



Caso práctico

En **BK Programación**, **Juan** ha estado diseñando la aplicación web que deben desarrollar. El trabajo le está sirviendo para actualizar sus conocimientos en el lenguaje **PHP**. Y al mismo tiempo, se está dando cuenta de que cuanto más conoce, más herramientas puede utilizar para programar.



no de los detalles que le preocupa del nuevo proyecto, es la posibilidad de reutilizar en el futuro parte del código que se genere. Por ejemplo, si se crea una función para ver las unidades almacenadas, en las tiendas, de un producto concreto, sería bueno que esa función se pudiera usar no sólo desde la propia aplicación web, sino también desde cualquier otra aplicación que pueda necesitar esa información.

Lo ha estado hablando con **Esteban**, y antes de avanzar más han decidido tomarse un tiempo para evaluar las distintas posibilidades con las que cuentan al respecto. Seguramente retrase un tiempo el proyecto, pero a cambio la aplicación que obtendrán resultará más abierta y su información podrá aprovecharse de forma sencilla cuando sea necesario.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Servicios web.



Caso práctico

De su trabajo anterior con la programación de aplicaciones, **Juan** conoce algunos mecanismos que podrían aplicarse en este caso. En uno de sus primeros proyectos utilizó una técnica llamada **RPC**, que le permitía a un programa ejecutar de forma remota funciones que se encontraban en otro equipo y obtener su resultado.



pero sabe que ahora existen otros métodos más potentes de funcionamiento similar: los servicios web. Aunque nunca los ha utilizado, tiene una idea general sobre su funcionamiento. Tiene que profundizar sobre ellos para ver si se adaptan a lo que busca y conocer de forma más precisa cómo se pueden integrar con una aplicación web programada en lenguaje **PHP**.

En ocasiones, las aplicaciones que desarrolles necesitarán compartir información con otras aplicaciones.

Sin ir más lejos, cojamos la aplicación de tienda web que estuvimos utilizando en los ejemplos y ejercicios del módulo. La información que se almacena sobre los productos incluye su código, nombre, descripción, PVP, etc. Seguramente los proveedores a los que se compran los artículos, manejen la misma o parecida información. Y quizás puedas aprovechar esa información para tu propia aplicación.

O puede ser que, una vez que esté finalizada y funcionando, quieras programar una nueva aplicación (y no necesariamente una aplicación web) que la complemente para, por ejemplo, procesar la información sobre los pedidos realizados.

Para compartir la información que gestiona tu aplicación, normalmente es suficiente con dar acceso a la base de datos en que se almacena. Pero ésta generalmente no es una buena idea. Cuantas más aplicaciones utilicen los mismos datos, más posibilidades hay de que se generen errores en los mismos. Además, existen otros inconvenientes:

- ✓ Si ya tienes una aplicación funcionando, ya has programado la lógica de negocio correspondiente, y ésta no se podrá aprovechar en otras aplicaciones si utilizan directamente la información almacenada en la base de datos.
- ✓ Si quieres poner la base de datos a disposición de terceros, éstos necesitarán conocer su estructura. Y al dar acceso directo a los datos, será complicado mantener el control sobre las modificaciones que se produzcan en los mismos.

Por otro lado, gran parte de la información que gestionan las aplicaciones web ya está disponible para que otros la utilicen (dejando a un lado las consideraciones relacionadas con el control de acceso). Por ejemplo, si alguien quiere conocer el precio de un producto en la tienda web, basta con buscar ese producto en la página en que se listan todos los productos. Pero, para que esa misma información (el precio de un producto) la pueda

obtener un programa, éste tendría que contemplar un procedimiento para buscar el producto concreto dentro de las etiquetas HTML de la página y extraer su precio.

Para facilitar esta tarea existen los servicios web. Un servicio web es un método que permite que dos equipos intercambien información a través de una red informática. Al utilizar servicios web, el servidor puede ofrecer un punto de acceso a la información que quiere compartir. De esta forma controla y facilita el acceso a la misma por parte de otras aplicaciones.

Los clientes del servicio, por su parte, no necesitan conocer la estructura interna de almacenamiento. En lugar de tener que programar un mecanismo para localizar la información, tienen un punto de acceso directo a la que les interesa.

Volviendo al ejemplo de nuestra tienda, si quisiéramos aprovechar la información de que disponen nuestros proveedores, éstos tendrían que ofrecer un servicio web que nos permitiese recuperarla. Por ejemplo, enviándonos el código de un producto, podríamos obtener su nombre, descripción, precio, etc. Inversamente, si quisiéramos facilitar la obtención de datos de nuestra tienda por parte de otras aplicaciones, podríamos programar y ofrecer un servicio web de forma que, por ejemplo, devolviese el listado de pedidos del cliente que se requiera.

1.1.- Características.

Existen numerosos protocolos que permiten la comunicación entre ordenadores a través de una red: FTP, HTTP, SMTP, POP3, TELNET, etc. En todos estos protocolos se definen un servidor y un cliente. El servidor es la máquina que está esperando conexiones (escuchando) por parte de un cliente. El cliente es la máquina que inicia la comunicación. Cada uno de estos protocolos tiene asignado además un puerto (TCP o UDP) concreto, que será el que utilicen normalmente los equipos servidores.



[Everaldo Coelho and YellowIcon \(GNU/GPL\)](#)

Cada uno de los protocolos que hemos nombrado ha sido creado para un fin específico: FTP para transferencia de archivos, HTTP para páginas web, SMTP y POP3 para correo electrónico y TELNET para acceso remoto. No han sido diseñados para transportar peticiones de información genéricas entre aplicaciones, como solicitar el PVP de un producto. Sin embargo, ya desde hace tiempo existen otras soluciones para este tipo de problemas. Una de las más populares es RPC.

El protocolo RPC se creó para permitir a un sistema acceder de forma remota a funciones o procedimientos que se encuentren en otro sistema. El cliente se conecta con el servidor, y le indica qué función debe ejecutar. El servidor la ejecuta y le devuelve el resultado obtenido. Así, por ejemplo, podemos crear en el servidor RPC una función que reciba un código de producto y devuelva su PVP.

RPC usa su propio puerto, pero normalmente solo a modo de directorio. Los clientes se conectan a él para obtener el puerto real del servicio que les interesa. Este puerto no es fijo; se asigna de forma dinámica.

Los servicios web se crearon para permitir el intercambio de información al igual que RPC, pero sobre la base del protocolo HTTP (de ahí el término web). En lugar de definir su propio protocolo para transportar las peticiones de información, utilizan HTTP para este fin. La respuesta obtenida no será una página web, sino la información que se solicitó. De esta forma pueden funcionar sobre cualquier servidor web; y, lo que es aún más importante, utilizando el puerto **80** reservado para este protocolo. Por tanto, cualquier ordenador que pueda consultar una página web, podrá también solicitar información de un servicio web. Si existe algún cortafuegos en la red, tratará la petición de información igual que lo haría con la solicitud de una página web.

Existen al menos dos cuestiones que debería resolver un servicio web para poder funcionar correctamente:

- ✓ Cómo se transmite la información. Si se va a usar HTTP para las peticiones y las respuestas, el cliente y el servidor tendrán que ponerse de acuerdo en la forma de enviar unas y otras. Es decir, ¿cómo hace el cliente para indicar que quiere conocer el PVP del artículo con código **X**?, y también, ¿cómo envía el servidor la respuesta obtenida?
- ✓ Cómo se publican las funciones a las que se puede acceder en un servidor determinado. Este punto es opcional, pero muy útil. Es decir, el cliente puede saber que la función del servidor que tiene que utilizar se llama "getPVPArticulo()", y que debe recibir como parámetro el código del artículo. Pero si no lo sabe, sería útil que hubiera un mecanismo donde pudiera consultar las funciones que existen en el servidor y cómo se utiliza cada una.

Cada uno de los métodos que podemos utilizar hoy en día para crear un servicio web responde a estas preguntas de formas distintas. Para la primera cuestión, nosotros veremos el protocolo SOAP, que utiliza el lenguaje XML para intercambiar información. En

cuanto a la segunda cuestión, la resolveremos con un lenguaje llamado **WSDL**, que también está basado en **XML** y fue creado para describir servicios web, es decir, indicar cómo se debe acceder a un servicio y utilizarlo.



Autoevaluación

Relaciona las siglas con aquello a que hacen referencia:

Ejercicio de relacionar

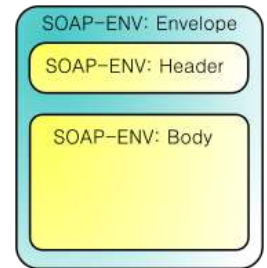
Siglas	Relación	Significado
SOAP.	<input type="checkbox"/>	1. Protocolo para transmitir páginas web.
HTTP.	<input type="checkbox"/>	2. Protocolo para ejecutar código de forma remota.
RPC.	<input type="checkbox"/>	3. Protocolo para intercambiar información en un servicio web.
WSDL.	<input type="checkbox"/>	4. Lenguaje para describir servicios web.

Enviar

Es importante tener claros los conceptos antes de profundizar en su funcionamiento.

1.2.- Intercambio de información: SOAP.

SOAP es un protocolo que indica cómo deben ser los mensajes que se intercambien el servidor y el cliente, cómo deben procesarse éstos, y cómo se relacionan con el protocolo que se utiliza para transportarlos de un extremo a otro de la comunicación (en el caso de los servicios web, este protocolo será HTTP).



[Silver Spoon Sokpor](#) (CC BY-SA)

Aunque nosotros vamos a utilizar HTTP para transmitir la información, SOAP no requiere el uso de un protocolo concreto para transmitir la información. SOAP se limita a definir las reglas que rigen los mensajes que se deben intercambiar el cliente y el servidor. Cómo se envíen esos mensajes no es relevante desde el punto de vista de SOAP. En lugar de utilizar HTTP para transmitirlos, se podrían utilizar, por ejemplo, correos electrónicos (claro que en este caso ya no sería un servicio web).

El nombre SOAP surgió como acrónimo de "*Simple Object Access Protocol*", pero, a partir de la versión **1.2** del protocolo, el nombre SOAP ya no se refiere a nada en concreto.

Al igual que su antecesor, XML-RPC, SOAP utiliza XML para componer los mensajes que se transmiten entre el cliente (que genera una petición) y el servidor (que envía una respuesta) del servicio web.



Diagrama elaborado con DIA (Elaboración propia.)

Veamos un ejemplo. Si implementamos un servicio web para informar sobre el precio de los artículos que se venden en la tienda web, una petición de información para el artículo con código '**KSTMSDHC8GB**' podría ser de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1=""
  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:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
```

```
<soap:Body>
  <ns1:getPVP>
    <param0 xsi:type="xsd:string">KSTMSDHC8GB</param0>
  </ns1:getPVP>
</soap:Body>
</soap:Envelope>
```

Y su respectiva respuesta:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns1=""
  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:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <soap:Body>
    <ns1:getPVPPResponse>
      <return xsi:type="xsd:string">10.20</return>
    </ns1:getPVPPResponse>
  </soap:Body>
</soap:Envelope>
```

1.2.1.- Intercambio de información: SOAP (II).

En un mensaje SOAP, como mínimo debe figurar un elemento "Envelope", que es lo que identifica al documento XML como un mensaje SOAP, y donde se deben declarar al menos los siguientes espacios de nombres:

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

El espacio de nombres que se utilice para el elemento Envelope indica la versión del protocolo SOAP utilizado. En la versión **1.1** (la del ejemplo anterior), el espacio de nombres es: `http://schemas.xmlsoap.org/soap/envelope/`. En la versión **1.2** se debe utilizar: `http://www.w3.org/2003/05/soap-envelope`.

Al cambiar la versión de SOAP también se deben cambiar los espacios de nombres relativos al estilo de codificación. En la versión **1.1**, se debe utilizar: `http://schemas.xmlsoap.org/soap/encoding/`, y en la versión **1.2**: `http://www.w3.org/2003/05/soap-encoding`.

Como primer miembro del elemento "Envelope", puede haber de forma opcional un elemento Header. Si existe, puede contener varios elementos con información adicional sobre cómo procesar el mensaje SOAP. A continuación debe figurar obligatoriamente un elemento "Body", que es dónde se incluye, dependiendo del tipo de mensaje, la petición o la respuesta.

Sería muy complejo programar un servicio web que procesase el XML recibido en cada petición SOAP, y generase el XML relativo a cada respuesta correspondiente. Existen mecanismos de ayuda que nos evitan tener que tratar con las interioridades del protocolo SOAP.

De las implementaciones de SOAP que podemos usar con PHP, cabe destacar tres: NuSOAP, PEAR::SOAP y PHP SOAP. Las tres nos permiten crear tanto un cliente como un servidor SOAP, pero existen algunas características que las diferencian:

- ✓ PHP SOAP es la implementación de SOAP que se incluye con PHP a partir de la versión **5** del lenguaje. En versiones anteriores se tenía que recurrir a otras opciones para trabajar con SOAP. Es una extensión nativa (escrita en lenguaje **C**) y por tanto más rápida que las otras posibilidades. Como veremos más adelante, su gran inconveniente es que no permite la generación automática del documento WSDL una vez programado el servidor SOAP correspondiente.
- ✓ NuSOAP es un conjunto de clases programadas en PHP que ofrecen muchas funcionalidades para utilizar SOAP. Al contrario que PHP SOAP, funcionan también con PHP4, y además permite generar automáticamente el documento WSDL correspondiente a un servicio web.
- ✓ PEAR::SOAP es un paquete PEAR que permite utilizar SOAP con PHP a partir de su versión **4**. Al igual que NuSOAP, también está programado en PHP.



Debes conocer

Debido a una coincidencia en el nombre de las clases, **NuSOAP** es incompatible con **PHP SOAP**. Ambas incluyen una clase de nombre **SoapClient**. Si quieres programar con **NuSOAP** en **PHP5**, es recomendable cambiar el nombre de esta clase o utilizar la alternativa **NuSOAP for PHP5**, que utiliza en su lugar el nombre **SOAPClientNuSOAP**.

[NuSOAP for PHP5.](#)

Más adelante aprenderás a crear y utilizar servicios web desde **PHP** con **PHP SOAP**.



Autoevaluación

El elemento **Envelope** debe figurar como raíz en un mensaje **SOAP**, y obligatoriamente deberá contener un elemento:

- ☐ **Header**.
- ☐ **Body**.

El elemento **Envelope** puede contener un elemento **Header**, pero su uso es opcional.

Efectivamente. El elemento **Header** es opcional, pero el elemento **Body** debe figurar obligatoriamente dentro del elemento **Envelope**.

Solución

1. Incorrecto
2. Opción correcta

1.3.- Descripción del servicio: WSDL.

Una vez que hayas creado un servicio web, puedes programar el correspondiente cliente y comenzar a utilizarlo. Como el servicio lo has creado tú, sabrás cómo acceder a él: en qué URL está accesible, qué parámetros recibe, cuál es la funcionalidad que aporta, y qué valores devuelve. Sin embargo, si lo que quieres es que el servicio web sea accesible a aplicaciones desarrolladas por otros programadores, deberás indicarles cómo usarlo, es decir, crear un documento WSDL que describa el servicio.

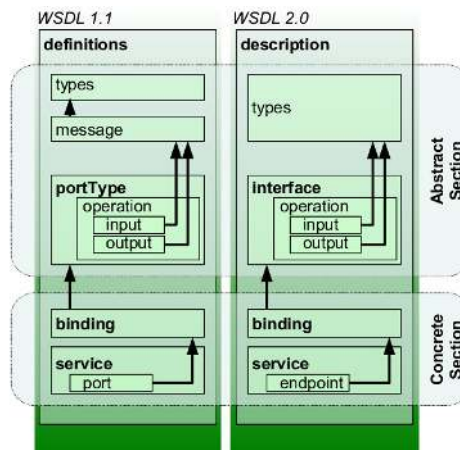
WSDL es un lenguaje basado en XML que utiliza unas reglas determinadas para generar el documento de descripción de un servicio web. Una vez generado, ese documento se suele poner a disposición de los posibles usuarios del servicio (normalmente se accede al documento WSDL añadiendo "?wsdl" a la URL del servicio).

El espacio de nombres de un documento WSDL es: <http://schemas.xmlsoap.org/wsdl/>, aunque en un documento WSDL se suelen utilizar también otros espacios de nombres. La estructura de un documento WSDL es la siguiente:

```
<definitions
  name="..."
  targetNamespace="http://..."
  xmlns:tns="http://..."
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  ...
>
  <types>
    ...
  </types>
  <message>
    ...
  </message>
  <portType>
    ...
  </portType>
  <binding>
    ...
  </binding>
  <service>
    ...
  </service>
</definitions>
```

El objetivo de cada una de las secciones del documento es el siguiente:

- ✓ **types.** Incluye las definiciones de los tipos de datos que se usan en el servicio.
- ✓ **message.** Define conjuntos de datos, como la lista de parámetros que recibe una función o los valores que devuelve.
- ✓ **portType.** Cada **portType** es un grupo de funciones que implementa el servicio web. Cada función se define dentro de su **portType** como una operación (**operation**).
- ✓ **binding.** Define cómo va a transmitirse la información de cada **portType**.
- ✓ **service.** Contiene una lista de elementos de tipo **port**. Cada **port** indica dónde (en qué URL) se puede acceder al servicio web.



[Cristcost](#) (Dominio público)

Las secciones anteriores son las correspondientes a la versión **1.1** de **WSDL**. En la versión **2.0**, los conceptos y la nomenclatura cambia ligeramente; por ejemplo, lo que en **1.1** es un **portType** se denomina **interface** en la versión **2.0**.

En los siguientes puntos veremos paso a paso cómo crear cada una de las secciones de un documento **WSDL**.

1.3.1.- Descripción del servicio: WSDL (II).

Existen servicios web sencillos a los que puedes pasar como parámetro un número o una cadena de texto (por ejemplo, las siglas de una moneda, [USD](#)), y te devuelven también un dato de un tipo simple, como un número decimal (la tasa de conversión actual). E igualmente existen también servicios web más elaborados, que pueden requerir o devolver un array de elementos, o incluso objetos.

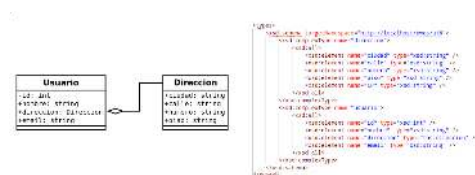
Para crear y utilizar estos servicios, deberás definir los tipos de elementos que se transmiten: de qué tipo son los valores del array, o qué miembros poseen los objetos que maneja. La definición de tipos en **WSDL** se realiza utilizando la etiqueta **types**. Veamos un ejemplo:



[Everaldo Coelho \(Yellowicon\)](#)
(GNU/GPL)

```
<types>
  <xsd:schema targetNamespace="http://localhost/dwes/ut6">
    <xsd:complexType name="direccion">
      <xsd:all>
        <xsd:element name="ciudad" type="xsd:string" />
        <xsd:element name="calle" type="xsd:string" />
        <xsd:element name="numero" type="xsd:string" />
        <xsd:element name="piso" type="xsd:string" />
        <xsd:element name="CP" type="xsd:string" />
      </xsd:all>
    </xsd:complexType>
    <xsd:complexType name="usuario">
      <xsd:all>
        <xsd:element name="id" type="xsd:int" />
        <xsd:element name="nombre" type="xsd:string" />
        <xsd:element name="direccion" type="tns:direccion" />
        <xsd:element name="email" type="xsd:string" />
      </xsd:all>
    </xsd:complexType>
  </xsd:schema>
</types>
```

En el código anterior, se definen dos tipos de datos usando XML Schema: **direccion** y **usuario**. De hecho, los tipos **direccion** y **usuario** son la forma en que se definen en **WSDL** las clases para transmitir la información de sus objetos.



Elaborado con DIA y Gimp (Elaboración propia)

En WSDL, las clases se definen utilizando los tipos complejos de XML Schema. Al utilizar all dentro del tipo complejo, estamos indicando que la clase contiene esos miembros, aunque no necesariamente en el orden que se indica (si en lugar de all hubiésemos utilizado sequence, el orden de los miembros de la clase debería ser el mismo que figura en el documento).

Obviamente, los métodos de la clase forman parte de la lógica de la aplicación y no se definen en el documento WSDL.

Aunque en WSDL se puede usar cualquier lenguaje para definir los tipos de datos, es aconsejable usar XML Schema, indicándolo dentro de la etiqueta types e incluyendo el espacio de nombres correspondiente en el elemento definitions.

```
<definitions
  name="WSDLusuario"
  targetNamespace="http://localhost/dwes/ut6"
  xmlns:tns="http://localhost/dwes/ut6"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  . . .
>
```

El otro tipo de datos que necesitaremos definir en los documentos WSDL son los **arrays**. Para definir un array, no existe en el XML Schema un tipo base adecuado que podamos usar. En su lugar, se utiliza el tipo Array definido en el esquema encoding de SOAP. Por ejemplo, podríamos añadir un tipo array de usuarios al documento anterior haciendo:

```
<xsd:complexType name="ArrayOfusuario">
  <xsd:complexContent>
    <xsd:restriction base="soapenc:Array">
      <xsd:attribute ref="soapenc:arrayType" arrayType="tns:usuario[]" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

Al definir un array en WSDL, se debe tener en cuenta que:

- ✓ El atributo arrayType se utiliza para indicar qué elementos contendrá el array.
- ✓ Se debe añadir al documento el espacio de nombres SOAP encoding:

```
<definitions
  name="WSDLusuario"
  targetNamespace="http://localhost/dwes/ut6"
  xmlns:tns="http://localhost/dwes/ut6"
```

```
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
. . .
>
```

- ✓ El nombre del array debería ser **ArrayOfXXX**, dónde **xxx** es el nombre del tipo de elementos que contiene el array.

En muchas ocasiones no será necesario definir tipos propios, y por tanto en el documento **WSDL** no habrá sección **types**; será suficiente con utilizar alguno de los tipos propios de **XML Schema**, como **xsd:string**, **xsd:float** o **xsd:boolean**.



Autoevaluación

En la sección **types** de un documento **WSDL**, se deben definir:

- ☐ Todos los tipos de elementos que se usen en el servicio web.
- ☐ Los tipos de elementos compuestos que se usen en el servicio web, como los objetos y arrays.

Como acabas de ver, en ocasiones no será necesario definir tipos en el documento **WSDL**, puesto que se pueden usar los tipos propios de **XML Schema**.

Al usar como base los tipos propios de **XML Schema**, solo será necesario definir aquellos tipos compuestos que se utilicen en el servicio web.

Solución

1. Incorrecto
2. Opción correcta

1.3.2.- Descripción del servicio: WSDL (III).

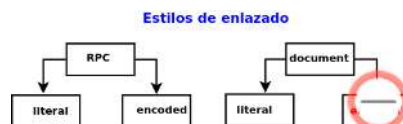
Ahora que ya sabes cómo definir los tipos de datos que se usan en un servicio web, el siguiente paso es indicar cómo se agrupan esos tipos para formar los parámetros de entrada y de salida. Veámoslo con un ejemplo. Siguiendo con los usuarios que acabamos de definir, podríamos crear en el servicio web una función `getUsuario()` para dar acceso a los datos de un usuario. Como parámetro de entrada de esa función vamos a pedir el "id" del usuario, y como valor de salida se obtendrá un objeto "usuario". Por tanto, debemos definir los siguientes mensajes:



[Everaldo Coelho and YellowIcon](#) (GNU/GPL)

```
<message name="getUsuarioRequest">
  <part name="id" type="xsd:int"/>
</message>
<message name="getUsuarioResponse">
  <part name="getUsuarioReturn" type="tns:usuario"/>
</message>
```

Como ves, normalmente por cada función del servicio web se crea un mensaje para los parámetros de entrada, y otro para los de salida. Dentro de cada mensaje, se incluirán tantos elementos **part** como sea necesario. Cada mensaje contendrá un atributo **name** que debe ser único para todos los elementos de este tipo. Además, es aconsejable que el nombre del mensaje con los parámetros de entrada acabe en **Request**, y el correspondiente a los parámetros de salida en **Response**.



DIA (Elaboración propia)

En un documento WSDL podemos especificar dos estilos de enlazado: **document** o **RPC**. La selección que hagamos influirá en cómo se transmitan los mensajes dentro de las peticiones y respuestas SOAP. Por ejemplo, un mensaje SOAP con estilo **document** podría ser:

```
<SOAP-ENV:Body>
  <producto>
    <codigo>KSTMSDHC8GB</codigo>
  </producto>
</SOAP-ENV:Body>
```

Y un mensaje con estilo **RPC** sería por ejemplo:

```
<SOAP-ENV:Body>
  <ns1:getPVP>
```

```
<param0 xsi:type="xsd:string">KSTMSDHC8GB</param0>
</ns1:getPVP>
</SOAP-ENV:Body>
```

El estilo de enlazado **RPC** está más orientado a sistemas de petición y respuesta que el **document** (más orientado a la transmisión de documentos en formato **XML**). En este estilo de enlazado, cada elemento **message** de **WSDL** debe contener un elemento **part** por cada parámetro (de entrada o de salida), y dentro de éste indicar el tipo de datos del parámetro mediante un atributo **type**, como se muestra en el ejemplo anterior.

Además, cada estilo de enlazado puede ser de tipo **encoded** o **literal** (aunque en realidad la combinación **document/encoded** no se utiliza). Al indicar **encoded**, estamos diciendo que vamos a usar un conjunto de reglas de codificación, como las que se incluyen en el propio protocolo **SOAP** (espacio de nombres "<http://schemas.xmlsoap.org/soap/encoding/>"), para convertir en **XML** los parámetros de las peticiones y respuestas.

El ejemplo anterior de **RPC** es en realidad **RPC/encoded**. Un ejemplo de un mensaje **SOAP** con estilo **RPC/literal** sería:

```
<SOAP-ENV:Body>
  <ns1:getPVP>
    <param0>KSTMSDHC8GB</param0>
  </ns1:getPVP>
</SOAP-ENV:Body>
```

En lo sucesivo, trabajaremos únicamente con estilo de enlazado **RPC/encoded**.

1.3.3.- Descripción del servicio: WSDL (IV).

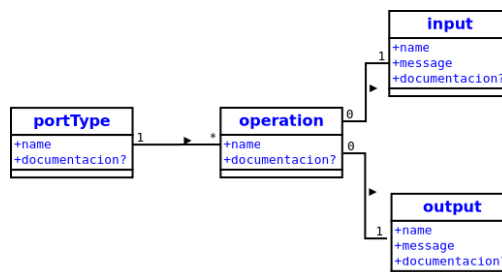
Las funciones que creas en un servicio web, se conocen con el nombre de **operaciones** en un documento **WSDL**. En lugar de definir las una a una, es necesario agruparlas en lo que en **WSDL** se llama **portType**. Un **portType** contiene una lista de funciones, indicando para cada función (**operation**) la lista de parámetros de entrada y de salida que le corresponden. Por ejemplo:

```
<portType name="usuarioPortType">
  <operation name="getUsuario">
    <input message="tns:getUsuarioRequest"/>
    <output message="tns:getUsuarioResponse"/>
  </operation>
</portType>
```



[Everaldo Coelho and YellowIcon](#) (GNU/GPL)

A no ser que estés generando un servicio web bastante complejo, el documento **WSDL** contendrá un único **portType**. Podrías necesitar dividir las funciones del servicio en distintos **portType** para, por ejemplo, utilizar un estilo de enlazado distinto para las funciones de cada grupo.



DIA (Elaboración propia)

Cada **portType** debe contener un atributo **name** con el nombre (único para todos los elementos **portType**). Cada elemento **operation** también debe contener un atributo **name**, que se corresponderá con el nombre de la función que se ofrece. Además, en función del tipo de operación de que se trate, contendrá:

- ✓ Un elemento **input** para indicar funciones que no devuelven valor (su objetivo es sólo enviar un mensaje al servidor).
- ✓ Un elemento **input** y otro **output**, en este orden, para el caso más habitual: funciones que reciben algún parámetro, se ejecutan, y devuelven un resultado.

Es posible (pero muy extraño) encontrarse funciones a la inversa: sólo con un parámetro **output** (el servidor envía una notificación al cliente) o con los parámetros **output** e **input** por ese orden (el servidor le pide al cliente alguna información). Por tanto, al definir una función (un elemento **operation**) se debe tener cuidado con el orden de los elementos **input** y **output**.

Normalmente, los elementos `input` y `output` contendrán un atributo `message` para hacer referencia a un mensaje definido anteriormente.



Autoevaluación

En un documento WSDL, cada una de las funciones que implementa el servicio se refleja en un elemento de tipo:

- ☐ `operation`.
- ☐ `portType`.

Efectivamente, pues el elemento `portType` solo es una forma de agrupar funciones (elementos `operation`), y únicamente se utiliza en caso de servicios complejos.

Revisa los ejemplos que hemos visto hasta el momento, e identifica el lugar en el que se definen las funciones del servicio web.

Solución

1. Opción correcta
2. Incorrecto

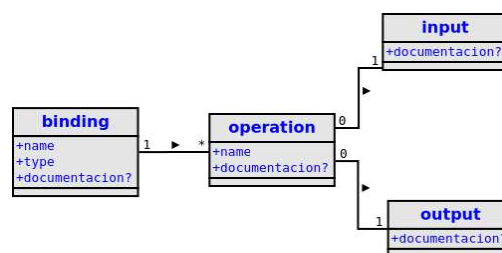
1.3.4.- Descripción del servicio: WSDL (V).

El siguiente elemento de un documento WSDL es **binding**. Antes comentábamos que existían distintos estilos de enlazado, que influían en cómo se debían crear los mensajes. En el elemento **binding** es dónde debes indicar que el estilo de enlazado de tu documento sea **RPC/encoded**.



[Everaldo Coelho and YellowIcon](#) (GNU/GPL)

Aunque es posible crear documentos WSDL con varios elementos **binding**, la mayoría contendrán solo uno (si no fuera así, sus atributos **name** deberán ser distintos). En él, para cada una de las funciones (**operation**) del **portType** que acabamos de crear, se deberá indicar cómo se codifica y transmite la información.



Elaborado con DIA (Elaboración propia)

Para el **portType** anterior, podemos crear un elemento **binding** como el siguiente:

```
<binding name="usuarioBinding" type="tns:usuarioPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getUsuario">
    <soap:operation soapAction="http://localhost/dwes/ut6/getUsuario.php?getUsuario" />
    <input>
      <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost/dwes/ut6" />
    </input>
    <output>
      <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost/dwes/ut6" />
    </output>
  </operation>
</binding>
```

Fíjate que el atributo **type** hace referencia al **portType** creado anteriormente. El siguiente elemento indica el tipo de codificación (**RPC**) y, mediante la URL correspondiente, el protocolo de transporte a utilizar (**HTTP**). Obviamente, deberás añadir el correspondiente espacio de nombres al elemento raíz:

```
<definitions
  name="WSDLusuario"
```

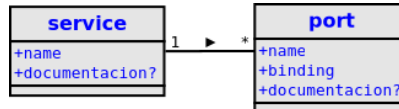
```
targetNamespace="http://localhost/dwes/ut6"
xmlns:tns="http://localhost/dwes/ut6"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
. . .
```

>

El elemento **soap:operation** debe contener un atributo **soapAction** con la URL para esa función (**operation**) en particular. Dentro de él habrá normalmente un elemento **input** y otro **output** (los mismos que en la **operation** correspondiente). En ellos, mediante los atributos del elemento **soap:body**, se indica el estilo concreto de enlazado (**encoded** con su **encodingStyle** correspondiente).

1.3.5.- Descripción del servicio: WSDL (VI).

Por último, falta definir el elemento **service**. Normalmente sólo encontraremos un elemento **service** en cada documento **WSDL**. En él, se hará referencia al **binding** anterior utilizando un elemento **port**, y se indicará la **URL** en la que se puede acceder al servicio.



Elaborado con DIA (Elaboración propia)

Por ejemplo:

```
<service name="usuario">
  <port name="usuarioPort" binding="tns:usuarioBinding">
    <soap:address location="http://localhost/dwes/ut6/getUsuario.php" />
  </port>
</service>
```

Para finalizar, veamos cómo quedaría el documento **WSDL** correspondiente a un servicio con una única función encargada de devolver el **PVP** de un producto de la tienda web a partir de su código.

```
<?xml version="1.0" encoding="UTF-8" ?>

<definitions name="WSDLgetPVP" targetNamespace="http://localhost/dwes/ut6"
  xmlns:tns="http://localhost/dwes/ut6"
  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/">
  <message name="getPVPRequest">
    <part name="codigo" type="xsd:string" />
  </message>
  <message name="getPVPResponse">
    <part name="PVP" element="xsd:float" />
  </message>
  <portType name="getPVPPortType">
    <operation name="getPVP">
      <input message="tns:getPVPRequest" />
      <output message="tns:getPVPResponse" />
    </operation>
  </portType>
  <binding name="getPVPBinding" type="tns:getPVPPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
```

```

<operation name="getPVP">
  <soap:operation soapAction="http://localhost/dwes/ut6/getPVP.php?getPVP" />
  <input>
    <soap:body namespace="http://localhost/dwes/ut6"
      use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </input>
  <output>
    <soap:body namespace="http://localhost/dwes/ut6"
      use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
</operation>
</binding>
<service name="getPVPService">
  <port name="getPVPPort" binding="tns:getPVPBinding">
    <soap:address location="http://localhost/dwes/ut6/getPVP.php" />
  </port>
</service>
</definitions>

```



Autoevaluación

Relaciona los elementos que componen un documento WSDL, con la parte del servicio web a que hacen referencia:

Ejercicio de relacionar

Elemento	Relación	Hace referencia a
types	<input type="checkbox"/>	1. Conjunto de datos, como las listas de parámetros de las funciones.
definitions	<input type="checkbox"/>	2. Caracterización del servicio, que incluye la <u>URL</u> de acceso.
message	<input type="checkbox"/>	3. Elemento raíz de un documento <u>WSDL</u> .
port	<input type="checkbox"/>	4. Definición de los tipos de elementos usados en el servicio.

Enviar

Puedes volver sobre los esquemas y ejemplos vistos con anterioridad.

2.- Extensión PHP SOAP.



Caso práctico

Juan ya tiene claro cuál es la solución que ha estado buscando: los servicios web. Ha estado leyendo sobre ellos y realizando algunas pruebas de funcionamiento, y se ajustan de forma precisa a lo que necesita.

demás ha visto que tiene distintas posibilidades para utilizar servicios web desde **PHP** y ha investigado sobre las ventajas e inconvenientes de cada una. Y en este punto es en dónde tiene más dudas. No ha encontrado una solución perfecta que le permita aprovechar de forma sencilla todas las posibilidades de los servicios web. De las opciones que existen en la actualidad, ha decidido probar con "**PHP SOAP**".



Se reúne con **Carlos** y le pone al día sobre todo lo que ha ido averiguando en los últimos días. Deben probar el funcionamiento de la extensión "**PHP SOAP**", antes de utilizarla en el nuevo proyecto. Para ello, diseñan dos servicios web sencillos, y deciden implementarlos entre ambos. Cada uno se encargará de programar un servidor y un cliente, y al final de la prueba compartirán la experiencia adquirida para tomar una decisión definitiva.

Como ya comentamos, de las posibilidades que tenemos para utilizar **SOAP** en **PHP** vamos a aprender a utilizar la extensión que viene incluida con el lenguaje a partir de su versión 5: "**PHP SOAP**".

Gracias a "**PHP SOAP**", puedes utilizar y crear de forma sencilla servicios web en tus aplicaciones. En el momento de escribir este texto, es compatible con las versiones "**SOAP 1.1**" y "**SOAP 1.2**", así como con "**WSDL 1.1**", aunque no permite la generación automática del documento **WSDL** a partir del servicio web programado.

Para poder usar la extensión, deberás comprobar si ya se encuentra disponible (por ejemplo, consultando la salida obtenida por la función `phpinfo()`):

Normalmente la extensión **SOAP** formará parte de **PHP** (se habrá compilado con el ejecutable) y podrás utilizarla directamente. Si no fuera así, deberás instalarla y habilitarla en el fichero "**php.ini**".

Las dos clases principales que deberás utilizar en tus aplicaciones son "**SoapClient**" y "**SoapServer**". La primera te permitirá comunicarte con un servicio web, y con la segunda podrás crear tus propios servicios.

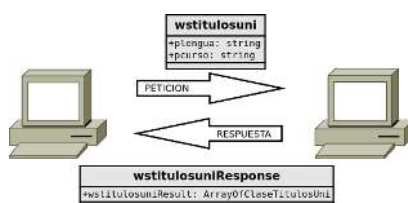
2.1.- Utilización de un servicio web.

Vamos a comenzar viendo cómo crear en PHP una aplicación que se comunique con un servicio web para obtener información. Por ejemplo, imagínate que quieres conocer lo estudios oficiales que ofrece una Universidad para un curso concreto.

Lo primero que necesitas es una universidad que ofrezca esta información via WSDL. Para el siguiente ejemplo usaremos la Universidad de Alicante.

Para crear un cliente del servicio, deberás conocer los detalles del mismo (como mínimo, los parámetros de entrada y salida que debes usar, y cuál es la URL del servicio) y emplear en tu código la clase SoapClient. Para averiguar los detalles del servicio, puedes consultar el documento WSDL del servicio, disponible en la dirección: "https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl"

Este es el documento WSDL obtenido:  [Documento obtenido](#) (pdf - 0,35 MB)



Elaborado con DIA (Elaboración propia)

- ✓ El alias del espacio de nombres correspondiente al XML Schema que utiliza el documento es:

```
<wsdl:definitions targetNamespace="http://UASI/WS_GESTDOCENTE.wsdl">
```

- ✓ En el documento puedes observar lo siguiente, se espera un elemento de tipo "wstitulosuni" que serán dos códigos: "plengua" y "pcurso" de tipo string, además en ese orden:

```
<s:element name="wstitulosuni">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="plengua" type="s:string"/>
      <s:element minOccurs="0" maxOccurs="1" name="pcurso" type="s:string"/>
    </s:sequence>
  </s:complexType>
</s:element>
```

- ✓ La respuesta "wstitulosuniResponse" será un elemento de tipo "wstitulosuniResult".


```
<s:element name="wstitulosuniResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="wstitulosuniResult" type="tns:ArrayOfC
    </s:sequence>
  </s:complexType>
</s:element>
```

- ✓ El tipo `wstitulosuniResult` es un **array** de tipo `ClaseTitulosUni` (`ArrayOfClaseTitulosUni`). Este **array** tendrá los elementos siguientes:

```
<s:complexType name="ClaseTitulosUni">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="codigo" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="nombre" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="tipo" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="area" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="url" type="s:string"/>
    <s:element minOccurs="0" maxOccurs="1" name="imagen" type="s:string"/>
  </s:sequence>
</s:complexType>
```



Autoevaluación

El principal inconveniente de la extensión **PHP SOAP** es que:

- ☐ No ofrece un interface de programación orientado a objetos.
- ☐ Una vez que has programado un servicio web, no permite generar de forma automática el documento WSDL.

Revisa lo que acabas de ver sobre la utilización de la extensión **PHP SOAP**.

Efectivamente, aunque existen librerías disponibles en varios lenguajes de programación que permiten la generación automática del documento WSDL a partir de un servicio web, no es el caso de **PHP SOAP**.

Solución

1. Incorrecto
2. Opción correcta

2.1.1.- Utilización de un servicio web (II).

El estilo de enlazado es "document/literal" (recuerda que nosotros vimos el "RPC/encoded" solamente), por lo que los elementos de tipo "message" tienen un formato distinto. Sin embargo, en base a su contenido (fíjate en los elementos que terminan en "Soap") se puede deducir también que:

- ✓ El nombre de la función a la que debes llamar es "wstitulosuni".

```
<wsdl:operation name="wstitulosuni">
```

- ✓ Como parámetro de entrada le tienes que pasar un elemento de tipo "wstitulosuni" (dos string), y devolverá un elemento "wstitulosuniResponse" (un array).

```
<wsdl:message name="wstitulosuniSoapIn">
  <wsdl:part name="parameters" element="tns:wstitulosuni"/>
</wsdl:message>
<wsdl:message name="wstitulosuniSoapOut">
  <wsdl:part name="parameters" element="tns:wstitulosuniResponse"/>
</wsdl:message>
. . .
<wsdl:portType name="pub_gestdocenteSoap">
  . . .
  <wsdl:operation name="wstitulosuni">
    <wsdl:input message="tns:wstitulosuniSoapIn"/>
    <wsdl:output message="tns:wstitulosuniSoapOut"/>
  </wsdl:operation>
  . . .
</wsdl:portType>
```

- ✓ La URL para acceder al servicio, la puedes ver más abajo, de igual manera puedes observar que el servicio se ofrece para versión **1.1** y **1.2** de SOAP.

```
<wsdl:service name="pub_gestdocente">
  <wsdl:port name="pub_gestdocenteSoap" binding="tns:pub_gestdocenteSoap">
    <soap:address location="https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.as
  </wsdl:port>
  <wsdl:port name="pub_gestdocenteSoap12" binding="tns:pub_gestdocenteSoap12">
    <soap12:address location="https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.
  </wsdl:port>
  . . .
</wsdl:service>
```

Con la información anterior, para utilizar el servicio desde PHP creas un nuevo objeto de la clase "**SoapClient**". Como el servicio tiene un documento WSDL asociado, en el constructor le indicas dónde se encuentra:

```
$cliente=new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
```

Y para realizar la llamada a la función "wstitulosuni", incluyes los parámetros en un **array**:

```
$parametros=[  
    'plengua'=>'es',  
    'pcurso'=>'2019'  
];  
$titulos=$cliente->wstitulosuni($parametros);
```

La llamada devuelve un objeto de una clase predefinida en PHP llamada: **stdClass**. Si hacemos "var_dump(\$titulos)" obtenemos lo siguiente:

```
object(stdClass)[2]  
  public 'wstitulosuniResult' =>  
    object(stdClass)[3]  
      public 'ClaseTitulosUni' =>  
        array (size=113)
```

Vemos que nos ha devuelto un objeto **stdClass** dentro podemos observar que nos ha devuelto 113, títulos. Si recorremos \$titulos con el código siguiente:

```
foreach($titulos as $k=>$v){  
    foreach($v as $k1=>$v1){  
        echo var_dump($v1);  
    }  
}
```

Veremos lo siguiente:



Captura de pantalla Firefox (Elaboración propia)



Para saber más

En **PHP** existe una clase predefinida en el lenguaje que se llama "**stdClass**", que no tiene ni propiedades, ni métodos, ni clase padre; es una clase vacía. Podemos usar esta clase cuando necesitamos un objeto genérico al que luego el podremos añadir propiedades.

Hay que tener en cuenta, que esta clase no pertenece a la clase de la que heredan todas las clases.

2.1.2.- Utilización de un servicio web (III).

El constructor de la clase `SoapClient` puede usarse de dos formas: indicando un documento `WSDL`, o sin indicarlo. En el primer caso, la extensión `SOAP` examina la definición del servicio y establece las opciones adecuadas para la comunicación, con lo cual el código necesario para utilizar un servicio es bastante simple.

En el segundo caso, si no indicas en el constructor un documento `WSDL` (bien porque no existe, o porque necesitas configurar manualmente alguna opción), el primer parámetro debe ser `null`, y las opciones para comunicarse con el servicio las tendrás que establecer en un **array** que se pasa como segundo parámetro.

Si el servicio dispone del correspondiente documento `WSDL`, en muchos lenguajes de programación existen utilidades que facilitan aún más el desarrollo de aplicaciones que lo utilicen. Nosotros vamos a ver la herramienta `WsdL2PhpGenerator`.

Se trata de un guion escrito en lenguaje `PHP` que examina un documento `WSDL` y genera un fichero `PHP` específico para comunicarse con el servicio web correspondiente. Su uso es muy sencillo: se ejecuta pasándole como parámetro la `URL` en que se encuentra el documento `WSDL`, y como resultado genera un fichero con código `PHP`.

Lo podemos instalar independientemente o con `"composer"`. Se recomienda hacerlo de esta última manera pues podemos indicar entre otras cosas el `"namespace"` para las clases que se creen. El enlace en **Packagist** es el siguiente: [wsdL2phpgenerator](https://packagist.org/packages/wsdL2phpgenerator/wsdL2phpgenerator).

La instalación es sencilla, en la carpeta donde vayamos a hacerlo nos creamos la estructura usual de un proyecto, las carpetas `"public"` y `"src"`. La carpeta `"public"` será para lo que vayamos a publicar y en `"src"` meteremos las clases.

Iniciamos **Composer** (el comando era `"composer init"`) integramos el `autoload` e instalamos la librería en cuestión, esto último también podemos hacerlo una vez generado el archivo: `"composer.json"` con el comando `"composer require wsdL2phpgenerator/wsdL2phpgenerator"`. Al final se nos habrá creado la carpeta `"vendor"` y nuestro archivo: `"composer.json"` deberá ser parecido a este:

```
{
  "name": "usuario/usuario",
  "description": "Prueba wsdL",
  "type": "project",
  "license": "gnu/gpl",
  "config": {
    "optimize-autoloader": true
  },
  "autoload": {
    "psr-4": {
      "Clases\\": "src"
    }
  },
  "authors": [
    {
      "name": "usuario",
      "email": "usuario@correo.es"
    }
  ],
}
```

```
"require": {
    "wsdl2phpgenerator/wsdl2phpgenerator": "^3.4"
}
```

En la carpeta "**public**" nos crearemos el archivo "**generar.php**" para generarnos las clases a partir del documento . (Una vez generadas deberíamos borrar este archivo) Si consultas la documentación del proyecto es fácil crear este archivo, fíjate en su contenido:

```
<?php
//utilizamos el autoload de composer
require '../vendor/autoload.php';

use Wsdl2PhpGenerator\Generator;
use Wsdl2PhpGenerator\Config;

$generator = new Generator();
$generator->generate(
    new Config([
        'inputFile' => "https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl", //wsdl
        'outputDir' => '../src', //directorio donde vamos a generar las clases
        'namespaceName' => 'Clases' //namespace que vamos a usar con ellas (lo indicamos en composer)
    ])
);
```

Podrás observar que ahora en "**src**" tenemos una serie de clases creadas. Ya estamos en condiciones de traernos los datos. Nos creamos el archivo "**titulaciones2019.php**" por ejemplo con el contenido siguiente:

```
<?php
require '../vendor/autoload.php';

use Clases\Pub_gestdocente;
use Clases\wstitulosuni;

$service = new Pub_gestdocente();
$request=new wstitulosuni('es', '2020');
$titulos=$service->wstitulosuni($request);

var_dump($titulos);
```

La diferencia es que ahora **\$titulos** es un **array** de objetos de la clase: "**ArrayOfClaseTitulosUni**" y no de la clase especial "**stdClass**".



Ejercicio resuelto

Inspecciona el servicio disponible en: "<http://webservices.gama-system.com/exchangerates.asmx?WSDL>", que ofrece información sobre el cambio de Euros a distintas divisas y viceversa. En el documento `wsdl` que puedes ver cuando pinchas en un enlace aparece una función "`ConvertToForeign`" que da el cambio del Euro a distintas divisas, en una fecha determinada y por un Banco (Esloveno) determinado. Investigando un poco en la página: "<http://webservices.gama-system.com/>" y con ayuda del traductor podemos investigar los parámetros. Se deja en retroalimentación un ayuda sobre este tema.

Con la ayuda de "`wsdl2PhpGenerator`" nos haremos un formulario donde una vez elegida la cantidad en **Euros**, la divisa a la que los cambiaremos, el tipo de operación (compra, venta, intermedio), el banco, de los disponibles, del que buscamos la información y la fecha, nos ofrecerá el cambio buscado o el error caso de haberlo.

Para simplificar ofreceremos el cambio para simplificar solo para tres divisas "Dolar USA", "Libra Esterlina" y "Yen Japonés", los códigos son respectivamente "`USD`", "`GBP`", "`JPY`".



Captura de pantalla Firefox (Elaboración propia)

Mostrar retroalimentación

Enlace donde se explica el tipo de dato esperado:

`"http://webservices.gama-system.com/descriptions-ER-slo.asp?strFunction=ConvertToForeign#ConvertToForeign"`

En este otro puedes ver los códigos de la monedas extranjeras (`strCurrency`) y la operaciones soportadas (`intRank`), con ayuda del traductor podemos observar que 1 es para la compra. 2 para el intermedio y 3 para la venta.

`"http://webservices.gama-system.com/descriptions-ER-slo.asp?strFunction=Common#Common"`

Échale un vistazo a la solución propuesta y compárala con la tuya:
[Solución propuesta](#). (zip - 0,73 MB)



Autoevaluación

Al utilizar la clase "SoapClient" para comunicarte con un servicio web:

- ☐ La herramienta `wsdl2phpgenerator` no es necesaria para obtener las opciones de configuración del mismo a partir del documento `wsdl`.
- ☐ Si el servicio dispone de una descripción en formato `WSDL`, puedes utilizar la herramienta `wsdl2phpgenerator` para no especificar a mano las opciones del mismo en la llamada al constructor de la clase `SoapClient`.

Correcta. El objetivo de la herramienta `wsdl2phpgenerator` no es simplificar la configuración del cliente, sino obtener una estructura de clases que facilite la utilización del servicio.

Incorrecta. Aunque la herramienta `wsdl2phpgenerator` facilita la utilización de un servicio web, si dispones del documento `WSDL` asociado al mismo, su función no es ayudar a configurar las opciones de utilización de éste en la clase `SoapClient`.

Solución

1. Opción correcta
2. Incorrecto

2.1.3.- Utilización de un servicio web (IV).

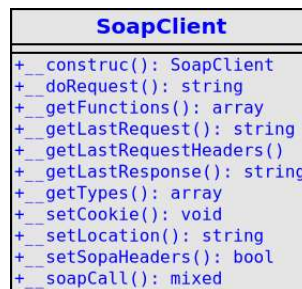


Diagrama DIA (Elaboración propia)

Si estás usando un documento WSDL para acceder al servicio web, la clase **SoapClient** implementa dos métodos que muestran parte de la información que contiene; concretamente, los tipos de datos definidos por el servicio, y las funciones que ofrece. Para conocer esta información, una vez creado el objeto, debes utilizar los métodos **__getTypes** y **__getFunctions** respectivamente.

```
<?php
$cliente = new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
$funciones = $cliente->__getFunctions();
echo "<ul>";
foreach ($funciones as $k => $v) {
    echo "<li><code>$v</code></li>";
}
echo "</ul>";
```

```
<?php
$cliente = new SoapClient("https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl");
$tipos = $cliente->__getTypes();
echo "<ul>";
foreach ($tipos as $k => $v) {
    echo "<li><code>$v</code></li>";
}
echo "</ul>";
```

El resultado obtenido es:

Para "**__getFunctions()**"



Captura de pantalla Firefox (Elaboración propia)

Para "__getTypes()"



Captura de pantalla Firefox (Elaboración Propia)

Donde las funciones aparecen duplicadas, dado que el servicio web, como ya vimos, ofrece dos versiones de la misma: una para **SOAP 1.1** y otra para **SOAP 1.2**.

La extensión "**PHP SOAP**" también incluye opciones de depuración muy útiles para averiguar qué está pasando cuando la conexión al servicio web no funciona como debería. Para habilitarlas, cuando hagas la llamada al constructor de la clase **SoapClient**, debes utilizar la opción **trace** en el array de opciones del segundo parámetro.

```
$cliente = new SoapClient(
    "https://cvnet.cpd.ua.es/servicioweb/publicos/pub_gestdocente.asmx?wsdl",
    array('trace'=>true)
);
```



Para saber más

Existen bastantes opciones que se pueden utilizar con el constructor "**SoapClient**", pero si el servicio web dispone de un documento **WSDL**, normalmente no necesitarás utilizar ninguna. En caso contrario deberás definir al menos las opciones "**location**" (la **URL** en la que se encuentra el servicio) y "**uri**" (la ruta (**target**) del espacio de nombres (**namespace**) del servicio **SOAP**)

[Constructor SoapClient.](#)



Reflexiona

Una vez activada la depuración, podrás utilizar los siguientes métodos para revisar los últimos mensajes SOAP enviados y recibidos.

Métodos para depuración de la clase SoapClient

Método	Significado
<code>__getLastRequest</code>	Devuelve el <u>XML</u> correspondiente a la última petición enviada.
<code>__getLastRequestHeaders</code>	Devuelve el <u>XML</u> correspondiente a los encabezados de la última petición enviada.
<code>__getLastResponse</code>	Devuelve el <u>XML</u> correspondiente a la última respuesta recibida.
<code>__getLastResponseHeaders</code>	Devuelve el <u>XML</u> correspondiente a los encabezados de la última respuesta recibida.

2.2.- Creación de un servicio web.

SoapServer
<div>+__construct(): SoapServer +addFunction(): void +addSoapHeader(): void +fault(): void +getFunctions(): array +handle(): void +setClass(): void +setObject(): void +setPersistence(): void</div>

DIA (Elaboración propia)

En **PHP SOAP**, para crear un servicio web, debes utilizar la clase **SoapServer**. Veamos un ejemplo sencillo. En una carpeta por ejemplo "**unidad6**" crearemos las carpetas "**clienteSoap**" y "**servidorSoap**" (lógicamente puedes usar los nombre que quieras). Dentro de "**servidorSoap**" crearemos el archivo "**servidor.php**" con el contenido siguiente:

```
<?php
class Operaciones
{
    public function resta($a, $b)
    {
        return $a - $b;
    }
    public function suma($a, $b)
    {
        return $a + $b;
    }
    public function saludo($texto)
    {
        return "Hola $texto";
    }
}
$uri='http://localhost/unidad6/servidorSoap';
$parametros=['uri'=>$uri];
try {
    $server = new SoapServer(NULL, $parametros);
    $server->setClass('Operaciones');
    $server->handle();
} catch (SoapFault $f) {
    die("error en server: " . $f->getMessage());
}
```

El código anterior crea un servicio web con tres funciones: **suma**, **resta** y **saludo**. Cada función recibe unos parámetros y devuelve un valor. Para añadir las funciones de la clase "**Operaciones**" a nuestro "**Servidor Soap**" fíjate que hemos añadido "**\$server->setClass('Operaciones')**". Si hubiésemos implementados las funciones directamente, sin usar una clase, tendríamos que haberlas añadido usando "**\$server->addFunction('nombre')**"

para cada función, antes de "\$server->handle()". El método `handle` es el encargado de procesar las peticiones, recogiendo los datos que se reciban utilizando **POST** por HTTP.

Para consumir este servicio, en la carpeta "`clienteSoap`" crearemos el fichero "`consumir.php`" necesitas escribir el siguiente código:

```
<?php
$url = 'http://localhost/unidad6/servidorSoap/servidor.php';
$uri = 'http://localhost/unidad6/servidorSoap';
$paramsaludo = ['texto' => "Manolo"];
$params = ['a' => 51, 'b' => 29];
try {
    $cliente = new SoapClient(null, ['location' => $url, 'uri' => $uri, 'trace'=>true]);
} catch (SoapFault $ex) {
    echo "Error: ".$ex->getMessage();
}
$saludo = $cliente->__soapCall('saludo', $paramsaludo);
$suma = $cliente->__soapCall('suma', $params);
$resta=$cliente->__soapCall('resta', $params);
echo $saludo. " La suma es: $suma y la resta es: $resta";
//También podríamos hacer
$saludo=$cliente->saludo("Manolo");
```

El servicio que has creado no incluye un documento WSDL (por eso al crear el servidor hemos puesto "`null`" como primer parámetro) para describir sus funciones. Sabes que existen los métodos `suma`, `resta` y `saludo`, y los parámetros que debes utilizar con ellos, porque conoces el código interno del servicio. Un usuario que no tuviera esta información, no sabría cómo consumir el servicio.

Fíjate que utilizamos el método mágico "`__soapCall()`", básicamente le paso el nombre del método que estoy llamando y los parámetros de dicho método en un array. Podíamos haber hecho directamente: "`$cliente->suma(4, 5)`", "`$cliente->resta(4,5)`", "`$cliente->saludo("Nombre")`".

Al igual que sucedía con `SoapClient` al programar un cliente, cuando utilizas `SoapServer` puedes crear un servicio sin documento WSDL asociado (como en el caso anterior, de nuevo "`null`" ha sido el primer parámetro), o indicar el documento WSDL correspondiente al servicio; pero antes deberás haberlo creado.

El primer parámetro del constructor indica la ubicación del WSDL correspondiente. El segundo parámetro es una colección de opciones de configuración del servicio. Si existe el primer parámetro, ya no hace falta más información. **PHP SOAP** utiliza la información del documento WSDL para ejecutar el servicio. Si, como en el ejemplo, no existe WSDL, deberás indicar en el segundo parámetro al menos la opción "`uri`", con el espacio de nombres destino del servicio.

En ambos casos, y sobre todo cuando estamos en un entorno de desarrollo, es recomendable usar un bloque "`try catch()`" para capturar posibles excepciones.



Para saber más

El constructor "SoapServer" permite indicar, además de `uri`, otras opciones en el segundo parámetro. Por ejemplo, la opción `soap_version` indica si se va a usar SOAP 1.1 o SOAP 1.2.

[Constructor SoapServer.](#)



Autoevaluación

Al utilizar la clase SoapServer para crear un servicio web:

- ☐ Si no creas y le asocias un documento `WSDL`, deberás indicar las opciones del mismo en la llamada al constructor "SoapServer".
- ☐ Debes indicar la ubicación del documento `WSDL` de descripción del servicio.

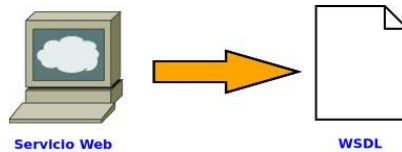
Correcta. Efectivamente. Si no se especifica como primer parámetro un documento `WSDL`, el segundo parámetro que recibe el constructor debe contener las opciones de configuración del servicio.

Incorrecta. No es obligatorio crear un documento `WSDL` asociado al servicio web, pero en este caso deberás indicar las opciones del mismo en la llamada al constructor.

Solución

1. Opción correcta
2. Incorrecto

2.2.1.- Creación de un servicio web (II).



Elaborado con DIA (Elaboración propia)

Para crear un documento WSDL de descripción del servicio, tendrás que seguir los pasos vistos anteriormente.

Al programar un servicio web, es importante cambiar en el fichero "php.ini" la directiva "soap.wsdl_cache_enabled" a "0". En caso contrario, con su valor por defecto ("1") los cambios que realices en los ficheros WSDL no tendrán efecto de forma inmediata.

El elemento raíz del documento será:

```
<definitions
  name="Operaciones"
  targetNamespace="http://localhost/unidad6/servidorSoap/servidor.php"
  xmlns:tns="http://localhost/unidad6/servidorSoap/servidor.php"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
>
```

En este caso no necesitas definir ningún tipo nuevo, por lo que no tendrás sección **types**. Los elementos **message** serán:

```
<message name="restaIn">
  <part name="a" type="xsd:float"/>
  <part name="b" type="xsd:float"/>
</message>
<message name="restaOut">
  <part name="return" type="xsd:float"/>
</message>
<message name="sumaIn">
  <part name="a" type="xsd:float"/>
  <part name="b" type="xsd:float"/>
</message>
```



```

<message name="sumaOut">
  <part name="return" type="xsd:float"/>
</message>
<message name="saludoIn">
  <part name="texto" type="xsd:string"/>
</message>
<message name="saludoOut">
  <part name="return" type="xsd:string"/>
</message>

```

El portType:

```

<portType name="OperacionesPort">
  <operation name="resta">
    <input message="tns:restaIn"/>
    <output message="tns:restaOut"/>
  </operation>
  <operation name="suma">
    <input message="tns:sumaIn"/>
    <output message="tns:sumaOut"/>
  </operation>
  <operation name="saludo">
    <input message="tns:saludoIn"/>
    <output message="tns:saludoOut"/>
  </operation>
</portType>

```

Suponiendo que el servicio web está en el fichero "servidor.php", dentro de las carpetas "servidorSoap" y "unidad6" la parte correspondiente al "binding" será:

```

<binding name="OperacionesBinding" type="tns:OperacionesPort">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="resta">
    <soap:operation soapAction="http://localhost/unidad6/servidorSoap/servidor.php#resta" />
    <input>
      <soap:body
        use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost/unidad6/ServidorSoap/servidor.php" />
    </input>
    <output>
      <soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost/unidad6/ServidorSoap/servidor.php" />
    </output>
  </operation>
  <operation name="suma">
    <soap:operation soapAction="http://localhost/unidad6/servidorSoap/servidor.php#suma" />
    <input>
      <soap:body
        use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

```

```

        namespace="http://localhost/unidad6/servidorSoap/servidor.php" />
    </input>
    <output>
        <soap:body
            use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://localhost/unidad6/servidorSoap/servidor.php" />
    </output>
</operation>
<operation name="saludo">
    <soap:operation soapAction="http://localhost/unidad6/servidorSoap/servidor.php#saludo" />
    <input>
        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://localhost/unidad6/servidorSoap/servidor.php" />
    </input>
    <output>
        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://localhost/unidad6/servidorSoap/servidor.php" />
    </output>
</operation>
</binding>

```

Y para finalizar, el elemento **service**:

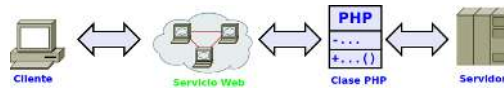
```

<service name="OperacionesService">
    <port name="OperacionesPort" binding="tns:OperacionesBinding">
        <soap:address location="http://localhost/unidad6/servidorSoap/servidor.php"/>
    </port>
</service>

```

Aunque el fichero parece extenso, fíjate que se repite casi lo mismo para cada función. En el apartado siguiente veremos que lo podemos generar automáticamente con una librería.

2.2.2.- Creación de un servicio web (III).



Elaborado con DIA (Elaboración propia.)

Vimos que es aconsejable definir una clase que implemente los métodos que queramos publicar en el servicio.

```
class Operaciones{
    public function resta($a, $b) {
        return $a - $b;
    }
    public function suma($a, $b){
        return $a + $b;
    }
    public function saludo($texto){
        return "Hola $texto";
    }
}
```

Y que al hacerlo de esta forma, en lugar de añadir una a una las funciones, podemos añadir la clase completa al servidor utilizando el método `setClass` de `SoapServer`. Lo ideal es que la clase con las funciones la implementemos en un archivo aparte y la llamemos con `"require"`.

```
require '../src/Operaciones.php';
$uri='http://localhost/unidad6/servidorSoap';
$parametros=['uri'=>$uri];
try {
    $server = new SoapServer(NULL, $parametros);
    $server->setClass('Operaciones');
    $server->handle();
} catch (SoapFault $f) {
    die("error en server: " . $f->getMessage());
}
```

En lugar de una **clase**, también es posible indicar un **objeto** para procesar las peticiones SOAP utilizando el método `"setObject"` de la clase `"SoapServer"`.

Aunque como ya sabes, **"PHP SOAP"** no genera el documento WSDL de forma automática para los servicios que crees, existen algunos mecanismos que nos permiten generarlo, aunque

siempre es aconsejable revisar los resultados obtenidos antes de publicarlos. Una de las formas más sencillas es utilizar la librería `php2wsdl`, disponible desde **Composer**.

[Librería `php2wsdl`.](#)

Esta librería revisa los comentarios que hayas añadido al código de la clase que quieras a publicar (debe ser una clase, no funciones aisladas), y genera como salida el documento WSDL correspondiente.

Para que funcione correctamente, es necesario que los comentarios de las clases sigan un formato específico: el mismo que utiliza la herramienta de documentación **PHPDocumentor**.



Para saber más

PHPDocumentor es una herramienta de código libre para generación automática de documentación, similar a **Javadoc** (para el lenguaje **Java**). Si comentamos el código de nuestras aplicaciones siguiendo unas normas, **PHPDocumentor** es capaz de generar, a partir de los comentarios que introduzcamos en el código mientras programamos, documentación en diversos formatos. En el enlace dejado puedes consultar la documentación.

Aunque esta herramienta está en **Packagist**, para instalarla nos recomiendan bajarnos el fichero PHAR correspondiente. Si vas a usar **PHPDocumentor** asegúrate de tener activada e instalada la extensión "`php-iconv`".

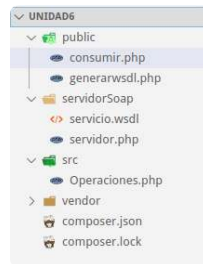
[PHPDocumentor.](#)

Para generar correctamente el documento WSDL con `php2wsdl` los métodos deben estar comentados siguiendo las reglas de **PHPDocumentor**. Los comentarios se deben ir introduciendo en el código distribuidos en bloques, y utilizando ciertas marcas específicas como "`@param`" para indicar un parámetro y "`@return`" para indicar el valor devuelto por una función, además para indicar los métodos que queremos que se generen en WSDL utilizaremos "`@soap`".

```
class Operaciones{
    /**
     * @soap
     * @param float $a
     * @param float $b
     * @return float
     */
    public function resta(float $a, float $b) :float{
        return $a - $b;
    }
    /**
     * @soap
     * @param float $a
     * @param float $b
```

```
    * @return float
    */
    public function suma($a, $b){
        return $a + $b;
    }
    /**
    * @soap
    * @param string $texto
    * @return string
    */
    public function saludo($texto){
        return "Hola $texto";
    }
}
```

2.2.3.- Creación de un servicio web (IV).



Recorte Captura
pantalla Visual Studio
Code (Elaboración
propia)

Para utilizar **Php2wsdl** lo integraremos a nuestro proyecto con **Composer**, además utilizaremos el **"autoload"** y espacios de nombre. Seguiremos la estructura de siempre, una carpeta **"src"** (**namespace** Clases) para las clases, la carpeta **"public"** donde estará **"consumir.php"** y la carpeta **"servidorSoap"** donde estará el archivo **"servidor.php"**.

Para integrarlo podemos hacerlo en el **"composer init"** buscando la dependencia o a posteriori con el comando: **"composer require php2wsdl/php2wsdl"**

Para generar el documento **WSDL** a partir de la clase **Operaciones** anterior, debes crear un nuevo fichero, por ejemplo hemos creado en la carpeta **"public"** el fichero **"generarwsdl.php"** con el siguiente código:

```
<?php
require '../vendor/autoload.php';

use PHP2WSDL\PHPClass2WSDL;

$class = "Clases\\Operaciones";
$uri = 'http://localhost/unidad6/servidorSoap/servidorW.php';
$wsdlGenerator = new PHPClass2WSDL($class, $uri);
$wsdlGenerator->generateWSDL(true);
$fichero = $wsdlGenerator->save('../servidorSoap/servicio.wsdl');
```

Es decir, crear un nuevo objeto de la clase **PHPClass2WSDL**, e indicar como parámetros:

- ✓ El nombre de la clase que gestionará las peticiones al servicio. Fíjate en **"\\"** la primera es para "escapar" la segunda en las comillas dobles.
- ✓ El **URI** en que se ofrece el servicio. Es importante que tengas en cuenta que si hemos generado el archivo **WSDL** es para usarlo, por lo que **"servidorW.php"** será el nombre de nuestro nuevo servidor **SOAP** que va a hacer uso del archivo (podemos reescribir si queremos **"servidor.php"**).

A continuación llamamos al método **"generateWSDL()"** con **"true"** esto es para que solo convierta los métodos públicos con el parámetro **"@soap"**

El método "save()" obtiene como salida el documento WSDL de descripción del servicio. Le indicamos nombre y ruta donde guardarlo, en la ruta elegida el Apache tiene que tener permiso de escritura, si no, no podrá crearlo.

Revisa el documento creado, pues posiblemente tengas que realizar algunos cambios (por ejemplo, pasar el formato de codificación a UTF-8, o cambiar el nombre de alguna de las clases que contiene y de su constructor respectivo).

Cuando esté listo, publícalo con tu servicio. Para ello, copia el fichero obtenido en una ruta accesible vía web, e indica la URL en que se encuentra cuando instancias la clase SoapServer. Nos creamos el archivo "servidorW.php" para utilizar el archivo WSDL, que acabamos de generar.

```
$server = new SoapServer("http://localhost/unidad6/servidorSoap/servicio.wsdl");
```

Si añades a la URL del servicio el parámetro GET wsdL, verás el fichero de descripción del servicio. En nuestro caso la URL sería: "http://localhost/unidad6/servidorSoap/servidorW.php?wsdl". Se deja de ejemplo el código de como quedaría el servidor y un archivo "consumir.php" donde vemos algunos ejemplos de como consumir los servicios:

"servidor.php"

```
<?php
require '../vendor/autoload.php';
$url = "http://localhost/unidad6/servidorSoap/servicio.wsdl";
try {
    $server = new SoapServer($url);
    $server->setClass('Clases\Operaciones');
    $server->handle();
} catch (SoapFault $f) {
    die("error en server: " . $f->getMessage());
}
```

"consumir.php"

```
<?php
$url = "http://localhost/unidad6/servidorSoap/servicio.wsdl";
try {
    $cliente = new SoapClient($url);
} catch (SoapFault $ex) {
    die("Error en el cliente: " . $ex->getMessage());
}
//Vemos las funciones que nos ofrece el servicio
var_dump($cliente->__getFunctions());
echo "<br>";
//una manera de llamar a suma
echo $cliente->suma(20, 40);
echo "<br>";
//otra forma de llamar a suma
$para = ['a' => 12, 'b' => 45];
echo $cliente->__soapCall('suma', $para);
```

En la variable "\$url" del archivo "consumir.php" podríamos haber puesto \$url="http://localhost/unidad6/servidorSoap/servidorW.php?wsdl" y hubiese funcionado igual.



Autoevaluación

Relaciona los términos siguientes con el concepto a que hacen referencia:

Ejercicio de relacionar

Elemento	Relación	Hace referencia a
wsdl2php	<input type="checkbox"/>	1. Herramienta para generar documentación, a partir de unas clases <u>PHP</u> adecuadamente documentadas.
PHPDocumentor	<input type="checkbox"/>	2. Método de la clase "Php2wsdl" que genera el documento <u>WSDL</u> .
php2wsdl	<input type="checkbox"/>	3. Herramienta para crear un documento <u>WSDL</u> , a partir de un servicio web programado como una clase <u>PHP</u> adecuadamente documentada.
save()	<input type="checkbox"/>	4. Herramienta para construir clases <u>PHP</u> a partir de un documento <u>WSDL</u> .

Enviar

Revisa los contenidos de la presente unidad y las herramientas que has utilizado en la misma.