WEB SERVICES CON PHP (desarrolloweb)

Introducción a los Web Services en PHP

El presente artículo esta dirigido a las personas que ya poseen conocimientos sobre Internet y programación y quieren comenzar a conocer el mundo de los web services (servicios web).

¿Cómo esta estructurado el artículo?

Comenzaremos dando una introducción a lo que son los Web Services (Servicios Web), luego definiremos los elementos que lo componen, y al finalizar veremos un ejemplo de cómo utilizar un web service por medio de NuSOAP, para el cual utilizaremos dos archivos PHP.

En el apéndice pueden encontrar el código de los dos archivos que consumen el web service para poder verlos en línea, o bajarlos comprimidos (ZIP). El archivo comprimido también provee de otros archivos de soporte.

Introducción:

Antes de comenzar explicando NuSoap, creo que es conveniente realizar una introducción a lo que son los Servicios Web, y para esto debemos comenzar con la definición de Web Service.

Web Service:

Es un sistema software diseñado para soportar la interoperabilidad máquina - máquina a través de una red. Este tiene una interfaz descripta en un formato que puede ser procesado por una máquina (específicamente WSDL, que veremos más adelante). Otros sistemas interactúan con el Web service utilizando mensajes SOAP los cuales se encuentran establecidos previamente.

Entonces podríamos decir que un Web Service es una comunicación por medio de mensajes SOAP (son mensajes especiales que más adelante veremos detenidamente) entre diferentes equipos a través de una red.

Ahora pasaremos a definir los elementos que componen a los Web Services, para luego ver como ellos estan interrelacionados.

¿Qué es XML, SOAP, WSDL, UDDI?

Para conocer cómo se realiza el intercambio de mensajes en los Web Services debemos primero saber cuales son los elementos fundamentales que lo componen. Estos son el XML, SOAP, WSDL, y UDDI.

XML - eXtensible Markup Language:

Es un subconjunto simplificado del SGML el cual fue diseñado principalmente para documentos Web. Deja a los diseñadores crear sus propias “etiquetas” o "tags" (Ej: <libro>), habilitando la definición, transmisión, validación, y la interpretación de datos entre aplicaciones y entre organizaciones. Un punto que considero que es importante aclarar es que el HTML y el XML tienen funciones diferentes. El HTML tiene por objeto

mostrar información, mientras que el XML se ocupa de la información propiamente dicha (el contenido). Este concepto es importante tenerlo en cuenta, ya que muchas personas al escuchar sobre XML piensan que es el sucesor de HTML.

Ejemplo de un documento XML sobre información de autos:

```
<?xml version="1.0" encoding="UTF-8"?>
<vehiculos>
  <coche>
      <marca>Toyota</marca>
      <modelo>Corolla</modelo>
      <fechaCompra>2002</fechaCompra>
  </coche>
  <coche>
      <marca>Honda</marca>
      <modelo>Civic</modelo>
      <fechaCompra>2003</fechaCompra>
  </coche>
</vehiculos>
```

Seguramente este tipo de archivos no sea una novedad para la mayoría de los que lean este artículo, ya que muchos documentos en Internet están en formato XML. Igualmente explicaremos brevemente las secciones del mismo:

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

Todo documento XML debe comenzar indicando que es un documento XML, la versión del mismo y su codificación. Es por eso que se utiliza el tag <?xml .... ?> . Todos los elementos a partir de aquí son definidos por el usuario.

<vehiculos>

Luego se indica un tag raiz (<vehiculos>) el cual contendrá a los demás elementos. Es como en HTML el tag <HTML> .. </HTML>, que dentro de el se encuentran los demás tags. Se lo conoce generalmente como root del documento.

<coche>

Luego definimos un tag <coche> el cual contendrá un coche en particular (en este caso solo incorpore 3 características de un coche: marca, modelo, y fecha de compra).

```
<marca>Toyota</marca>
<modelo>Corolla</modelo>
<fechaCompra>2002</fechaCompra>
```

Se definen 3 tags (marca, modelo, y fechaCompra), los cuales contienen los datos para un coche en particular.

```
<vehiculos>
  <coche>
    <marca>..</marca>
    <modelo>..</modelo>
    <fechaCompra>..</fechaCompra>
  </coche>
</vehiculos>
```

En el documento podemos apreciar que todos los tags que son abiertos <marca> deben ser cerrados </marca>. Esto es una exigencia del XML.

Se pueden anidar tags <coche><marca></marca>…..</coche>. Marca esta dentro del tag coche. El XML provee muchas otras posibilidades como puede ser que las etiquetas tengan atributos (<coche color="rojo">..</coche>), pero no nos detendremos en ellas ya que la idea es simplemente ofrecer los conceptos básicos para

conocer de que se trata el mundo de los Web Services.

SOAP. Simple Object Access Protocol

Es un protocolo que permite la comunicación entre aplicaciones a través de mensajes por medio de Internet. Es independiente de la plataforma, y del lenguaje. Esta basado en XML y es la base principal de los Web Services. Los mensajes SOAP son documento XML propiamente dicho, pero esto lo veremos más adelante cuando veamos un ejemplo de un mensaje SOAP.

Veamos como es la estructura básica del protocolo y la correspondiente explicación:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  Soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
      ...
    </soap:Header>
    <soap:Body>
      ...
      <soap:Fault>
        ...
      </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Explicación del código anterior:

<?xml version="1.0"?>

Como podemos ver en esta linea SOAP es un documento XML, y como tal, debe comenzar con el tag <?xml….?> y la versión correspondiente.

<soap:Envelope

Aquí se indica que comienza el envelope (sobre) del mensaje

xmlns:soap = "http://www.w3.org/2001/12/soap-envelope"

Un mensaje SOAP debe contener siempre un elemento envelope asociado con el namespace (espacio de nombres) http://www.w3.org/2001/12/soap-envelope

Soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

En esta línea lo que se hace es indicar donde se encuentran definidos los tipos de datos utilizados en el documento.

<soap:Header>

Esta línea indica el comienzo del Header (encabezado). En esta sección se incluye información específica del mensaje, como puede ser la autenticación.

</soap:Header>

Como todo documento XML los tags que son abiertos deben ser cerrados, esta línea indica la finalización del Header (encabezado).

<soap:Body>

Aquí comienza el cuerpo del mensaje, en esta sección se incorpora toda la información necesaria para el nodo final. Por ejemplo, los parámetros para la ejecución, o la respuesta a una petición.

<soap:Fault>

Cualquier tipo de fallo que se produzca será notificado en esta sección. La cual esta contenida dentro del cuerpo del mensaje.

</soap:Fault>

Cierre de la sección Fault.

</soap:Body>
Indica el final del cuerpo del mensaje.

</soap:Envelope>
Fin del mensaje SOAP.

WSDL y UDDI

WSDL - Web Services Description Language

Es un protocolo basado en XML que describe los accesos al Web Service. Podriamos decir que es el manual de operación del web service, porque nos indica cuales son las interfaces que provee el Servicio web y los tipos de datos necesarios para la utilización del mismo.

Veamos un ejemplo de un documento WSDL:

```
<?xml version="1.0">
<definitions>
   <types>
    ...
   </types>
   <message>
    ...
   </message>
   <portType>

    ...
   </portType>
   <binding>
    ...
   </binding>
</definitions>
```

Explicación del código anterior:

<?xml version="1.0">	Este es otro documento XML, es por esto que debe comenzar con el tag <?xml .. ?>
<definitions>	Comienzo del documento, este tag agrupa a todos los demás.
<types>	Se definen los tipos de datos utilizados en el Web Service.
</types>	Fin de la definición de tipos.
<message>	Se definen los métodos y parámetros para realizar la operación. Cada message puede consistir en una o más partes (parámetros).
</message>	Fin de la definición de los parámetros.
<portType>	Esta sección es la más importante, ya que se definen las operaciones que pueden ser realizadas, y los mensajes que involucran (por ejemplo el mensaje de petición y el de respuesta).
</portType>	Fin de la definición de las operaciones y mensajes.
<binding>	Se definen el formato del mensaje y detalles del protocolo para cada portType.
</binding>	Fin de la definición del formato del mensaje y detalles del protocolo para cada PortType.

</definitions> Fin del documento WSDL

UDDI - Universal Discovery Description and Integration

Es un modelo de directorios para Web Services. Es una especificación para mantener directorios estandarizados de información acerca de los Web Services, sus capacidades, ubicación, y requerimientos en un formato reconocido universalmente. UDDI utiliza WSDL para describir las interfaces de los Web Services.
Es un lugar en el cual podemos buscar cuales son los Servicios web disponibles, una especie de directorio en el cual podemos encontrar los Web Services publicados y publicar los Web Services que desarrollemos.

http://soapclient.com/uddisearch.html
http://uddi.org/find.html

Historia de los Web Services

Los Servicios Web surgieron ante una necesidad de estandarizar la comunicación entre distintas plataformas (PC, Mainframe, Mac, etc.) y lenguajes de programación (PHP, C#, Java, etc.).

Anteriormente se habían realizado intentos de crear estándares pero fracasaron o no tuvieron el suficiente éxito, algunos de ellos son DCOM y CORBA, por ser dependientes de la implementación del vendedor DCOM - Microsoft, y CORBA - ORB (a pesar que CORBA de múltiples vendedores pueden operar entre si, hay ciertas limitaciones para aplicaciones de niveles más altos en los cuales se necesite seguridad o administración de transacciones).

Otro gran problema es que se hacía uso de RPC (Remote Procedure Call) para realizar la comunicación entre diferentes nodos. Esto, además de presentar ciertos problemas de seguridad, tiene la desventaja de que su implementación en un ambiente como es Internet, es casi imposible (muchos firewalls bloquean este tipo de mensajes, lo que hace prácticamente imposible a dos computadoras conectadas por Internet comunicarse).

Los Web Services surgieron para finalmente poder lograr la tan esperada comunicación entre diferentes plataformas. En la actualidad muchos sistemas legacy están pasando a ser web services.

Es por esto que en 1999 se comenzó a plantear un nuevo estándar, el cual terminaría utilizando XML, SOAP, WSDL, y UDDI.

¿Los Web services pueden ser solo utilizados con HTTP?

A pesar de mucho limitar el uso de los Web services al protocolo HTTP, los Web services no fueron pensados para un protocolo en particular, es decir, nada nos impide utilizar SOAP sobre algún otro protocolo de Internet (SMTP, FTP, etc.).

Se utiliza principalmente HTTP por ser un protocolo ampliamente difundido y que se encuentra menos restringido por firewalls (generalmente se bloquean puertos como el FTP, pero el HTTP es muy probable que no este bloqueado).

Comenzamos a utilizar NuSOAP

¿Este no era un artículo de NuSOAP?

Si, este es un artículo de como constuir Web Services desde PHP con NuSOAP, pero antes debía dar una introducción a lo que son los Web services y cuales son las partes que los componen.

Creo que siempre es interesante y necesario poder saber que hay detrás de bambalinas, para poder aprovechar al máximo las herramientas, o productos, o servicios, etc.

¿Que es NuSOAP?

NuSOAP es un kit de herramientas (ToolKit) para desarrollar Web Services bajo el lenguaje PHP. Está compuesto por una serie de clases que nos harán mucho más fácil el desarrollo de Web Services. Provee soporte para el desarrollo de clientes (aquellos que consumen los Web Services) y de servidores (aquellos que los proveen). NuSOAP está basado en SOAP 1.1, WSDL 1.1 y HTTP 1.0/1.1

¿NuSOAP es el único soporte para Web Services en PHP?

No, no es el único, existen otros, pero es uno de los que están en una fase de desarrollo mucho más avanzada. Sin ir más lejos, PHP a partir de su versión 5 comienza a dar soporte para SOAP, pero aún está en fase experimental.

¿Por qué NuSOAP y no otro?

1. Está en una fase madura de desarrollo.
2. No necesita módulos adicionales.
3. Es muy fácil su instalación y uso

¿Cómo instalo NuSOAP?

La instalación es bastante sencilla, sólo basta ir a la pagina en sourceforge de NuSOAP http://sourceforge.net/projects/nusoap/ y bajar el archivo comprimido (es un .zip).

Lo descomprimimos en un directorio de nuestro servidor web (como puede ser /lib que es el directorio por default), y listo, ya podemos hacer uso de NuSOAP.

Comenzamos a utilizar NuSOAP II

¿Como lo usamos?

Para responder a esta pregunta no hay nada mejor que un ejemplo de utilización de NuSOAP, el cual nos ayudará también a terminar de ver los conceptos mencionados anteriormente (SOAP, WSDL, etc.).

Decidí ir a la página de XMethods y buscar algún servicio que pueda ser aplicable por

personas de todo el mundo, y que no esté limitado a los Estados Unidos. Buscando entre gran cantidad de Web Services de Cotizaciones, o datos específicos de los Estados Unidos, encontré uno que podía ser implementado en cualquier país, y era ni más ni menos que los datos del clima para todo el mundo. Este Web Service tiene 9 métodos que podemos utilizar (todos ellos se encuentran definidos en el archivo WSDL que se encuentra en http://live.capescience.com/wsdl/GlobalWeather.wsdl. Los métodos son: getStation(), isValidCode(), listCountries(), searchByCode(), searchByCountry(), searchByLocation(), searchByName(), searchByRegion(), getWeatherReport().

Para el ejemplo solo utilizare el método searchByCountry() y getWeatherReport().

En el ejemplo existen dos archivos:

seleccionarLocalidad.php: el cual nos permitirá seleccionar una localidad dado un país (el país esta dado por el valor de la variable $sPais). Este archivo PHP utiliza el método del Web Service searchByCountry()

mostrarPronostico.php: el cual mostrara el pronóstico para la localidad seleccionada. Este archivo PHP utiliza el método del Web Service getWeatherReport().

Si queremos ver un listado de los servicios publicados podemos ir a esta página: CapeScience

En esta página nos permiten probar los métodos de los Web Services en línea, sin necesidad de un cliente SOAP, esto nos es de utilidad principalmente para ver como funciona cada uno de los métodos.

Como mencione anteriormente, toda la información del Web Service se encuentra definida dentro del documento WSDL, para este Web service se encuentra definida en CapeScience.

Este documento en particular tiene 425 líneas, por lo que la lectura del mismo es bastante tediosa.

No voy a explicar todo el archivo WSDL, sino que simplemente me detendré en las líneas que son de importancia para nuestra aplicación (las que hacen referencia a los metodos searchByCountry() y getWeatherReport()). Ahora veamos donde se encuentran las definiciones de los métodos dentro del archivo:

Mensajes de searchByCountry: entre las lineas 277 a 282 podemos apreciar que se encuentra lo siguiente:

```
277. <message name="searchByCountryResponse">
278. <part name="return" type="xsd1:ArrayOfStation"/>
279. </message>
280. <message name="searchByCountry">
281. <part name="country" type="xsd:string"/>
282. </message>
```

SearchByCountry: comencemos por la línea 280 y dejaremos para después las líneas 277 a 279.

```
280. <message name="searchByCountry">
```

En esta línea se indica que existe un método llamado searchByCountry, que es el

método que estábamos buscando.

```
281. <part name="country" type="xsd:string"/>
```
Aquí se define que existe un parámetro para dicho método que se llama "country" (país) y es del tipo string.

```
282. </message>
```
Fin del método seachByCountry

Ejemplo de un mensaje real con el parámetro country seteado en "argentina":

```
<nu:searchByCountry>
  <country xsi:type="xsd:string">argentina</country>
</nu:searchByCountry>
```

SearchByCountryResponse: Ahora analicemos el método searchByCountryResponse, como su nombre indica, es la respuesta al metodo searchByCountry. Es decir cuando invocamos a searchByCountry, el servidor nos responderá con un mensaje con el formato de searchByCountryResponse. Las respuestas, generalmente, tienen anexado al final Response al nombre del método (esto se epecifica en el portType).

```
277. <message name="searchByCountryResponse">
```
Esta línea nos indica que es una respuesta al método searchByCountry.

```
278. <part name="return" type="xsd1:ArrayOfStation"/>
```
El método nos devuelve una variable "return" la cual es del tipo ArrayOfSation. Como vimos anteriormente, los documentos WSDL pueden definir sus propios tipos de datos, ArrayOfStation es un tipo de datos definido en este WSDL. Ver más adelante la definición del tipo de datos ArrayOfStation.

```
279. </message>
```
Fin del método searchByCountryResponse.

Como vimos anteriormente se hace referencia a un ArrayOfStation, para ver de que se trata esto debemos buscar este tipo de datos dentro del archivo WSDL. En las líneas comprendidas ente 148 y 157, podemos ver que se define este tipo de datos.

```
148. <xsd:complexType name="ArrayOfStation">
149.   <xsd:complexContent>
150.     <xsd:restriction base="SOAP-ENC:Array">
151.       <xsd:sequence>
152.         <xsd:element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd1:Station"/>
153.       </xsd:sequence>
154.       <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd1:Station[]"/>
155.     </xsd:restriction>
156.   </xsd:complexContent>
157. </xsd:complexType>
```

```
148. <xsd:complexType name="ArrayOfStation">
```
En esta línea podemos ver que se define un tipo de datos complejo (complexType) con el nombre de "ArrayOfStation", el tipo de datos que buscábamos.

```
149. <xsd:complexContent>
150. <xsd:restriction base="SOAP-ENC:Array">
151. <xsd:sequence>
```
Indicamos que es un elemento complejo (array).

```
152. <xsd:element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd1:Station"/>
```

Esta línea es la que más nos interesa, ya que se indica precisamente como esta compuesto el array. Nos dice que tiene un mínimo de ocurrencias de 0 (MinOccurs=" 0"), es decir, que puede no tener elementos. Un máximo de ocurrencias no limitado (maxOccurs="unbounded"), es decir que podemos tener todos los elementos que queramos. Y al finalizar nos indica el tipo de datos del array, que es "Station". Por desgracia ahora tenemos que ver que tipo de datos es Station. Como podemos ver un dato se define a partir de otro, hasta llegar a los tipos de datos básicos.

153. </xsd:sequence>
Se cierran los tags anteriormente abiertos.

154. <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd1:Station[]"/>
Terminamos de definir el array.

155. </xsd:restriction> 156. </xsd:complexContent> 157. </xsd:complexType>
Se cierran los demás tags que fueron abiertos.

Nos quedaba pendiente definir el tipo de datos "Station", nuevamente, al buscar en el código encontramos entre las líneas 134 a 147 la definición de este tipo de datos.

```
134.   <xsd:complexType name="Station">
135.     <xsd:sequence>
136.       <xsd:element maxOccurs="1" minOccurs="1" name="icao" nillable="true" type="xsd:string"/>
137.     <   xsd:element maxOccurs="1" minOccurs="1" name="wmo" nillable="true" type="xsd:string"/>
138.     <   xsd:element maxOccurs="1" minOccurs="1" name="iata" nillable="true" type="xsd:string"/>
139.     <   xsd:element maxOccurs="1" minOccurs="1" name="elevation" type="xsd:double"/>
140.     <   xsd:element maxOccurs="1" minOccurs="1" name="latitude" type="xsd:double"/>
141.     <   xsd:element maxOccurs="1" minOccurs="1" name="longitude" type="xsd:double"/>
142.     <   xsd:element maxOccurs="1" minOccurs="1" name="name" nillable="true" type="xsd:string"/>
143.     <   xsd:element maxOccurs="1" minOccurs="1" name="region" nillable="true" type="xsd:string"/>
144.     <   xsd:element maxOccurs="1" minOccurs="1" name="country" nillable="true"
type="xsd:string"/>
145.       <xsd:element maxOccurs="1" minOccurs="1" name="string" nillable="true" type="xsd:string"/>
146.     </xsd:sequence>
147.   </xsd:complexType>
```

134. <xsd:complexType name="Station">
Se define que comienza el tipo de datos complejo Station.

135. <xsd:sequence>
Indicamos que comenzamos con la definición del tipo secuencia (es decir que todos los elementos que se definen dentro están incluidos en el tipo de datos Station). Existen otro tipo que no es secuencia (sequence), y se llama choice (elección), en el cual se selecciona entre uno de los elementos dentro definidos (aquellos comprendidos entre <xsd:choice>y </xsd:choice>).

136. <xsd:element maxOccurs="1" minOccurs="1" name="icao" nillable="true" type="xsd:string"/>
137. <xsd:element maxOccurs="1" minOccurs="1" name="wmo" nillable="true" type="xsd:string"/>
138. <xsd:element maxOccurs="1" minOccurs="1" name="iata" nillable="true" type="xsd:string"/>
Se definen 3 elementos los cuales son del tipo string, dichos elementos son: icao, wmo, e iata.

139. <xsd:element maxOccurs="1" minOccurs="1" name="elevation" type="xsd:double"/>
140. <xsd:element maxOccurs="1" minOccurs="1" name="latitude" type="xsd:double"/>
141. <xsd:element maxOccurs="1" minOccurs="1" name="longitude" type="xsd:double"/>
Se definen otros 3 elementos, en este caso son del tipo double. Estos son: elevation, latitude, y longitude.

```
142. <xsd:element maxOccurs="1" minOccurs="1" name="name" nillable="true" type="xsd:string"/>
143. <xsd:element maxOccurs="1" minOccurs="1" name="region" nillable="true" type="xsd:string"/>
144. <xsd:element maxOccurs="1" minOccurs="1" name="country" nillable="true" type="xsd:string"/>
145. <xsd:element maxOccurs="1" minOccurs="1" name="string" nillable="true" type="xsd:string"/>
```

Se definen los últimos 4 elementos del tipo string. Los elementos aquí definidos son: name, region, country, y string.

```
146. </xsd:sequence>
147. </xsd:complexType>
```

Cierre de los tags abiertos anteriormente.

Veamos un ejemplo de respuesta:

```
<return xsi:type="cc1:ArrayOfStation" SOAP-ENC:arrayType="cc1:Station[86]">
  <item xsi:type="cc1:Station">
    <icao xsi:type="xsd:string">SARS</icao>
    <wmo xsi:type="xsd:string">87149</wmo>
    <iata xsi:nil="true"/>
    <elevation xsi:type="xsd:double">91.0</elevation>
    <latitude xsi:type="xsd:double">-26.817</latitude>
    <longitude xsi:type="xsd:double">-60.45</longitude>
    <name xsi:type="xsd:string">Presidencia Roque Saenz Pena Aerodrome</name>
    <region xsi:nil="true"/>
    <country xsi:type="xsd:string">
    Argentina </country>
    <string xsi:type="xsd:string">SARS - Presidencia Roque Saenz Pena Aerodrome, Argentina
    @ -26.817&amp;apos;S -60.45&amp;apos;W 91m</string>
    </item>
  <item xsi:type="cc1:Station">
</return>
```

Para aclarar el tipo de estructura de ArrayOfStation usaremos analogías con otros lenguajes de programación:

// Delphi //

```
Station = record
icao : String;
wmo : String;
iata : String;
elevation : String;
latitude : String;
longitude : String;
name : String;
region : String;
country : String;
string_ : String;
end;
ArrayOfSation : array of Station;
```

// C# //

```
struct Station
{
String icao;
String wmo;
String iata;
String elevation;
String latitude;
String longitude;
String name;
String region;
String country;
```

```
String string_;
}
Station[] ArrayOfStation;
```

Como vemos no es fácil seguir todo lo que esta definido dentro de un documento WSDL, por suerte muchas veces no nos tenemos que preocupar por todo esto ya que los mismos sitios donde se encuentran alojados los Web Services nos proveen de la información necesaria sin necesidad de tener que recurrir a ver los archivos WSDL. También existen aplicaciones como son Altova XMLSPY que nos proveen de la información de manera gráfica

Utilizando NuSOAP III

Comenzamos a programar!

Como mencione anteriormente he dividido la aplicación de prueba del Web Service en dos (2) archivos, uno que mostrara todas las localidades donde existen pronósticos (seleccionarLocalidad.php) y otro que mostrara el pronóstico para la localidad seleccionada (mostrarPronostico.php)

Veamos como consumimos un web service (searchByCountry):

En el código se han eliminado las comprobaciones de errores para poder facilitar la lectura de los pasos. Veamos cuales son los pasos para consumir un servicio web desde PHP:

```
require_once("lib/nusoap.php");
```

Debemos incluir la librería NuSOAP. En este ejemplo asumimos que el directorio donde se encuentra el archivo nusoap.php esta ubicado en el subdirectorio lib (a partir de donde esta nuestro script).

```
$oSoapClient = new soapclient('http://live.capescience.com/wsdl/GlobalWeather.wsdl', true);
```

Debemos instanciar la clase soapclient, ya que en esta ocasión utilizaremos solo el cliente que nos provee NuSOAP. Los parámetros enviados son la ubicación del documento WSDL y true como segundo parametro (este último parámetro indica que el primer documento es un WSDL) .

Definimos un array con el país para el cual se quieren obtener las localidades que tienen pronósticos disponibles. Volviendo al documento WSDL podemos ver que la variable que debemos enviar es "country" y que es del tipo string

```
$aParametros = array("country" => "argentina");

280. <message name="searchByCountry">

281. <part name="country" type="xsd:string"/>

282. </message>

$aRespuesta = $oSoapClient->call("searchByCountry", $aParametros);
```

Llamamos al método call del objeto soapclient. Al mismo le pasamos como parámetro

el nombre del método que queremos ejecutar en el web service (searchByCountry) y los parámetros ($aParametros). En nuestra variable $aRespuesta tenemos un array con la respuesta del Web Service. Este array tiene el formato descripto en "ArrayOfStation" (el cual fue visto anteriormente). Es decir podemos acceder a las variables descriptas en ArrayOfStation de la siguiente manera: $aRespuesta["wmo"], $aRespuesta["elevation"], $aRespuesta["name"], etc.

Ahora que ya conocemos c&oacte;mo sería el procedimiento para consumir un Web Service, veamos un poco más en profundidad los métodos de soapclient que utilizamos:

Al crear una instancia de soapclient, existen varios parámetros que podemos enviar para su creación:

```
$oSoapClient = new soapclient(

  <url donde se encuentra el web service o WSDL>,
  [<booleana indicando si el primer parámetro es un WSDL>],
  [<entero con el Portéame>],
  [<cadena proxyHost>],
  [<cadena proxyPort>],
  [<cadena nombre de usuario>],
  [<cadena password>],
  [<entero con el timeout de la conexión>],
  [<entero con el timeout de la respuesta>]);
```

Solo el primer parámetro es necesario (la ubicación del Web Service o del documento WSDL del Web Service), todos los demás son opcionales.

Al ejecutar el Web Service, existen varios parámetros que podemos enviar al método:

```
call( <método que queremos ejecutar en el servidor>,

  [<un array asociativo con los parámetros que debemos enviar (si existen)>],
  [<cadena con el espacio de nombres (namespace)>],
  [<cadena con el valor de la acción SOAP>],
  [<booleana indicando si esta presente los valores de SOAPVAL en los headers>],
  [<booleana la cual ya no se utiliza>],
  [<cadena con el style a usar cuando se realiza la serialización de los parametros>],
  [<cadena que puede ser "encoded" o "literal" utilizada para serialización de los parametros>]);
```

Cómo vemos existen muchos más parametros de los que habiamos utilizado, pero como exigen un conocimiento más profundo sobre los Web Services los cuales escapan a los alcances de este artículo no los trateremos aquí.

Utilizando NuSOAP IV

¿Cómo lo podemos usar?:

Existen muchas formas en que podríamos usar este Web Service, podemos dejar que el visitante seleccione su país y localidad, y que nuestro sitio le retorne cual es el pronostico actual (solo deberíamos incorporar un paso anterior al que se ofrece en el código, el de selección del país).

Otro uso que podemos darle es el de mostrar cual es el clima actual para una localidad determinada (lo mismo que hacen muchos diarios en sus cabeceras). Si decidimos

implementar la segunda opción debemos tener especial cuidado en como lo hacemos, ya que si por cada petición que haga el usuario debemos conectarnos con el servidor que ofrece el web service, esto provocaría una gran demora para el usuario, y una sobrecarga innecesaria del servidor. Para solucionar esto podemos guardar en un archivo de texto o en una base de datos, el clima actual, para leerlo desde allí. Este archivo, o base de datos, la actualizaríamos cada un tiempo prudencial (como puede ser una hora o más).

Otros Web Services:

Actualmente existen gran cantidad de Web Services, gratuitos y no. En el sitio de XMethods pueden encontrar gran cantidad de ellos. Sin ir más lejos Google ofrece un web service para realizar búsquedas (solo hay que registrarse en http://www.google.com/apis/) y nos permitirá realizar 1000 consultas por día (creo que puede ser más que interesante para muchos).

Conclusión:

Los Web Services son un tema apasionante, ya que brindan gran funcionalidad y posibilidad de realizar cambios y mejorar nuestras aplicaciones. Un mismo web service puede ser consumido tanto por aplicaciones que se ejecutan en un servidor y entregan código html al cliente, como por aplicaciones que corren directamente en la máquina del cliente. Como vemos las posibilidades son infinitas, es sólo cuestión de comenzar a experimentar.

Apéndice:

Código de seleccionarLocalidad.php

```
/**
* Codigo para consumir un servicio web (Web Service) por medio de NuSoap.
* La distribucion del codigo es totalmente gratuita y no tiene ningun tipo de restriccion.
* Se agradece que mantengan la fuente del mismo.

*/ $sPais = "argentina"; // Nombre del pais que queremos el listado de localidades

// Inclusion de la libreria nusoap (la que contendra toda la conexión con el servidor //
require_once('lib/nusoap.php');

$oSoapClient = new soapclient('http://live.capescience.com/wsdl/GlobalWeather.wsdl', true);

if ($sError = $oSoapClient->getError()) {
echo "No se pudo realizar la operación [" . $sError . "]";
die();
}
$aParametros = array("country" => $sPais);
$respuesta = $oSoapClient->call("searchByCountry", $aParametros);

  // Existe alguna falla en el servicio? if ($oSoapClient->fault) { // Si
echo 'No se pudo completar la operación';
die();
} else { // No
$sError = $oSoapClient->getError();
// Hay algun error ?
if ($sError) { // Si
echo 'Error:' . $sError;
die();
}
```

```
}
?>
<html>
  <body>
    <form action="mostrarPronostico.php" method="post" name="frmLocalidades" id="frmLocalidades">
      <table width="400" border="0" cellspacing="0" cellpadding="0">
    <tr>
      <td colspan="2"><div align="center">Seleccione una localidad</div></td>
    </tr>
    <tr>
      <td width="61"> </td>
      <td width="339"> </td>
    </tr>
    <tr>
      <td>Localidad:</td>
      <td><select name="codLocalidad" id="codLocalidad">
$aElemento)
    echo "<option value='".$aElemento["wmo"]."'>".$aElemento["name"]."</option>";
?>
      </select></td>
    </tr>
    <tr>
      <td> </td>
      <td> </td>
    </tr>
    <tr>
        /td>
        td><input type="submit" name="Submit" value="Quiero ver el pronostico"></td>
      /tr>
      /table>
    /form>
  /body>
</html>
```

Utilizando NuSOAP V

Código de mostrarPronostico.php

```
<?php

/**
* Codigo para consumir un servicio web (Web Service) por medio de NuSoap
* La distribucion del codigo es totalmente gratuita y no tiene ningun tipo de restriccion.
* Se agradece que mantengan la fuente del mismo.
*/
// Le indicamos a PHP que no muestre los Notices (por si el servicio no retorna datos)
// (esto se puede evitar preguntando si hay datos antes de mostrarlos)
error_reporting(1);

// Inclusion de la libreria nusoap (la que contendra toda la conexión con el servidor
//require_once('lib/nusoap.php');
$oSoapClient = new soapclient('http://live.capescience.com/wsdl/GlobalWeather.wsdl', true);
if ($sError = $oSoapClient->getError()) {
  echo "No se pudo realizar la operación [" . $sError . "]";
  die();
}

// Viene de un POST => Selecciono una ciudad
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $aParametros = array("code" => $_POST["codLocalidad"]);
  $aRespuesta = $oSoapClient->call("getWeatherReport", $aParametros);
}
  // Existe alguna falla en el servicio?
```

```
  if ($oSoapClient->fault) { // Si
    echo 'No se pudo completar la operación';
    die();
  } else { // No
    $sError = $oSoapClient->getError();
    // Hay algun error ?
    if ($sError) { // Si
      echo 'Error:' . $sError;
    }
}
?
><html>
  <body>
    <table width="367" border="0" cellspacing="0" cellpadding="0">
    <tr>
      <td colspan="2"><div align="center">Datos del tiempo</div></td>
    </tr>
    <tr>
      <td width="147"> </td>
      <td width="220"> </td>
    </tr>
    <tr>
      <td>Nombre:</td>
      <td><?=$aRespuesta["station"]["name"];?></td>
    </tr>
    <tr>
      <td>Elevación:</td>
      <td><?=$aRespuesta["station"]["elevation"]; ?></td>
    </tr>
    <tr>
      <td>Fecha y Hora:</td>
      <td><?=$aRespuesta["timestamp"];?></td>
    </tr>
    <tr>
      <td>País:</td>
      <td><?=$aRespuesta["station"]["country"];?></td>
    </tr>
   <tr>
      <td>Latitud:</td>
      <td><?=$aRespuesta["station"]["latitude"];?></td>
    </tr>
      <tr>
    <td>Longitud:</td>
      <td><?=$aRespuesta["station"]["longitude"];?></td>
    </tr>
    <tr>
      <td>Presión:</td>
      <td><?=$aRespuesta["pressure"]["string"];?></td>
    </tr>
    <tr>
      <td>Temperatura:</td>
      <td><?=$aRespuesta["temperature"]["string"];?></td>
  <   /tr>
    <tr>
      <td>Visibilidad:</td>
      <td><?=$aRespuesta["visibility"]["distance"];?> mts.</td>
    </tr>
    </table>
  </body>
</html>
```

Autores del manual:

Hay que agradecer a diversas personas la dedicación prestada para la creación de este manual. Sus nombres junto con el número de artículos redactados por cada uno son los siguientes:

- **Orlando Fabián Brea**
  http://www.smartsol.com.ar
  (9 capítulos)