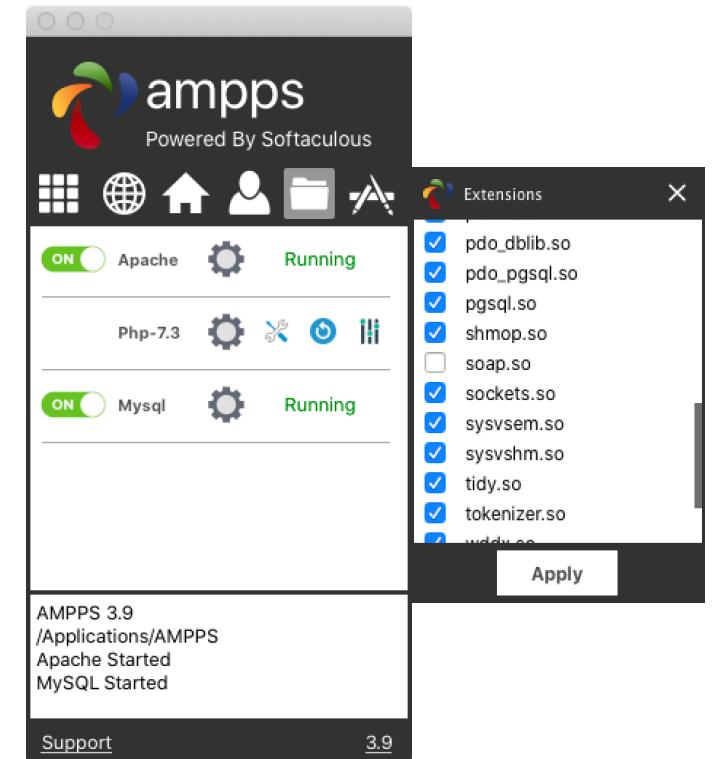
Steps :

- Step 1: write App code Helloyou (.php: our App)
- Step 2: write contract Helloyou.wsdl (contract)
- Step 3: write Web service HelloYouWS .php (.php)
- Step 4 : write Client HelloyouClient .php (.php) or any other programming language.



Implementing top-down service

- All programming languages have APIs to implement web services in Top-down approaches
- For php, we will be using : **soap.dll**, an extension to php
- In Ampps, you need to activate it.





Implementing top-down service

Step 1: write App code

The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane displays a file tree for a project named 'MYWEBSERVICES'. Inside 'topdown', there are four files: 'HelloYou.php' (selected), 'HelloYou.wsdl', 'HelloYouClient.php', and 'HelloYouService.php'. The main editor pane shows the contents of 'HelloYou.php':

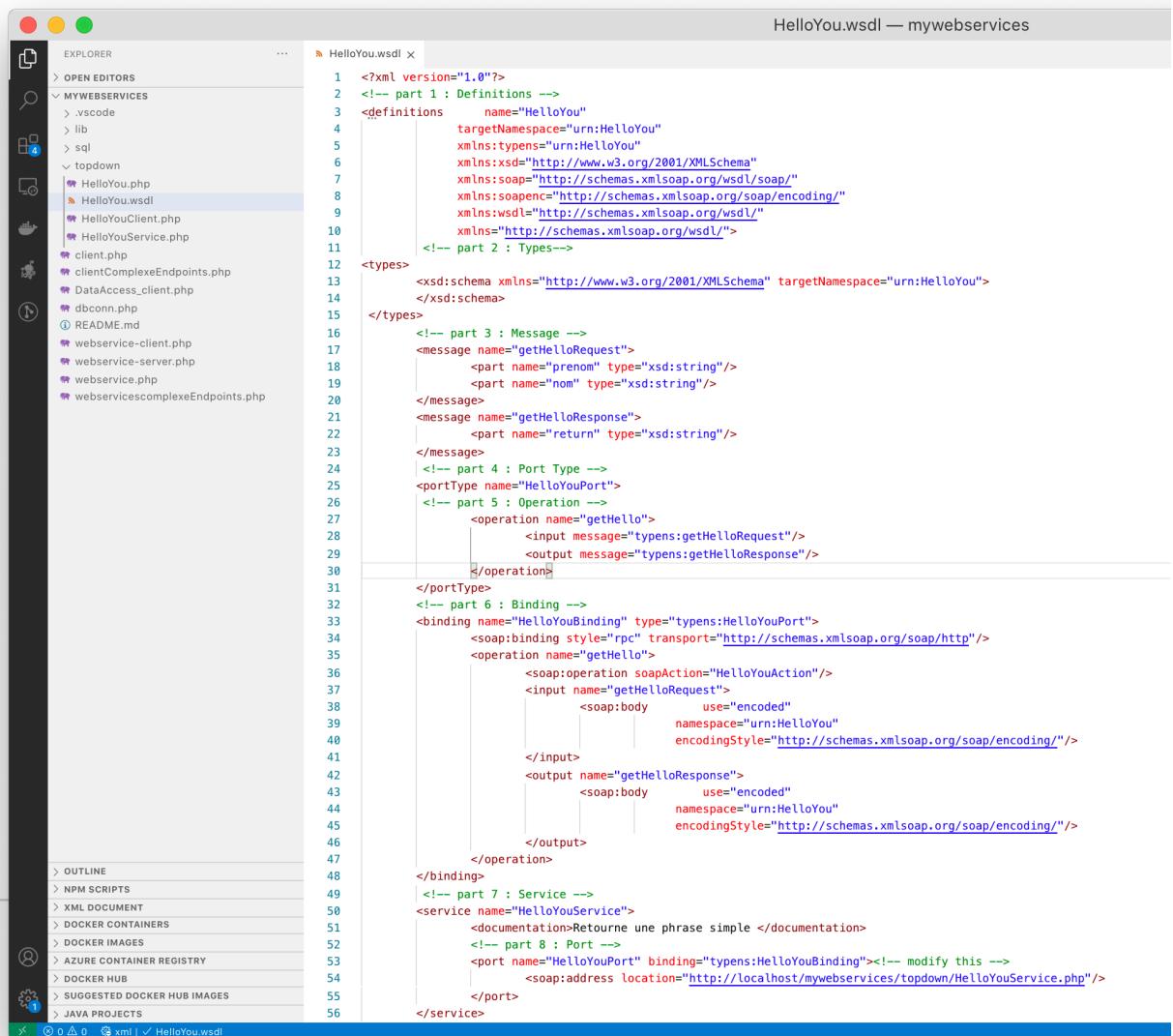
```
1 <?php
2 function getHello($prenom, $nom)
3 {
4     return 'Hello ' . $prenom . ' ' . $nom;
5 }
6
7 ?>
```

Step 2 : write (generate contract)

<http://localhost/mywebservices/topdown/HelloYou.wsdl>

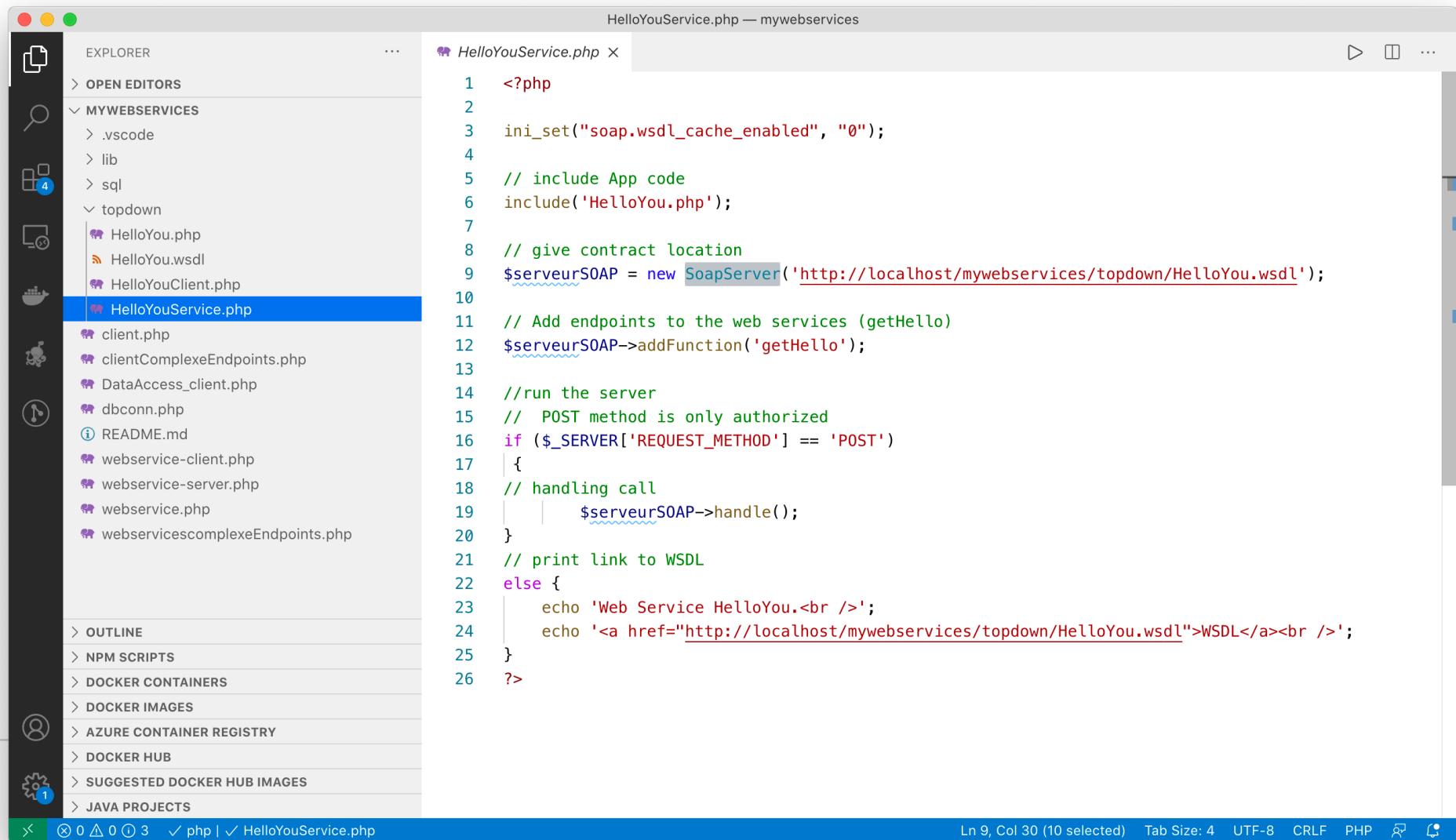
🟡 This approach requires .wsdl (contract file)

🟡 As we have App code in .php, we can use NUSOAP (bottom-up approach) to generate .wsdl file.



```
<?xml version="1.0"?>
<!-- part 1 : Definitions -->
<definitions name="HelloYou"
    targetNamespace="urn>HelloYou"
    xmlns:typens="urn>HelloYou"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
<!-- part 2 : Types-->
<types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn>HelloYou">
        </xsd:schema>
    </types>
    <!-- part 3 : Message -->
    <message name="getHelloRequest">
        <part name="prenom" type="xsd:string"/>
        <part name="nom" type="xsd:string"/>
    </message>
    <message name="getHelloResponse">
        <part name="return" type="xsd:string"/>
    </message>
    <!-- part 4 : Port Type -->
    <portType name="HelloYouPort">
        <!-- part 5 : Operation -->
        <operation name="getHello">
            <input message="typens:getHelloRequest"/>
            <output message="typens:getHelloResponse"/>
        </operation>
    </portType>
    <!-- part 6 : Binding -->
    <binding name="HelloYouBinding" type="typens>HelloYouPort">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getHello">
            <soap:operation soapAction="HelloYouAction"/>
            <input name="getHelloRequest">
                <soap:body use="encoded" namespace="urn>HelloYou" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            </input>
            <output name="getHelloResponse">
                <soap:body use="encoded" namespace="urn>HelloYou" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            </output>
        </operation>
    </binding>
    <!-- part 7 : Service -->
    <service name="HelloYouService">
        <documentation>Retourne une phrase simple </documentation>
        <!-- part 8 : Port -->
        <port name="HelloYouPort" binding="typens>HelloYouBinding"><!-- modify this -->
            <soap:address location="http://localhost/mywebservices/topdown>HelloYouService.php"/>
        </port>
    </service>
```

Step 3: write web service



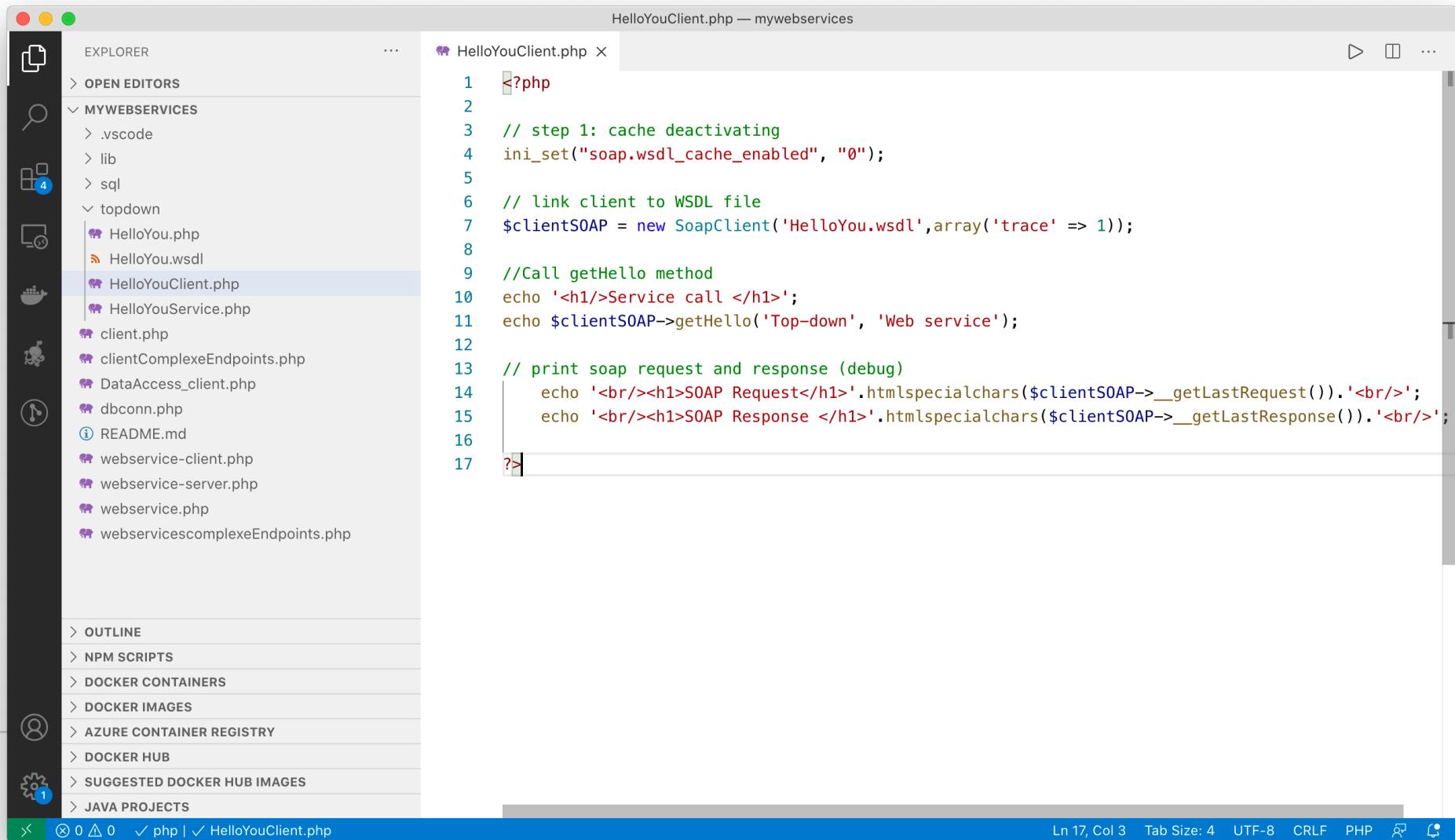
The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** HelloYouService.php — mywebservices
- Explorer View:** Shows the project structure under "MYWEBSERVICES". The file "HelloYouService.php" is selected and highlighted with a blue bar.
- Editor View:** Displays the PHP code for "HelloYouService.php". The code initializes a SoapServer and adds a function named "getHello". It also handles POST requests and prints a link to the WSDL document.

```
1 <?php
2
3 ini_set("soap.wsdl_cache_enabled", "0");
4
5 // include App code
6 include('HelloYou.php');
7
8 // give contract location
9 $serveurSOAP = new SoapServer('http://localhost/mywebservices/topdown/HelloYou.wsdl');
10
11 // Add endpoints to the web services (getHello)
12 $serveurSOAP->addFunction('getHello');
13
14 //run the server
15 // POST method is only authorized
16 if ($_SERVER['REQUEST_METHOD'] == 'POST')
17 {
18 // handling call
19 | $serveurSOAP->handle();
20 }
21 // print link to WSDL
22 else {
23 echo 'Web Service HelloYou.<br />';
24 echo '<a href="http://localhost/mywebservices/topdown/HelloYou.wsdl">WSDL</a><br />';
25 }
26 ?>
```

- Bottom Status Bar:** Shows file status (0 changes), line count (Ln 9), column count (Col 30), selected lines (10), tab size (4), encoding (UTF-8), line endings (CRLF), language (PHP), and other icons.

Step 4: Test service (write client)



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "MYWEBSERVICES".
- Editor View:** Displays the code for `HelloYouClient.php`. The code is a PHP script that includes a WSDL file, calls a service method, and prints SOAP requests and responses.
- Bottom Status Bar:** Shows file status (0 changes), file type (php), and other settings like tab size (4), encoding (UTF-8), and line endings (CRLF).
- Page Number:** 184 is visible in the bottom right corner.

```
<?php
// step 1: cache deactivating
ini_set("soap.wsdl_cache_enabled", "0");

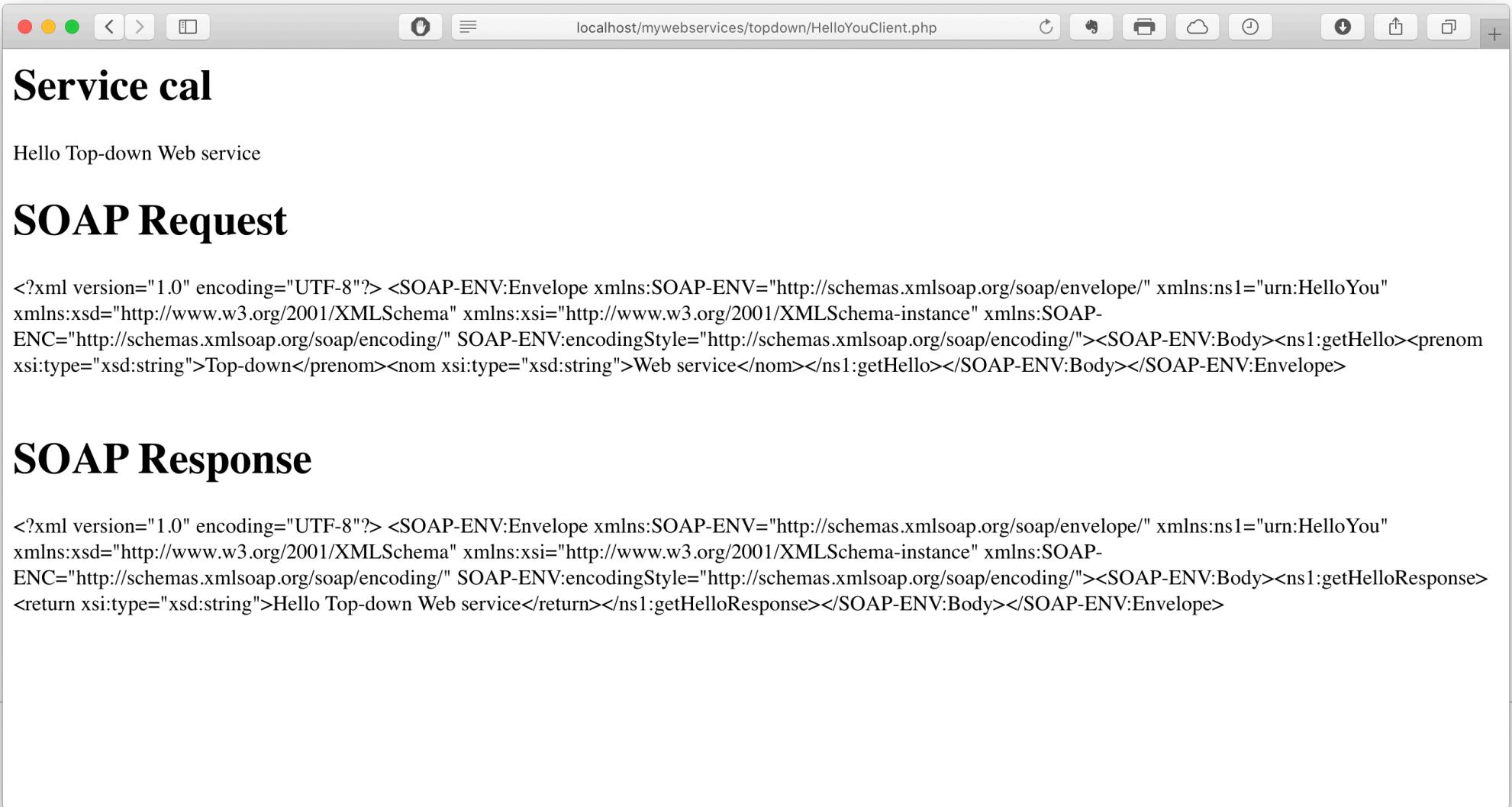
// link client to WSDL file
$clientSOAP = new SoapClient('HelloYou.wsdl',array('trace' => 1));

//Call getHello method
echo '<h1>Service call </h1>';
echo $clientSOAP->getHello('Top-down', 'Web service');

// print soap request and response (debug)
echo '<br/><h1>SOAP Request</h1>' . htmlspecialchars($clientSOAP->__getLastRequest()) . '<br/>';
echo '<br/><h1>SOAP Response </h1>' . htmlspecialchars($clientSOAP->__getLastResponse()) . '<br/>';

?>
```

Running Service client



The screenshot shows a web browser window with the following details:

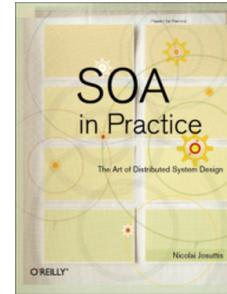
- Title Bar:** localhost/mywebservices/topdown/HelloYouClient.php
- Toolbar:** Standard browser icons for back, forward, search, and file operations.
- Content Area:**
 - Section Header:** Service cal
 - Text:** Hello Top-down Web service
 - Section Header:** SOAP Request
 - Text:** A large block of XML code representing a SOAP request message.
- Bottom Right:** Page number 185

SOAP Request XML:

```
<?xml version="1.0" encoding="UTF-8"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="urn:HelloYou" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-ENV:Body><ns1:getHello><prenom xsi:type="xsd:string">Top-down</prenom><nom xsi:type="xsd:string">Web service</nom></ns1:getHello></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

SOAP Response XML:

```
<?xml version="1.0" encoding="UTF-8"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="urn:HelloYou" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-ENV:Body><ns1:getHelloResponse><return xsi:type="xsd:string">Hello Top-down Web service</return></ns1:getHelloResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
```



[Nicolai M. Josuttis](#)

**SOA in Practice
The Art of Distributed System Design**

O'Reilly

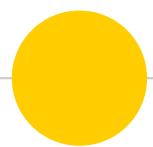
ISBN-10: 0-596-52955-4
ISBN-13: 978-0-596-52955-0

Bottom-up vs. Top-down

- In the *top-down* approach, you decompose a problem, system, or process into smaller chunks until you reach the level of (basic) services.
- In the *bottom-up* approach, you build business processes by composing services into more general chunks.
- All experts agree that in practice, a pure application of either of these approaches does not work. Of course, you can design your processes from the top down, which will help you to understand what is needed and what might be a "natural" separation of activities. However, ignoring existing realities might result in high costs compared with an approach that also takes the existing bottom layer into account. On the other hand, designing business processes from the bottom up introduces the danger of proposing technical details and constraints to very high levels, making the processes inflexible and unintuitive.
- [BloombergSchmelzer06] puts it as follows:

Your approach to SOA should be both top-down (through process decomposition) and bottom-up (exposing existing functionality as Services and composing them into processes). If you take only a top-down approach, you're likely to recommend building Services that are technically difficult or complex to implement. Taking solely a bottom-up approach can yield Services you don't need or, even worse, Services that don't fulfil the requirements of the business.

- So, in practice, a mixture of both approaches is appropriate. (There are many different names for such a mixed approach, such as "middle-out," "middle-ground," "meet in the middle," or just "agile.")
- Usually, your aim should be to fulfil a real business need. That is, a new or modified business process or a new functionality should come into play. However, you should also respect what you already have (adapting IT reality).



WSDL

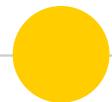
Web Service Description Language

Web Service Description Language

- Initiative of Ariba, IBM and Microsoft
- Standard by W3C

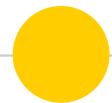
WSDL 1.1, (2001) : <http://www.w3.org/TR/wsdl>

WSDL 2.0 (2007) : <http://www.w3.org/TR/wsdl20/>



WSDL ?

- Is an **XML-based** specification **schema** for describing the **public** interface of a Web service.
- It is through this service description that the **service provider** communicates all the specifications for invoking a particular WS to the **service requestor**.
- WSDL is used to describe precisely:
 - **What** a service does, i.e., the **operations** the service provides,
 - **Where** it resides, i.e., details of the **protocol-specific address**, e.g., a URL, and
 - **How** to invoke it, i.e., details of the **data formats** and **protocols** necessary to access the service's operations.



WSDL content 1/2

● The WSDL specification can be conveniently divided into 2 parts. This enables each part to be defined separately and independently and reused by other parts :

1. Service interface definition (abstract interface)

- This contains all the **operations** supported by the service, the operation **parameters**, and abstract **data types**.
- ## 2. Service implementation (concrete implementation)
- This part binds the abstract interface to a concrete **network address**, to a **specific protocol**, and to concrete **data structures**.



WSDL content 2/2

● This public interface includes **operational** description of a Web service that encompasses:

- All publicly available **operations**,
- The XML **message** protocols supported by the Web service,
- **Data type** information for messages,
- **Binding** information about the specific **transport protocol** to be used, and
- **Address** information for **locating** the Web service.



WSDL structure

- WSDL is layered on top of the **XML schema**. It is stored in **.wsdl** file.
- **<definitions>** is the root element.
- Abstract interface describes ,messages, operations, and port types in a platform and language independent manner using **<types>**, **<message>**, **<part>**, **<portType>**, and **<operation>** elements.
- Concrete implementation part of WSDL contains the elements **<binding>** **<port>**, and **<service>** and describes how a particular service interface is implemented by a given service provider.

```
<definitions>
  <types>... </types>
  <message>... </message>
  <portType>...</portType>
  <binding>...</binding>
  <service>...>/service>
</definitions>
```

Several WSDL documents



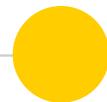
- Google Ads Platform
 - <https://developers.google.com/adwords/api/docs/guides/call-structure>
 - <https://adwords.google.com/api/adwords/cm/v201809/CampaignService?wsdl>
- eBay
 - <http://developer.ebay.com>
 - <http://developer.ebay.com/webservices/latest/ebaySvc.wsdl>
- PayPAL
 - <https://www.paypalobjects.com/wsdl/paypalsvc.wsdl>
- National Digital Forecast Database (NDFD)
 - <http://www.nws.noaa.gov/xml/>
 - <http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl>
- UPS Developer Kit APIs
 - <https://www.ups.com/cd/en/help-center/technology-support/developer-resource-center/ups-developer-kit/about.page?>
- Flightaware
 - <http://flightxml.flightaware.com/soap/FlightXML3/wsdl>
 - <https://flightaware.com/commercial/flightxml/v3/documentation.rvt>

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!-- part 1 : Definitions -->
3 <definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
7   xmlns:tns="http://www.mynamespace.com"
8   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
9   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
10  xmlns="http://schemas.xmlsoap.org/wsdl/"
11  targetNamespace="http://www.mynamespace.com">
12 <!-- part 2 : Types-->
13 <types>
14   <xsd:schema targetNamespace="http://www.mynamespace.com">
15     <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
16     <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
17   </xsd:schema>
18 </types>
19 <!-- part 3 : Message -->
20 <message name="SayHelloRequest">
21   <part name="name" type="xsd:string" />
22 </message>
23 <message name="SayHelloResponse">
24   <part name="hi" type="xsd:string" />
25 </message>
26 <!-- part 4 : Port Type -->
27 <portType name="mynamePortType">
28   <operation name="SayHello">
29     <input message="tns:SayHelloRequest" />
30     <output message="tns:SayHelloResponse" />
31   </operation>
32 </portType>
```

Abstract part

```
33 <!-- part 5 : Binding -->
34 <binding name="mynameBinding" type="tns:mynamePortType">
35   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
36   <operation name="SayHello">
37     <soap:operation soapAction="http://www.mynamespace.com#SayHello" style="rpc" />
38     <input>
39       <soap:body use="encoded"
40         namespace="http://www.mynamespace.com"
41         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
42     </input>
43     <output>
44       <soap:body use="encoded"
45         namespace="http://www.mynamespace.com"
46         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
47     </output>
48   </operation>
49 </binding>
50 <!-- part 6 : Service -->
51 <service name="myname">
52   <!-- part 7 : Port -->
53   <port name="mynamePort" binding="tns:mynameBinding">
54     <soap:address location="http://localhost/mywebservices/webservice.php" />
55   </port>
56 </service>
57 </definitions>
```

Concrete part

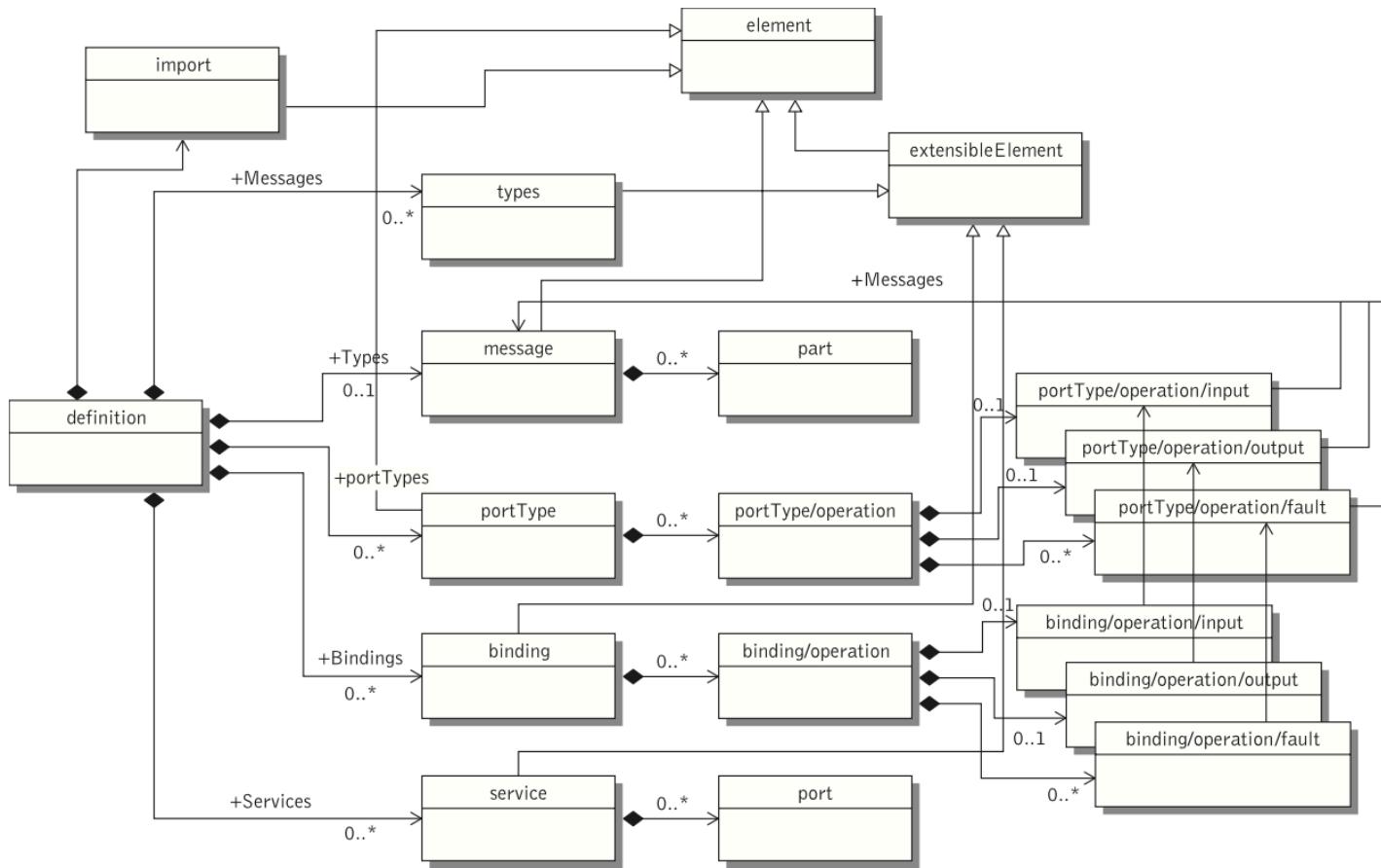


WSDL sample:

<http://www.dneonline.com/calculator.asmx?wsdl>

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"  
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/"  
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"  
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://tempuri.org/">\n    <wsdl:types>\n        <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">...</s:schema>\n    </wsdl:types>\n    <wsdl:message name="AddSoapIn">...</wsdl:message>\n    <wsdl:message name="AddSoapOut">...</wsdl:message>\n    <wsdl:message name="SubtractSoapIn">...</wsdl:message>\n    <wsdl:message name="SubtractSoapOut">...</wsdl:message>\n    <wsdl:message name="MultiplySoapIn">...</wsdl:message>\n    <wsdl:message name="MultiplySoapOut">...</wsdl:message>\n    <wsdl:message name="DivideSoapIn">...</wsdl:message>\n    <wsdl:message name="DivideSoapOut">...</wsdl:message>\n    <wsdl:portType name="CalculatorSoap">...</wsdl:portType>\n    <wsdl:binding name="CalculatorSoap" type="tns:CalculatorSoap">...</wsdl:binding>\n    <wsdl:binding name="CalculatorSoap12" type="tns:CalculatorSoap">...</wsdl:binding>\n    <wsdl:service name="Calculator">\n        <wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">...</wsdl:port>\n        <wsdl:port name="CalculatorSoap12" binding="tns:CalculatorSoap12">...</wsdl:port>\n    </wsdl:service>\n</wsdl:definitions>
```

WSDL : Overall structure meta-model expressed in UML



<definitions> element

```
<!-- part 1 : Definitions -->
<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  ...
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://www.mynamespace.com"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://www.mynamespace.com">
```

- The **<definitions>** element signals the start of a WSDL document (i.e., the parent for all other WSDL elements) and it contains all relevant, globally declared namespaces.
 - The first attribute in the **<definitions>** element is **name**, which is used to name the entire WSDL document. The **<definitions>** element also declares an attribute called **targetNamespace**, which identifies the logical namespace for elements defined within the WSDL document and characterizes the service.
 - Required element (1 instance)

<types> element

```
<!-- part 2 : Types-->
<types>
    <xsd:schema targetNamespace="http://www.mynamespace.com">
        <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
        <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
    </xsd:schema>
</types>
```

- **<types>...</types>** (0..1, only one is authorized)
- Contains all abstract data types that define a WS interface.
- Contains XML schemas or external references to XML schemas that describe the data type definitions.
- Uses the built-in primitive data types such as int, float, long, short, string, boolean, and so on.

<types> element sample

<HTTP://WWW.DNEONLINE.COM/CALCULATOR.ASMX?WSDL>

```
▼<wsdl:types>
  ▼<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
    ▼<s:element name="Add">
      ▼<s:complexType>
        ▼<s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
          <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    ▶<s:element name="AddResponse">...</s:element>
    ▶<s:element name="Subtract">...</s:element>
    ▶<s:element name="SubtractResponse">...</s:element>
    ▶<s:element name="Multiply">...</s:element>
    ▶<s:element name="MultiplyResponse">...</s:element>
    ▶<s:element name="Divide">...</s:element>
    ▶<s:element name="DivideResponse">...</s:element>
  </s:schema>
</wsdl:types>
```

<message> element

```
<!-- part 3 : Message -->
<message name="SayHelloRequest">
| <part name="name" type="xsd:string" />
</message>
<message name="SayHelloResponse">
| <part name="hi" type="xsd:string" />
</message>
```

● <message> ... </message> (1..N)

- The message element declares the form of a message that the WS sends and receives.
- It describes the payload of outgoing and incoming messages.
- It corresponds to a single piece of information moving between the invoker and a WS.
- The message element defines what kind of message is expected as **input**, **output** and **fault** by this WS. Each message is constructed from a number of XML Schema-typed part elements. A message may contain multiple parts <part>.

<portType> element

```
<!-- part 4 : PortType -->
<portType name="mynamePortType">
    <operation name="SayHello">
        <input message="tns:SayHelloRequest" />
        <output message="tns:SayHelloResponse" />
    </operation>
</portType>
```

- A WSDL definition can contain 0..N <portType>.
- <portType> contains a named set of operations. It defines the functionality of the Web service. A portType is a way of grouping operations.
- A WSDL <portType> element may have one or more <operation> elements, each of which defines an RPC-style or document-style Web service method.
- Each <operation> element is composed of :
 - 0..1 <input> element
 - 0..1 <output> element and
 - 0..N (any number) <fault> elements.

Operations in WSDL are the equivalent of method signatures in programming languages. They represent the various methods being exposed by the service. An operation defines a method on a Web service, including the name of the method and the input and output parameters. A typical operation defines the input and output parameters or exceptions (faults) of an operation.

<operation> exchange pattern

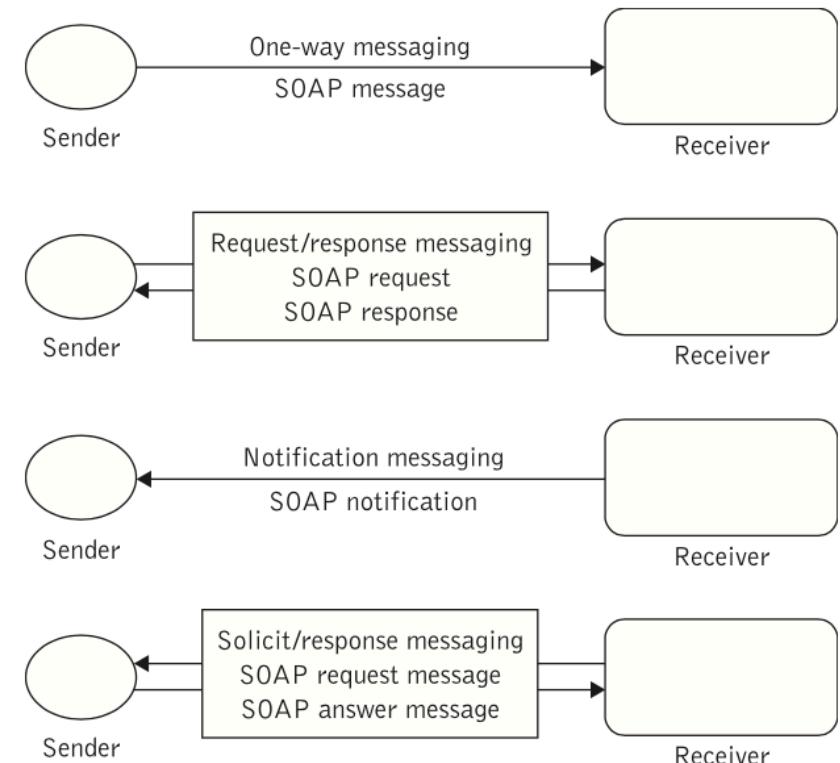
- The operation element indicates a message exchange pattern:

Synchronous interaction

1. **Request-response** (i.e., Input–Output): The WS receives a message and sends a correlated message (or fault).
2. **Solicit-response** (i.e., Output–Input): The WS generates a message and receives a correlated message (or fault) in return.

Asynchronous interaction

3. **One-way** (i.e., Input only):
4. **Notification** (i.e., Output only): The WS sends a message. It does not expect anything in return.



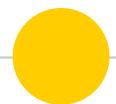
WSDL messaging patterns



Request-response (i.e., Input-Output)

- The WS receives a message and sends a correlated message (or fault).
- **Example:** a **SendPurchase** operation receives as input a message containing a purchase order (order number and date) and customer details and responds with a message containing an invoice.

```
<!-- portType element describes the abstract interface of a Web
     service -->
<wsdl:portType name="PurchaseOrder_PortType">
    <wsdl:operation name="SendPurchase">
        <wsdl:input message="tns:POMessage" />
        <wsdl:output message="tns:InvMessage" />
    </wsdl:operation>
</wsdl:portType>
```



One-way (i.e., Input only)

- The WS receives a message. The WS consumes the message and does not produce any output or fault message.
- **Example** : an operation representing the submission of an order to a purchasing system. Once the order is sent, no immediate response is expected.

```
<!-- portType element describes the abstract interface of a Web
     service -->
<wsdl:portType name="SubmitPurchaseOrder_PortType">
    <wsdl:operation name="SubmitPurchaseOrder">
        <wsdl:input name="order"
                    message="tns:SubmitPurchaseOrder_Message" />
    </wsdl:operation>
</wsdl:portType>
```



Notification (i.e., Output only)

- The WS sends a message. It does not expect anything in return.
- This type of messaging is used by services that need to **notify** clients of events (*push model*).: The assumption is that the client (subscriber) has registered with the WS to receive messages (notifications) about an event.
- **Example** : a service model in which events are reported to the service and where the endpoint periodically reports its status, for instance subscribing to a flight status.

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken">
            <wsdl:output name="nmtoken"? message="qname" />
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```



Solicit-response (i.e., Output-Input)

- The WS generates a message and receives a correlated message (or fault) in return.
- the service endpoint is initiating the operation (soliciting the client), rather than responding to a request.
- It is like notification messaging, except that the client is expected to respond to the WS. As with notification messaging, clients of the solicit/response WS must subscribe to the service in order to receive messages.
- An example of this operation might be a service that sends out order status to a client and receives back a receipt.

```
<wsdl:portType .... >
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
        <wsdl:output name="nmtoken" ? message="qname" />
        <wsdl:input name="nmtoken" ? message="qname" />
        <wsdl:fault name="nmtoken" message="qname" />
    </wsdl:operation>
</wsdl:portType >
```

<binding> element

```
<!-- part 5 : Binding -->
<binding name="mynameBinding" type="tns:mynamePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="SayHello">
    <soap:operation soapAction="http://www.mynameSpace.com#SayHello" style="rpc" />
    <input>
      <soap:body use="encoded"
                 namespace="http://www.mynameSpace.com"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded"
                 namespace="http://www.mynameSpace.com"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
```

○ <binding> element contains information of how the elements in an **abstract service interface** (<portType> and <operation> elements) are mapped to a **concrete representation**:

Protocols (e.g., SOAP or HTTP),

Messaging styles (e.g., RPC or documents styles),

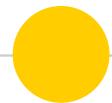
Formatting (encoding)

Styles (e.g., literal or SOAP encoding).

<binding> element

- A WSDL definition contains 1..N <binding>.
- Several bindings may represent various implementations of the same <portType> element: If a service supports more than one protocol, then the WSDL <portType> element should include a <binding> for each protocol it supports.

```
► <wsdl:portType name="CalculatorSoap">...</wsdl:portType>
► <wsdl:binding name="CalculatorSoap" type="tns:CalculatorSoap">...</wsdl:binding>
► <wsdl:binding name="CalculatorSoap12" type="tns:CalculatorSoap">...</wsdl:binding>
▼ <wsdl:service name="Calculator">
  ► <wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">...</wsdl:port>
  ► <wsdl:port name="CalculatorSoap12" binding="tns:CalculatorSoap12">...</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```



Messaging style

- 2 messaging style are used :

- RPC-based style
 - <soap:binding style='**rpc**' ...>

- Document-based style
 - <soap:binding style='**document**' ...>

```
<env:Envelope ...>
  <env:Header> some header </env:Header>
  <env:Body>
    <m:GetProductPrice>
      <product-id> 450R60P </product-id>
    </m:GetProductPrice>
  </env:Body>
</env:Envelope>

<env:Envelope ...>
  <env:Header> some header </env:Header>
  <env:Body>
    <m:GetProductPriceResponse>
      <product-price> 134.32 </product-price>
    </m:GetProductPriceResponse>
  </env:Body>
</env:Envelope>
```

```
<env:Envelope ...>
  <env:Header> some header </env:Header>
  <env:Body>
    <po:PurchaseOrder orderDate="2004-12-02">
      <po:from>
        <po:accountName> RightPlastics </po:accountName>
        <po:accountNumber> PSC-0343-02 </po:accountNumber>
      </po:from>
      <po:to>
        <po:supplierName>Plastic Supplies Inc.</po:supplierName>
        <po:supplierAddress>Melbourne</po:supplierAddress>
      </po:to>
      <po:product>
        <po:product-name> injection molder </po:product-name>
        <po:product-model> G-100T </po:product-model>
        <po:quantity> 2 </po:quantity>
      </po:product>
    </po:PurchaseOrder>
  </env:Body>
</env:Envelope>
```



Messaging style : RPC

- ◉ <soap:binding style='rpc' ...>
- ◉ Stems from the conventional remote procedure call (RPC) style.
 - This style supports **synchronous** request-response interaction.

How to recognize SOAP RPC-style ?

- The SOAP <Body> must conform to a structure that indicates the **method** name and contains a set of **parameters**.
- The response always has '**Response**' appended after the request method.



Messaging style : Document

<soap:binding style='document' ...>

- Flexible communication style that provides the best interoperability, used by Modern SOAP engines.
- Described as a message-driven, **asynchronous** communication model.

How to recognize a SOAP document-style ?

- In this style, WSDL definitions must contain XML schema definitions of request and response messages.
- The body of each message is an **XML fragment** as defined in the WSDL document.
- The client sends an entire XML document (e.g., purchase order) as a request to an endpoint, rather than a discrete set of parameters and a service operation name.
- The endpoint on the server side is responsible for forwarding and dispatching the request to the right service operation that can process the message.

Why do we need several <binding> and <port> for a same service ?

Scenario :

- A same WS, could, for instance, contain 3 ports, all of which use the same <PortType> but are bound to :
 - SOAP over HTTP,
 - SOAP over SMTP, and
 - HTTP GET/POST, respectively.
- When many protocols are supported:
 - A client that wishes to interact with the service can choose which protocol it wants to use to communicate with the service
 - The service is more readily accessible on a wider range of platforms:
 - A PC desktop application may use SOAP over HTTP,
 - A mobile application may use HTTP GET/POST

<service> Element

- The <service> element appears at the end of the WSDL document and it points to the specific network endpoint for a binding (i.e., access point of the service)

```
<!-- part 6 : Service -->
<service name="myname">
    <!-- part 7 : Port -->
    <port name="mynamePort" binding="tns:mynameBinding">
        <soap:address location="http://localhost/mywebservices/webservice.php" />
    </port>
</service>
</definitions>
```

<service> Element

- It binds the Web service to a specific network addressable location. It takes the bindings that were declared previously and ties them to one or more <port> elements, each of which represents a Web service.

```
► <wsdl:binding name='CalculatorSoap' type="tns:CalculatorSoap">...</wsdl:binding>
► <wsdl:binding name='CalculatorSoap12' type="tns:CalculatorSoap12">...</wsdl:binding>
▼ <wsdl:service name="Calculator">
    ▼ <wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">
        <soap:address location="http://www.dneonline.com/calculator.asmx" />
    </wsdl:port>
    ▼ <wsdl:port name="CalculatorSoap12" binding="tns:CalculatorSoap12">
        <soap12:address location="http://www.dneonline.com/calculator.asmx" />
    </wsdl:port>
</wsdl:service>
```

<service> Element with several ports

