Using the Google Web APIs with PHP

By Harish Kamath

Table of Contents

Getting High

Unless you've been living in a cave for the past couple of years, you already know what Google is. In fact, if you're anything like most Webheads, you've probably used it in the last sixty minutes. With its unique indexing technology and huge database of Web pages, Google has rapidly become the best search engine on the Web, bar none − and the company's constant innovation promises to keep them ahead of the pack for a while longer.

Easily one of the most interesting things to emerge from Google HQ in recent months has to be the Google Web APIs, which allow developers to query the complete Google database using a series of SOAP−based remote procedure calls. By making a portion of Google's cutting−edge technology available to the public, the Google Web APIs allow developers to get ever more creative with their Web−based applications, and enable them to push the envelope with new XML−based applications.

Now, your favourite language and mine, PHP, comes with a couple of classes designed specifically for SOAP−based remote procedure calls over HTTP. This makes PHP ideal for developers looking to integrate the Google Web APIs into their Web applications. The only problem? Not too many people know how to do it.

That's where this tutorial comes in. Over the next few pages, I'll be demonstrating how you can use PHP, in combination with the Google Web APIs, to add powerful search capabilities to your Web application. Don't get too close − this is potent stuff.

Remote Control

Before we get into the code, though, you need to have a clear understanding of how the Google Web APIs work. This involves getting up close and personal with a complicated little critter known as SOAP, the Simple Object Access Protocol.

According to the primer available on the W3C's site (http://www.w3.org/2002/ws/), SOAP is a "lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol at the core of which is an envelope that defines a framework for describing what is in a message and how to process it and a transport binding framework for exchanging messages using an underlying protocol."

If you're anything like me, that probably made very little sense to you. So here's the Reader's Digest version, which is far more cogent: "SOAP is a simple XML based protocol to let applications exchange information over HTTP." (http://www.w3schools.com/soap/soap_intro.asp)

SOAP is a client−server paradigm which builds on existing Internet technologies to simplify the task of invoking procedures and accessing objects across a network. It uses XML to encode procedure invocations (and decode procedure responses) into a package suitable for transmission across a network, and HTTP to actually perform the transmission.

At one end of the connection, a SOAP server receives SOAP requests containing procedure calls, decodes them, invokes the function and packages the response into a SOAP packet suitable for retransmission back to the requesting client. The client can then decode the response and use the results of the procedure invocation in whatever manner it chooses. The entire process is fairly streamlined and, because of its reliance on existing standards, relatively easy to understand and use.

Here's a quick example of what a SOAP request for the procedure getFlavourOfTheDay() might look like:

```
<?xml version="1.0" ?>
<SOAP−ENV:Envelope
xmlns:SOAP−ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xmlns:SOAP−ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:ns6="http://testuri.org"
SOAP−ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP−ENV:Body>
<ns6:getFlavourOfTheDay>
<day xsi:type="xsd:string">monday</day>
</ns6:getFlavourOfTheDay>
</SOAP−ENV:Body>
</SOAP−ENV:Envelope>
```

And here's what the response might look like:

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:ns6="http://testuri.org"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns6:getFlavourOfTheDayResponse>
<flavour xsi:type="xsd:string">pineapple</flavour>
</ns6:getFlavourOfTheDayResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I'm not going to get into the details of how SOAP works in this article, preferring instead to focus on how SOAP can be exploited in the context of PHP and the Google Web APIs. If you're new to SOAP, the information above should be sufficient to explain the basic concept and ensure that you can follow the material that comes next; however, if you're interested in learning more about SOAP (or if you just have trouble falling asleep at night), you should read the W3C's specifications on the protocol − links are included at the end of this article.

The Bare Necessities

There are a couple of things you'll need before you can get started with PHP and the Google Web APIs. Obviously, you need a working build of PHP − I recommend the latest version, PHP 4.2.1, which you can download from http://www.php.net/

You'll also need an external PHP class named NuSOAP, which exposes a number of methods that can be used to instantiate a SOAP client and perform SOAP transactions over HTTP. You can download NuSOAP from http://dietrich.ganx4.com/.

You also have the option of using the PEAR (the PHP Extension and Application Repository) SOAP class instead of the NuSOAP class, which exposes much of the same functionality. You can get this class from the PHP CVS repository, at http://cvs.php.net/.

In case you're wondering, I've used both the NuSOAP class and the PEAR class to build the examples in this tutorial.

Finally, you need a passport to the Google kingdom. Drop by http://www.google.com/apis/, register with Google (all you need is an email address), and pick up your free Google license key. This license key will be used in all your interaction with the Google database, so keep it carefully − you're going to need it very soon.

While you're on the Google site, you might also want to download the Google Web API developer kit, which contains numerous examples of how the Web APIs can be used on the Java and .NET platforms. It doesn't have anything on PHP yet − but hey, that's why you're reading this article, isn't it?

All set up? Let's rock and roll.

Plugging In

Google has made three methods available in their Web API. Here's what they look like:

doGoogleSearch() − search for a specified term in the Google database;

doGetCachedPage() − retrieve a page from the Google cache;

doSpellingSuggestion() − retrieve a spelling suggestion from Google.

This might not seem like much, but you'll soon see that it's more than enough to build some fairly powerful applications. Take a look at the following example, which provides a gentle introduction to running a search query on Google with PHP:

```
<?php
// include the class
include("nusoap.php");

// create a instance of the SOAP client object

// remember that this script is the client,
// accessing the web service provided by Google

$soapclient = new
soapclient("http://api.google.com/search/beta2");

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your-google-license-key-xxxxxxxx', // Google license
key
'q' => 'melonfire', // search term
'start' => 0, // start from result
n
'maxResults' => 10, // show a total of n
results
'filter' => false, // remove similar
results
'restrict' => '', // restrict by topic
'safeSearch' => false, // remove adult
links
'lr' => '', // restrict by
language
'ie' => '', // input encoding
'oe' => '' // output encoding
);

// invoke the method on the server
$result = $soapclient->call("doGoogleSearch", $params,
"urn:GoogleSearch", "urn:GoogleSearch");

// print the results of the search
print_r($result);
?>
```

The first order of business is to include the SOAP class which contains all the methods needed to access SOAP services. I've used NuSOAP here, but I'll show you the same thing using the PEAR class a little further down.

```
<?php
// include the class
include("nusoap.php");
?>
```

Now, in this SOAP universe, Google provides the SOAP server, and this PHP script works as the client. So, the next step is to instantiate this client, using the class constructs provided by NuSOAP.

```
<?php
// create a instance of the SOAP client object
$soapclient = new
soapclient("http://api.google.com/search/beta2");
?>
```

The class constructor accepts a single parameter, which is the URL of the SOAP service to be accessed (this is sometimes referred to by geeks as the "endpoint").

All that remains is to send a request to the SOAP server − something easily accomplished by the class' call() method.

```
<?php
// invoke the method on the server
$result = $soapclient->call("doGoogleSearch", $params,
"urn:GoogleSearch", "urn:GoogleSearch"); ?>
```

The call() method accepts four arguments − the name of the remote procedure to be invoked, an array containing arguments for this remote procedure, a method namespace and a SOAPAction value. The remote procedure name here is doGoogleSearch(), and the list of arguments to be passed to it are stored in the $params array, which looks like this:

```
<?php
// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your-google-license-key-xxxxxxxx', // Google license
key
'q' => 'melonfire', // search term
'start' => 0, // start from result
n
'maxResults' => 10, // show a total of n
results
'filter' => false, // remove similar
results
'restrict' => '', // restrict by topic
'safeSearch' => false, // remove adult
links
'lr' => '', // restrict by
language
'ie' => '', // input encoding
'oe' => '' // output encoding
);
?>
```

Here's what those arguments mean:

"key" − your Google license key;

"q" − the query string;

"start" − the index number of the first result to display;

"maxResults" − the maximum number of matches to display (limited to a maximum of 10);

"filter" − a Boolean indicating whether or not to display matches which are similar to each other;

"restrict" − limit the search to a specific section of the Google database;

"safeSearch" − filter out adult content from the result set;

"lr" − limit the search to a specific language;

"ie" − the character encoding of the query string;

"oe" − the character encoding of the result set.

If you take a look at the internals of the SOAP class, you'll see that the call() method uses the arguments passed to it to generate a SOAP request, which looks something like this:

```
POST /search/beta2 HTTP/1.0
User-Agent: NuSOAP v0.6
Host: api.google.com
Content-Type: text/xml
Content-Length: 946
SOAPAction: "urn:GoogleSearch"

<?xml version="1.0"?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:galactivism="urn:GoogleSearch">
<SOAP-ENV:Body>
<galactivism:doGoogleSearch>
<key
xsi:type="xsd:string">your-google-license-key-xxxxxxxx</key>
<q xsi:type="xsd:string">melonfire</q>
<start xsi:type="xsd:int">0</start>
<maxResults xsi:type="xsd:int">10</maxResults>
<filter xsi:type="xsd:boolean">0</filter>
<restrict xsi:type="xsd:string"></restrict>
<safeSearch xsi:type="xsd:boolean">0</safeSearch>
<lr xsi:type="xsd:string"></lr>
<ie xsi:type="xsd:string"></ie>
<oe xsi:type="xsd:string"></oe>
</galactivism:doGoogleSearch>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This request packet is transmitted to the SOAP server using the POST method, and a server response packet is transmitted back to the client. Here's what one such packet might look like:

```
HTTP/1.0 200 OK
Date: Fri, 21 Jun 2002 05:51:46 GMT
Content-Length: 10920
Content-Type: text/xml; charset=utf-8
Server: e h c a p a
Via: 1.1 HathCache (NetCache NetApp/5.0.1R2D3)
```

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="ns1:GoogleSearchResult">
<documentFiltering
xsi:type="xsd:boolean">false</documentFiltering>
<estimatedTotalResultsCount
xsi:type="xsd:int">3880</estimatedTotalResultsCount>
<directoryCategories
xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]">
</directoryCategories>
<searchTime xsi:type="xsd:double">0.247468</searchTime>
<resultElements
xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[10]">
<item xsi:type="ns1:ResultElement">
<cachedSize xsi:type="xsd:string">6k</cachedSize>
<hostName xsi:type="xsd:string"></hostName>
<snippet xsi:type="xsd:string"> &lt;b&gt;...&lt;/b&gt; Check
out. what
we&apos;ve cooked up in our brand-new PHP section. Copyright
1998-2002&lt;br&gt; &lt;b&gt;Melonfire&lt;/b&gt;. All rights
reserved
Terms and Conditions | Feedback. </snippet> <directoryCategory
xsi:type="ns1:DirectoryCategory"> <specialEncoding
xsi:type="xsd:string"></specialEncoding>
<fullViewableName
xsi:type="xsd:string">Top/Computers/Internet/Web_Design_and_Development/
Desi
gners/M</fullViewableName>
</directoryCategory>
<relatedInformationPresent
xsi:type="xsd:boolean">true</relatedInformationPresent>
<directoryTitle
xsi:type="xsd:string">&lt;b&gt;Melonfire&lt;/b&gt;
</directoryTitle>
<summary xsi:type="xsd:string">A production house with
business
divisions focusing on content production and Web development.
</summary>
<URL xsi:type="xsd:string">http://www.melonfire.com/</URL>
<title
```

```
xsi:type="xsd:string">&lt;b&gt;melonfire&lt;/b&gt;!</title>
</item>
<item xsi:type="ns1:ResultElement">
<cachedSize xsi:type="xsd:string">11k</cachedSize>
<hostName xsi:type="xsd:string"></hostName>
<snippet xsi:type="xsd:string"> &lt;b&gt;...&lt;/b&gt;
Copyright
1998−2002 &lt;b&gt;Melonfire&lt;/b&gt;. All rights
reserved&lt;br&gt;
Terms and Conditions | Feedback. </snippet> <directoryCategory
xsi:type="ns1:DirectoryCategory"> <specialEncoding
xsi:type="xsd:string"></specialEncoding>
<fullViewableName xsi:type="xsd:string"></fullViewableName>
</directoryCategory>
<relatedInformationPresent
xsi:type="xsd:boolean">true</relatedInformationPresent>
<directoryTitle xsi:type="xsd:string"></directoryTitle>
<summary xsi:type="xsd:string"></summary>
<URL
xsi:type="xsd:string">http://www.melonfire.com/community/columns/trog/ar
chiv
es.php?category=PHP</URL>
<title xsi:type="xsd:string">The &lt;b&gt;Melonfire&lt;/b&gt;
Community
− Trog</title> </item>
−− snip −−
</resultElements>
<endIndex xsi:type="xsd:int">10</endIndex>
<searchTips xsi:type="xsd:string"></searchTips>
<searchComments xsi:type="xsd:string"></searchComments>
<startIndex xsi:type="xsd:int">1</startIndex>
<estimateIsExact
xsi:type="xsd:boolean">false</estimateIsExact>
<searchQuery xsi:type="xsd:string">melonfire</searchQuery>
</return>
</ns1:doGoogleSearchResponse>

</SOAP−ENV:Body>
</SOAP−ENV:Envelope>
```

This response packet is decoded by the client into a native PHP structure, which can be used within the script. At the moment, all I'm doing is printing it − and here's what the output looks like:

```
Array
(
[documentFiltering] => false
[estimatedTotalResultsCount] => 3880
```

```
[directoryCategories] =>

[searchTime] => 0.05159
[resultElements] => Array
(
[0] => Array
(
[cachedSize] => 6k
[hostName] =>
[snippet] => <b>...</b> Check out. what we've
cooked up in our brand−new PHP section. Copyright
1998−2002<br>
<b>Melonfire</b>. All rights reserved Terms and Conditions |
Feedback.
[directoryCategory] => Array
(
[specialEncoding] =>
[fullViewableName] =>
Top/Computers/Internet/Web_Design_and_Development/Designers/M
)

[relatedInformationPresent] => true
[directoryTitle] => <b>Melonfire</b>
[summary] => A production house with business
divisions focusing on content production and Web development.
[URL] => http://www.melonfire.com/
[title] => <b>melonfire</b>!
)

[1] => Array
(
[cachedSize] => 11k
[hostName] =>
[snippet] => <b>...</b> Copyright 1998−2002
<b>Melonfire</b>. All rights reserved<br> Terms and Conditions
|
Feedback.
[directoryCategory] => Array
(
[specialEncoding] =>
[fullViewableName] =>
)

[relatedInformationPresent] => true
[directoryTitle] =>
[summary] =>
[URL] =>
http://www.melonfire.com/community/columns/trog/archives.php?category=PH
P
[title] => The <b>Melonfire</b> Community − Trog
)

// remaining array elements snipped out //

)

[endIndex] => 10
[searchTips] =>
[searchComments] =>
[startIndex] => 1
[estimateIsExact] => false
[searchQuery] => melonfire
)
```

Obviously, this is not very useful − but we're just getting started. Flip the page, and I'll show you how to massage all this raw data into something resembling a search results page.

Chasing Liberty

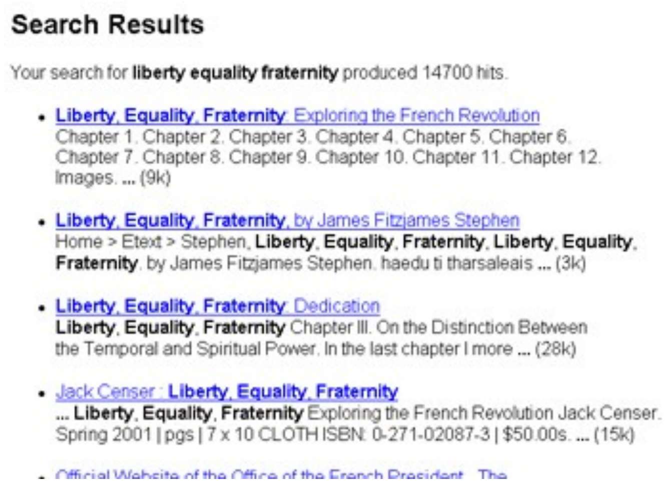
If you take a close look at the output of the previous example, you'll see that the call to doGoogleSearch() results in a PHP associative array containing a series of result elements, together with some statistics on the search itself. It's extremely simple to use this array to create an HTML page containing a properly−formatted list of matches to the query term. Here's an example:

```
<html>
<head><basefont face="Arial"></head>
<body>
<?php
// include the class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://api.google.com/search/beta2");

// uncomment the next line to see debug messages
// $soapclient−>debug_flag = 1;

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your−google−license−key−xxxxxxxx', // Google license
key
'q' => 'liberty equality fraternity', // search term
'start' => 0, // start from result
n
'maxResults' => 10, // show a total of n
results
'filter' => true, // remove similar
results
'restrict' => '', // restrict by topic
'safeSearch' => true, // remove adult
links
'lr' => 'lang_en|lang_fr', // restrict by
language
'ie' => '', // input encoding
'oe' => '' // output encoding
);

// invoke the method on the server
$result = $soapclient−>call("doGoogleSearch", $params,
"urn:GoogleSearch", "urn:GoogleSearch");

// print the results of the search
// if error, show error
if ($result['faultstring'])
{
?>
<h2>Error</h2>
<? echo $result['faultstring'];?>
<?
}
else
{
// else show list of matches with links
?>
<h2>Search Results</h2>
Your search for <b><?=$result['searchQuery']?></b> produced
<?=$result['estimatedTotalResultsCount']?> hits.
<br>
<ul>
<?
if (is_array($result['resultElements']))
{
foreach ($result['resultElements'] as $r)
{
echo "<li><a href=" . $r['URL'] . ">" .
$r['title'] . "</a>";
echo "<br>";
echo $r['snippet'] . "(" . $r['cachedSize'] .
")";
echo "<p>";
}
}
?>
</ul>
<?
}
?>
</body>
</html>
```

Most of this is identical to what you saw in the previous example, except that, this time, instead of just dumping the result array to the screen, I've used a "foreach" loop to iterate through it and display the matches as items in a bulleted list. Note how the various keys of the SOAP response array can be used to build the list of matching Web pages, with descriptions and URLs.

In the event that the procedure generates an error on the server, the response array will contain a SOAP fault. It's generally considered good programming practice to check for this and handle it appropriately − you'll see that I've done this in the script above.

Here's what the end result looks like:



Search Results

Your search for **liberty equality fraternity** produced 14700 hits.

- **Liberty, Equality, Fraternity**: Exploring the French Revolution
  Chapter 1. Chapter 2. Chapter 3. Chapter 4. Chapter 5. Chapter 6. Chapter 7. Chapter 8. Chapter 9. Chapter 10. Chapter 11. Chapter 12. Images. ... (9k)
- **Liberty, Equality, Fraternity**, by James Fitzjames Stephen
  Home > Etext > Stephen, **Liberty**, **Equality**, **Fraternity**. **Liberty**, **Equality**, **Fraternity**. by James Fitzjames Stephen. haedu ti tharsaleais ... (3k)
- **Liberty, Equality, Fraternity**: Dedication
  Liberty, **Equality**, **Fraternity** Chapter III. On the Distinction Between the Temporal and Spiritual Power. In the last chapter I more ... (28k)
- Jack Censer : **Liberty, Equality, Fraternity**
  ... **Liberty**, **Equality**, **Fraternity** Exploring the French Revolution Jack Censer. Spring 2001 | pgs | 7 x 10 CLOTH ISBN: 0-271-02087-3 | $50.00s. ... (15k)

The Sum Of All Parts

Thus far, the two examples you've seen have had the search term hardwired into them. Needless to say, this isn't very useful in the real world − it's far more sensible to have the search term generated dynamically via user input. The next example does just that, demonstrating how you can add a full−fledged, Google−backed search engine to your application.

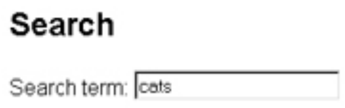
```
<html>
<head><basefont face="Arial"></head>
<body>
<?php

if (!$_POST['q'])
{
?>
<h2>Search</h2>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Search term: <input type="text" name="q">
</form>
<?
}
else
{
// include the class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://api.google.com/search/beta2");

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your-google-license-key-xxxxxxxx', // Google
license key
'q' => $_POST['q'], // search
term
'start' => 0, // start
from result n
'maxResults' => 10, // show a
total of n
results
'filter' => true, // remove
similar results
'restrict' => '', // restrict
by topic
'safeSearch' => true, // remove
adult links
'lr' => '', // restrict
by language
'ie' => '', // input
encoding
'oe' => '' // output
encoding
);

// invoke the method on the server
$result = $soapclient->call("doGoogleSearch", $params,
"urn:GoogleSearch", "urn:GoogleSearch");

// print the results of the search
if ($result['faultstring'])
{
?>
<h2>Error</h2>
<? echo $result['faultstring'];?>
<?
}
else
{
?>
<h2>Search Results</h2>
Your search for <b><?=$result['searchQuery']?></b>
produced <?=$result['estimatedTotalResultsCount']?> hits.
<br>
<ul>
<?
if (is_array($result['resultElements']))
{
foreach ($result['resultElements'] as $r)
{
echo "<li><a href=" . $r['URL'] . ">" .
$r['title'] . "</a>";
echo "<br>";
echo $r['snippet'] . "(" .
$r['cachedSize'] . ")";
echo "<p>";
}
}
?>
</ul>
<?
}
```

```
}
?>
</body>
</html>
```

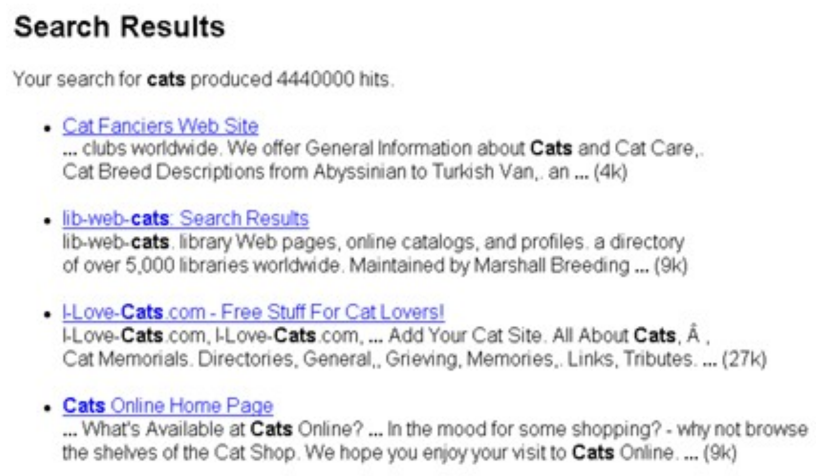
As you can see, this script is split into two sections, one for the search form and the other for the search results. An "if" statement, keyed on the presence or absence of the query string, is used to decide which section of the script to execute.

```
<?php

if (!$_POST['q'])
{
// display form
}
else
{
// execute query on Google
}
?>
```

Here's what the search form looks like:



Once the user enters a search term and submits the form, the second half of the script springs into action, initializes the SOAP client and performs the query on the Google SOAP server. The result is then formatted and displayed as in the previous example. Here's an example of what it might look like:



Search Results

Your search for **cats** produced 4440000 hits.

- Cat Fanciers Web Site
  ... clubs worldwide. We offer General Information about **Cats** and Cat Care,.
  Cat Breed Descriptions from Abyssinian to Turkish Van,. an ... (4k)
- lib-web-**cats**: Search Results
  lib-web-**cats**. library Web pages, online catalogs, and profiles. a directory
  of over 5,000 libraries worldwide. Maintained by Marshall Breeding ... (9k)
- I-Love-**Cats**.com - Free Stuff For Cat Lovers!
  I-Love-**Cats**.com, I-Love-**Cats**.com, ... Add Your Cat Site. All About **Cats**, Â ,
  Cat Memorials. Directories, General,, Grieving, Memories,. Links, Tributes. ... (27k)
- **Cats** Online Home Page
  ... What's Available at **Cats** Online? ... In the mood for some shopping? - why not browse
  the shelves of the Cat Shop. We hope you enjoy your visit to **Cats** Online. ... (9k)

And there you have it − your very own Google search!

Cache Cow

Google also allows you to retrieve cached Web pages from its database, via the exposed doGetCachedPage() procedure. Here's a quick example, which demonstrates retrieval of the URL "http://www.xmlphp.com/" from the Google cache.

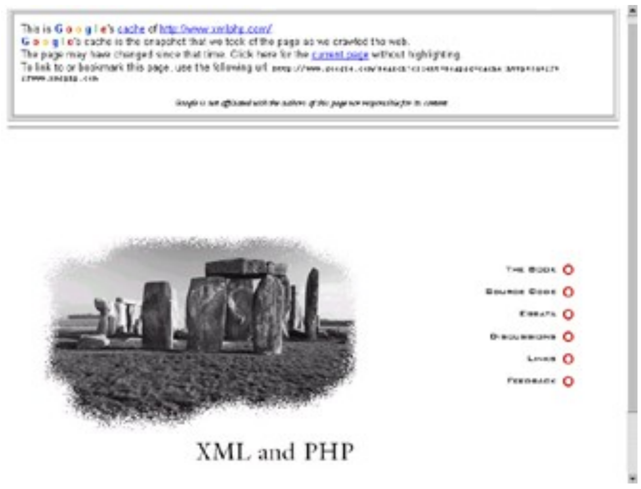
```
<?php
// include the class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://api.google.com/search/beta2");

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your-google-license-key-xxxxxxxx', // Google
license key
'url' => $_GET['u'], // URL to retrieve
);

// invoke the method on the server
$result = $soapclient->call("doGetCachedPage", $params,
"urn:GoogleSearch", "urn:GoogleSearch");

if ($result)
{
// fault?
if (is_array($result) && $result['faultstring'])
{
// display error
echo $result['faultstring'];
}
else
{
// decode return value using base64
// and display
echo base64_decode($result);
}
}
?>
```

The doGetCachedPage() procedure requires two parameters − your license key, and the URL to be retrieved. This cached content of this URL is retrieved as a base64−encoded data block, which needs to be decoded before it can be displayed. Luckily, PHP comes with a built−in base64_decode() function, which does just that.

Now, if you pass the script above the URL "http://www.xmlphp.com", like this,

```
http://localhost/retrieve.php?u=http://www.xmlphp.com
```

here's what you'll see:



This is Google's cache of http://www.xmlphp.com/
Google's cache is the snapshot that we took of the page as we crawled the web.
The page may have changed since that time. Click here for the current page without highlighting.
To link to or bookmark this page, use the following url: [illegible]

Google is not affiliated with the authors of this page nor responsible for its content.

You can link the doGoogleSearch() and doGetCachedPage() procedures to allow Web pages matching your search to be retrieved from the cache, in much the saw way as Google.com does. All that's needed is a slight modification to the search engine script created a couple pages back:

```
<?

// form submitted

// init soap client and call doGoogleSearch

// iterate through result

foreach ($result['resultElements'] as $r)
{
echo "<li><a href=" . $r['URL'] . ">" . $r['title'] .
"</a>";
echo "<br>";
echo $r['snippet'] . "(" . $r['cachedSize'] . ")";
echo "<br>";
echo "<a href=retrieve.php?u=" . $r['URL'] . ">Get
cached page</a>";
echo "<p>";
}

// snip

?>
```

As you can see, I've added a link below each displayed match to the script "retrieve.php", which is passed the URL to be retrieved from the cache. This URL is sent to the doGetCachedPage() procedure as a procedure argument, and the result is decoded and displayed.

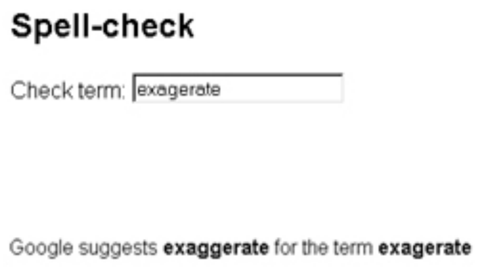
Spelling Bee

Finally, Google also allows you to vet spellings, via its doSpellingSuggestion() procedure. Here's a quick example:

```
<html>
<head><basefont face="Arial"></head>
<body>
<?php

if (!$_POST['p'])
{
?>
<h2>Spell-check</h2>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Check term: <input type="text" name="p">
</form>
<?
}
else
{

// include the class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://api.google.com/search/beta2");

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your-google-license-key-xxxxxxxx',
// Google
license key
'phrase' => $_POST['p']
// term to check
);

// invoke the method on the server
$result = $soapclient->call("doSpellingSuggestion", $params,
"urn:GoogleSearch", "urn:GoogleSearch");

// print the results of the search
if ($result)
{
if (is_array($result) && $result['faultstring'])
{
?>
<h2>Error</h2>
<? echo $result['faultstring'];?>
<?
}
else
{
echo "Google suggests <b>" . $result . "</b> for
the term <b>" . $_POST['p'] . "</b>";
}
}
else
{
echo "No suggestion";
}
}
?>
</body>
</html>
```

In this case, the doSpellingSuggestion() procedure expects two arguments − your Google license key, and the term to be checked against Google's internal dictionary. You can use this to build a rudimentary Web−based spell−checker (among other things). This spell−checker is actively used on the Google.com Web site, to identify and warn you about potential spelling errors in your query.

Here's what the output looks like:


Spell-check

Check term: exagerate

Google suggests **exaggerate** for the term **exagerate**

Alternatives

It's also possible to write equivalent code using the PEAR SOAP client, rather than the NuSOAP client. The procedure remains largely identical, though there is obviously a difference in the API used. As an illustration, consider the following code segment, which rewrites how the primitive search engine demonstrated a few pages back using the PEAR class:

```
<html>
<head><basefont face="Arial"></head>
<body>
<?php

if (!$_POST['q'])
{
?>
<h2>Search</h2>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Search term: <input type="text" name="q">
</form>
<?
}
else
{
// include the class
include("SOAP/Client.php");

// create a instance of the SOAP client object
$soapclient = new
SOAP_Client('http://api.google.com/search/beta2');

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'key' => 'your-google-license-key-xxxxxxxx', // Google
license key
'q' => $_POST['q'], // search
term
'start' => 0, // start
from result n
'maxResults' => 10, // show a
total of n
results
'filter' => true, // remove
similar results
'restrict' => '', // restrict
by topic
'safeSearch' => true, // remove
adult links
'lr' => '', // restrict
by language
'ie' => '', // input
encoding
'oe' => '' // output
encoding
);

// invoke the method on the server
$result = $soapclient->call("doGoogleSearch", $params,
"urn:GoogleSearch");

// print the results of the search
if (PEAR::isError($result))
{
?>
<h2>Error</h2>
<? echo $result['faultstring'];?>
<?
}
else
{
?>
<h2>Search Results</h2>
<? echo $result['estimatedTotalResultsCount'] . " hits
found in " . $result['searchTime'] . " ms"; ?>
<ul>
<?
if (is_array($result['resultElements']))
{
foreach ($result['resultElements'] as $r)
{
echo "<li><a href=" . $r['URL'] . ">" .
$r['title'] . "</a>";
echo "<br>";
echo $r['snippet'] . "(" . $r['cachedSize'] .
")";
echo "<p>";
}
}
?>
</ul>
<?
}
}
?>
</body>
</html>
```

This is very similar to what you've just seen − not surprising, considering that the PEAR SOAP client is based largely on code taken from Dietrich Ayala's NuSOAP classes. And, in case you want even more convenience, take a look at the SOAP_Google class, also part of the PEAR CVS repository, which provides a set of wrapper functions designed specifically for PHP developers looking to use the Google Web APIs.

Here's a quick example of what a script using the SOAP_Google class might look like:

```
<html>
<head><basefont face="Arial"></head>
<body>
<?php

if (!$_POST['q'])
{
?>
<h2>Search</h2>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Search term: <input type="text" name="q">
</form>
<?
}
else
{
// include the class
include("SOAP_Google.php");

// create a instance of the SOAP_Google client object
// pass the license key to the constructor
$google = new SOAP_Google('your−google−license−key−xxxxxxxx');

// invoke the search() method
$result = $google−>search(array('query' => $_POST['q']));

// print the results of the search
if ($result)
{
?>
<h2>Search Results</h2>
<? echo $result['estimatedTotalResultsCount'] . " hits
found in " . $result['searchTime'] . " ms"; ?>
<ul>
<?
if (is_array($result['resultElements']))
{
foreach ($result['resultElements'] as $r)
{
```

```
echo "<li><a href=" . $r['URL'] . ">" .
$r['title'] . "</a>";
echo "<br>";
echo $r['snippet'] . "(" . $r['cachedSize'] .
")";
echo "<p>";
}
}
?>
</ul>
<?
}
else
{
echo "<h2>Error</h2>";
}
}
?>
</body>
</html>
```

Closing Time

The methods exposed by the Google Web APIs make it possible to use a number of very cool features − Web search, cached document retrieval, phrase correction − in a simple, transparent and extensible manner. As the examples above demonstrates, PHP, with its powerful SOAP classes, is a natural fit for these APIs, and it opens up some very interesting new possibilities for PHP developers.

In case you'd like to learn more about the topics discussed in this article, take a look at the following links:

The W3C's Web Services section, at http://www.w3.org/2002/ws/

SOAP/XMLP, at http://www.w3.org/2000/xp/Group/

A Gentle Introduction To SOAP, at
http://radio.weblogs.com/0101679/stories/2002/03/16/aGentleIntroductionToSoap.html

Other SOAP tutorials, at http://www.w3schools.com/soap/default.asp and http://www.soapuser.com/

The NuSOAP classes, at http://dietrich.ganx4.com/nusoap/

The PEAR SOAP classes, at http://cvs.php.net/cvs.php/pear/SOAP

The PEAR SOAP_Google class, at http://www.sebastian−bergmann.de/?page=google

Finally, it's important to remember that merely understanding the Google Web APIs is not sufficient. You should treat that understanding as the stepping stone to bigger, brighter things. All that limits you is your imagination − so get out there, and start practicing!

Note: All examples in this article have been tested on Linux/i586 with Apache 1.3.24 and PHP 4.2.1. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!