

Use PHP to build a search engine optimization app, Part 1: Getting started

Track your SEO efforts with PHP, Derby, and some ingenuity

Skill Level: Intermediate

[Tyler Anderson \(tyleranderson5@yahoo.com\)](mailto:tyleranderson5@yahoo.com)
Freelance Writer
Stexar Corp.

14 Mar 2006

PHP, a dynamic Web-based programming language, takes a variety of input formats and uses a built-in SOAP client to obtain information from the Web. PHP, combined with applications using search engine optimization (SEO), is a powerful tool for obtaining information from major search engines, allowing this information to guide a webmaster's online marketing and SEO strategies. In Part 1 of this two-part "[Use PHP to build a search engine optimization app](#)" series, find out how to take advantage of these strategies by building the back end of an application to monitor and track your client's SEO efforts.

Section 1. Before you start

This tutorial is for PHP programmers and webmasters interested in learning about their search engine optimization (SEO) efforts. The example application accepts comma-separated values (CSV) files to obtain domain names and keyword combinations. The positions for each domain in a given search engine are obtained by taking the domain/keyword combinations to Google, Yahoo! and MSN. These results can then be downloaded in another CSV file containing the current positions of their domains.

About this series

This series creates a search engine optimization application that connects to search engines to obtain client positions for a given list of URLs and keywords. Matching results will be stored in an Apache Derby database for later processing.

Part 1 sets up and builds the back-end database and application in PHP. This includes code to retrieve positions from the three top search engines.

[Part 2](#) adds functionality to search Ask Jeeves and the Open Directory Project. You'll also extend the database and application to automatically bill clients, and provide two summary files in CSV format.

About this tutorial

This tutorial sets up the back end of an example SEO application using PHP. Through PHP and HTML, a CSV file will be read in to obtain a mapping of URLs to a set of keywords. The keywords will then be submitted to the three major search engines (Google, Yahoo! and MSN) via PHP's SOAP client or simple REST, and matching results will be stored with the current date in a Derby database.

Prerequisites

The following tools are needed to follow along with this tutorial:

Web server

Any operating system and any Web server can be used. Feel free to use [Apache V2.X](#) or the [IBM HTTP Server](#).

PHP

Due to the use of PHP data objects, [PHP V5.1](#) or higher is required for this tutorial. Be sure to configure PHP with the following option to include support for Derby and the SOAP extensions:

```
--with-pdo-odbc=ibm-db2,/home/db2inst1/sqlllib  
--enable-soap. See Resources for information about configuring Apache or the IBM HTTP Server with PHP.
```

Database

This tutorial uses [Apache Derby](#), which is open source and lightweight, the [IBM DB2 JDBC Universal Driver](#), and the [DB2 runtime client](#) from IBM. Make sure that you have set your classpath appropriately by following the given instructions on each page. You can follow either the [Linux®](#) or [Windows®](#) instructions for installing and downloading the DB2® runtime client. Cloudscape may also be used for this tutorial. The internals of Cloudscape are the same as Derby, however, the DB2 JDBC Universal Driver and other things are packaged into Cloudscape, and it is supported by IBM. Download

[Cloudscape V10.1](#), and the [DB2 runtime client](#) from IBM.

Java™ technology

Derby requires [Java technology](#) from Sun Microsystems or from [IBM](#).

Developer and application tokens

Each major search engine requires that you pass along some sort of ID with your SOAP or REST request for search results. You need to get one from each of them: [Google](#), [Yahoo!](#) and [MSN](#).

This tutorial assumes basic PHP knowledge of PHP syntax, including assignments, for loops and functions, and so forth.

Section 2. Overview

Getting good positions on search engines through SEO is a webmaster's ticket into the online community. Take a closer look at SEO, as well as the SEO application you'll build in this tutorial.

Search engine optimization

SEO involves best practices in Web design, as well as taking advantage of the built-in HTML features that allow you to insert keywords that are primarily the main focus of a Web site. See [Resources](#) to find a link to an excellent developerWorks tutorial outlining the basics of search engine optimization.

Basic SEO is pretty simple, and anyone can do it with a little know-how. However, it can be difficult to view and monitor your results and positions in the various search engines. Having a service that would allow webmasters to log and monitor their positions enables them to better focus their SEO efforts. The example application in this tutorial is such a service.

Let's take a look at the example application you'll be building.

The example SEO application

The example application in this tutorial will ask for a CSV file of domain names and keywords. It will then go through and query the three major search engines for a certain keyword or sequence of keywords to see if the domain name shows up in the top 30 results. Here are the contents of a sample CSV file so that you can get a feel

for the format used.

```
example.com, examples  
example.com, more examples  
ibm.com, ibm
```

Note how each line follows the same format:

```
<your-site.com>, <targeted keyword sequence>
```

Before diving in further, let's take a quick peek at what the example SEO application looks like and how the results are displayed (see figures 1 and 2).

Figure 1. Asking for a CSV file to find positions for

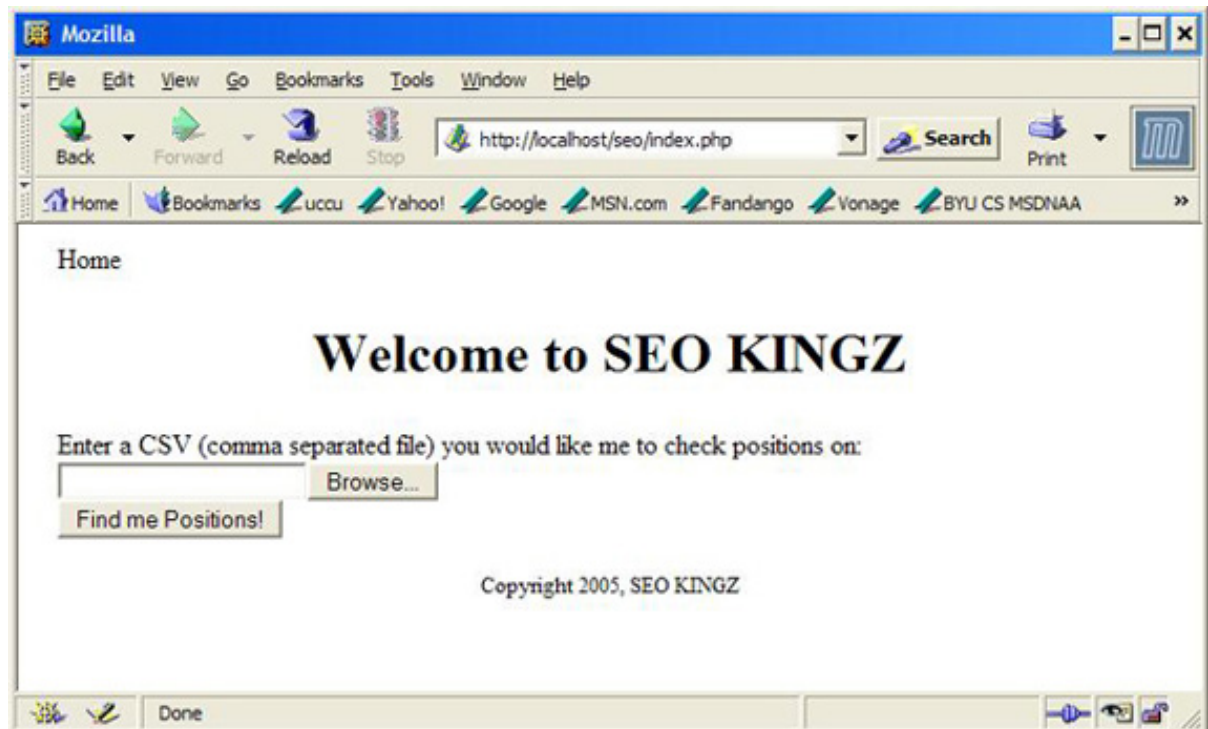


Figure 2. Returned results



Google, Yahoo!, and MSN

In an ideal world, the three major search engines would at least allow the same protocol to be used to obtain positions. However, this is not the case. Here are some of the important differences.

Google requires that you obtain a developer token, only allows SOAP with a service endpoint, and only allows a maximum of 10 returned positions and a 1,000 daily limit of requests.

MSN also requires you obtain a developer token and only allows SOAP while providing WSDL. However, MSN has no low maximum of 10 returned positions, and allows a much higher 10,000 daily limit.

Yahoo! is different from both Google and MSN. Yahoo! requires an ID of some sort. However, it is tied to the application and not the developer, and there is no daily limit associated with it. On top of that, it doesn't support SOAP at all, and only REST can be used to obtain positions. REST responses from Yahoo! contain the positions in

XML.

You need to keep these constraints in mind as you write the search code for these search engines.

Next, you'll set up everything you need to create and run the example application from the directory structure to the Derby database.

Section 3. Setting up

In this section, you are going to set up and configure Derby, create the table for storing the retrieved data, and review a list of the files and functions needed for the application code.

Setting up the Derby database

The first step is to start the Derby Network Server in a new console:

```
java
org.apache.derby.drda.NetworkServerControl
start
```

With the Network Server up and running, you can connect to it and create the database using Derby's `ij` tool. Start the `ij` tool in another console by typing:

```
java org.apache.derby.tools.ij
```

You should now be at the `ij` prompt. The next step is to create the SEO database, inserting your default user and password:

```
connect
'jdbc:derby:net://localhost:\
1527/SEO;create=true:user=seouser;
password=seopass;';
```

The SEO database within Derby is now ready to go. Next up is cataloging your new database using the IBM DB2 tool.

Cataloging with DB2

For your PHP Data Objects to know what database you're talking about, you need to catalog the new database in DB2. Start DB2 by typing db2.

You should now be at the db2 prompt. First, you need to catalog a node:

```
catalog tcpip node cns remote localhost server 1527
```

Then you need to catalog your database:

```
catalog db SEO at node cns authentication server
```

With the database cataloged, the PHP data objects should have no problem connecting to your new database. If you're having trouble, see [Resources](#) for helpful links.

You're now ready to initialize your database.

Introducing the SEO database

In Part 1, you need only one table for the application: *Positions*.

This table has three fields: domain, position, and search engine. The purpose of this table is to hold all the positions your PHP application matches during the position retrieval process. The domain field is simply the domain that matched, and the position field specifies the position it was found in. The search engine field will contain either google, yahoo, or msn, depending on which of the three search engines the domain was found on.

Initializing the database

You'll use the Positions table in conjunction with another table in Part 2 to bill your clients. The Positions table will be deleted and filled with fresh results with each run of the application. Therefore, creating and initializing the database will be done in the PHP application itself.

The directory structure

There is one main directory for this application that will contain a single subdirectory for include files. Create a directory titled seo on your Web server. Now create a single subdirectory to this directory titled includes.

The main directory in the Web server is the URL of your Web site (<http://localhost/seo> on my machine). The single subdirectory will contain all the functions of your application, for good code management and maintainability, as well as any required library files by your application. Next, let's look at the initial files and functions in the application.

Initial files and functions

It's important to get a big picture and overview of the files and functions you'll need in your code. Here are the files you need:

- index.php
- header.php
- footer.php
- includes/functions.php
- includes/xml.php

Get this open source library file from [Download](#) or by following the XML parsing link in [Resources](#). It will be used to parse the XML returned from Yahoo! in a PHP-readable data structure.

Let's fill in functions.php with the define and empty functions you'll create later (see Listing 1).

Listing 1. Defining the functions.php file

```
<?php
define(GOOGLE_KEY, 'Insert your Google token here');
define(MSN_KEY, 'Insert your MSN token here');
define(YAHOO_KEY, 'Insert your Yahoo app token here');
define(COMPANY_NAME, 'SEO KINGZ');
include('xml.php');

function db_connect($dbname='SEO',
                   $username='seouser',
                   $password='seopass'){
    $pdo = new PDO("odbc:$dbname", $username, $password);
    return $pdo;
}

function processFileCSV($filename, $tmpname){
}

function domainMatch($domain, $query, $pdo){
}

/***** GOOGLE *****/
function domainMatchGoogle($domain, $query, $pdo){
}
```

```

/***** MSN *****/
function domainMatchMSN($domain, $query, $pdo){
}

/***** YAHOO *****/
function domainMatchYahoo($domain, $query, $pdo){
}

function queryGoogle($query, $ct){
}

function queryMSN($query){
}

function queryYahoo($query){
}

function getDomain($url){
}

function displayTopBar(){
}

?>

```

The `db_connect ( )` function, in bold, connects to the Derby database you've already set up.

The above query functions will send the SOAP/REST requests to the search engine and return the results. The `domainMatch` functions call the query functions and match up the domains in the search results with the current domain being searched for.

You'll use quite a few constants throughout the code. `GOOGLE_KEY`, `MSN_KEY`, and `YAHOO_KEY` define the tokens or keys each major search engine requires. Be sure to fill these in with the token you receive from each search engine.

`COMPANY_NAME` is a simple constant where you can personalize the generic application with your company name.

Section 4. The application's architecture

Now you'll define a simple architecture where you or your clients can go to view Web-site status in each of the three major search engines.

Index page

The index page will take in requests to process CSV files and return matches found

in each search engine. Define the index.php file, as shown in Listing 2.

Listing 2. Defining the index.php file

```
<?php
include("includes/functions.php");
require('header.php');

print "Welcome to our SEO position checker";
require('footer.php');
?>
```

Next, you'll build a simple header and footer file.

Header

The header file helps make the layout look better. Define the header.php file, as shown in Listing 3.

Listing 3. Defining the header file

```
<?php

print('
<html>
<title>' . $title . '</title>
<body>
<table width="650px" align="center" valign="top">
<tr><td colspan="2">
');

displayTopBar();

print('
</td></tr>
<tr><td colspan="2">
<center><h1>Welcome to '.COMPANY_NAME.'</h1></center></td></tr>');

print('<tr><td height="100%">
');
?>
```

You'll use an HTML table to format the page. Notice the call to `displayTopBar()`, which toggles a link to enter another CSV file or display "Home." Define this function in the functions.php file, as shown in Listing 4.

Listing 4. Defining the displayTopBar function

```
function displayTopBar(){
    if(!isset($_FILES['filename']['name'])){
        print("Home");
    }
}
```

```
    }  
    else{  
        print("<a href='index.php'>Process another CSV file!</a>");  
    }  
}
```

Later, when you use the `<input type="file" ...` HTML tag to upload CSV files, the name of the actual file will be stored in the `FILES` array, as shown in Listing 4. If the filename exists, the file has already been stored, and search engine position results are displayed to the user. If this is the case, a link to process another CSV file will be displayed.

Footer

The purpose of the footer file is to close off the `<table>`, `<tr>`, and `<td>` tags started in the header file, as well as a brief copyright. Define the footer.php file, as shown in Listing 5.

Listing 5. Defining the footer file

```
    </td></tr>  
<tr><td align="center" colspan="2">  
    <font size="2px"><br>  
    <center>Copyright 2005, <?php print(COMPANY_NAME) ?></center>  
    </font>  
</td></tr></table>  
</body></html>
```

The PHP code, in bold, is inserted to display the constant `COMPANY_NAME`. This way, you can modify the constant in the `functions.php` file, and changes to the display will happen automatically as your PHP scripts refer to this constant. See Figure 3 for example browser output.

Figure 3. The current application



This completes the layout of your application. Next, begin processing CSV files.

Section 5. Processing CSV files

In this section, you'll create a form to upload a CSV file and a function to parse it, making your code ready to go to the search engine and query them for positions. You'll also add database and a PHP Data Objects (PDO) connection to the function.

The HTML form for uploading CSV files

A form is necessary to allow anyone to upload CSV files for processing at your Web site. This can be really handy for clients who pay you for your SEO services, as they will be eager to see how well you are at making their Web sites rise through the ranked positions in each of the search engines.

Thus, you need to display a form, which is stored in the index.php file, as shown in Listing 6.

Listing 6. The form for uploading CSV files

```
...
require(header.php);
print "<form enctype='multipart/form-\"
```

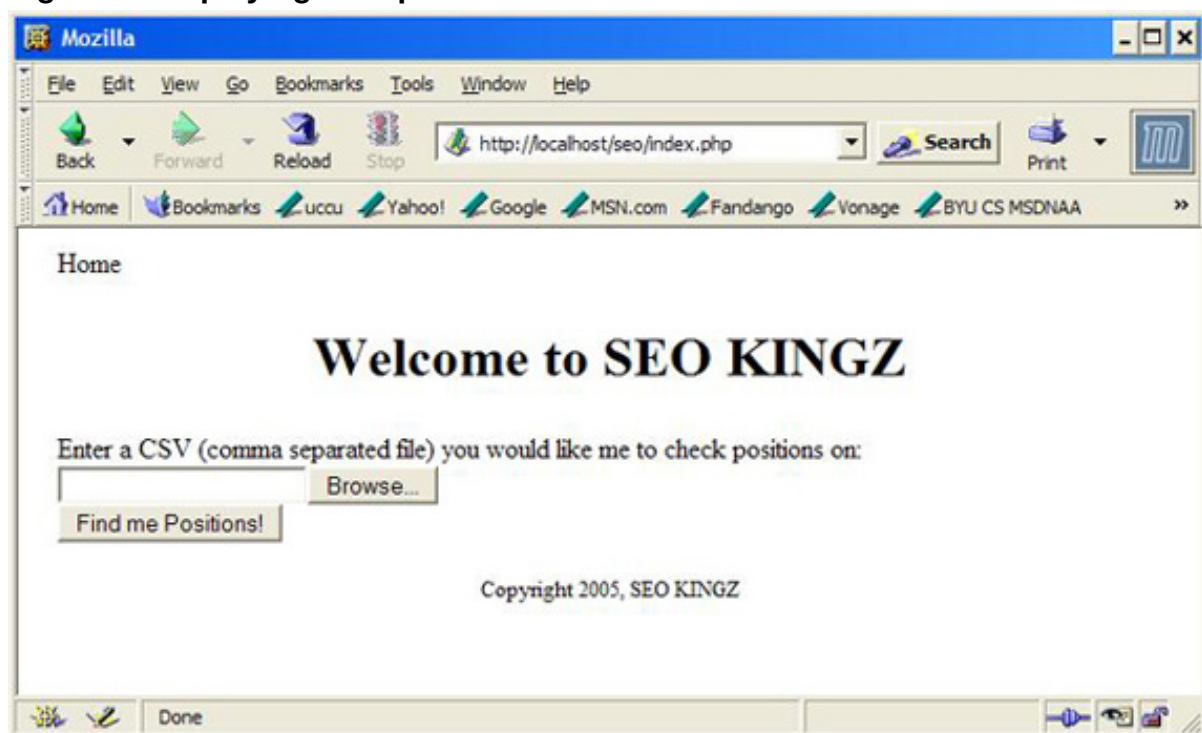
```

data' action='index.php'".
    " method='POST'>".
    "<input type='hidden' name='max\
    Size' value='102400' />".
    "Enter a CSV (comma separated
    file) you".
    " would like me to check positions
    on: ".
    "<input type='file' name=\
    'filename' /><br />".
    "<input type='submit' value='Find
    \
    me Positions!' /></form>";
require/footer.php);
...

```

Notice that the form specifies transfer via POST. It's best practice to transfer files via POST rather than GET. You wouldn't want people uploading files that are too large, and so you can communicate to the Web server a maximum file size in a hidden input type called `maxSize`. Finally, the file input type is used to upload the CSV file. See Figure 4 for example browser output of the form.

Figure 4. Displaying the upload CSV file form



Processing the upload CSV file request

Your script needs to know if it has arrived as planned. Do so by continuing to define the `index.php` file, as shown in Listing 7.

Listing 7. Processing upload requests

```

...
require('header.php');
if($_FILES['filename']['name'] != ''){
    processFileCSV($_FILES['filename']['name'],
                  $_FILES['filename']['tmp_name']);
}
else{
    print "<form enctype='multipart/form-data' action='index.php'".
    ...
        "<input type='submit' value='Find me Positions!' /></form>";
}
require('footer.php');
...

```

When a file is uploaded, the details are found on the server through the FILES array in PHP. Listing 7 checks that a file was transferred using the form from [Listing 6](#). If the file was uploaded as planned, the `processFileCSV()` function is called, which will open and parse the file.

Processing a CSV file into a PHP data structure

Great! The CSV file is loaded, and all you have to do is open it and transfer its contents into a PHP data structure. Begin defining the `processFileCSV()` function in the `functions.php` file, as shown in Listing 8.

Listing 8. Processing the contents of a CSV file

```

function processFileCSV($filename, $tmpname){
...
    $fp = fopen($tmpname, 'rb');
    $row = 0;
    while (($data[$row] = fgetcsv($fp, 1024, ",")) != false){
        $row++;
    }
    fclose($fp);
...
}

```

Open the file in read-buffer mode and store each row of the CSV file into the `$data` variable. Note the call to `fgetcsv()`. This is a special file-reader function for CSV files that ignores white space between text and the delimiter, as shown in the third parameter (", "). Each variable on a row can then be separately retrieved, as you'll see.

Looping through the loaded CSV data

With the data from the loaded CSV file in your PHP structure, you're ready to hand off the data to the domain matching code, which you'll write in [Storing matched domains in Derby](#). Continue defining the `processFileCSV()` function, as shown in

Listing 9.

Listing 9. Looping through the CSV data

```

    function processFileCSV($filename, $tmpname){
...
    fclose($fp);
    for($i = 0; $i < $row; $i++){
        print "Domain: ".$data[$i][0]."<br>Keywords: ".
            $data[$i][1]."<br>";
        $totalFound = domainMatch($data[$i][0],
                                urlencode($data[$i][1]), $pdo);
        print "Total found: ".$totalFound;
        print "<br>-----<br>";
    }
...
}

```

The new code is noted in bold. It loops through all the rows, as specified by the count contained in the \$row variable from [Listing 8](#). Since the domain is first and the keywords second in the original example CSV file, the domain is obtained by accessing \$data[\$i][0] and the keywords by accessing \$data[\$i][1]. A total is then retrieved from the domainMatch() function you'll write later and displayed to the user. Next, you'll write code to handle the SOAP and database exceptions that might be thrown.

Handling exceptions

There are several exceptions that might occur when running the code. For example, Google may take too long in servicing your request or throw SOAP faults, etc. The PDO or database may also throw exceptions. Continue defining the processFileCSV() function by adding exception handling code, as shown in Listing 10.

Listing 10. Handling exceptions

```

    function processFileCSV($filename, $tmpname){
    try{
        $fp = fopen($tmpname, 'rb');
...
        for($i = 0; $i < $row; $i++){
...
            print "<br>-----<br>";
        }
...
    } catch(Exception $e){
...
        print("<br>Exception caught: ".$e->getMessage());
    }
...
}

```

If an exception is thrown in any of your code within the `try` statement, code execution will go to the `catch` statement with the exception data being stored in the `$e` variable. Next, you'll add the database and PDO connection code to this function.

The PDO database transaction

With the function all ready to go, it's time to instantiate a PDO object that allows you to connect to the database and later add matches to the Derby code. Finish the `processFileCSV()` function, as shown in Listing 11.

Listing 11. Handling PDO transactions

```
function processFileCSV($filename, $tmpname){
    $pdo = db_connect();
    try{
        $pdo->beginTransaction();
        $pdo->exec("drop table positions");
        $pdo->exec("create table positions (domain varchar(1024),".
            " position integer, searchengine varchar(255))");
        $fp = fopen($tmpname, 'rb');
    ...
        for($i = 0; $i < $row; $i++){
    ...
            $totalFound = domainMatch($data[$i][0],
                                   urlencode($data[$i][1]), $pdo);
    ...
        }
        $pdo->commit();
    } catch(Exception $e){
        $pdo->rollBack();
        print("<br>Exception caught: ".$e->getMessage());
    }
    $pdo = null;
}
```

First, the database connection is created. At the beginning of the `try` statement, you begin a transaction with the call to `beginTransaction()`, meaning that you can roll back any changes if exceptions occur. Next, the `Positions` table is erased and recreated for this CSV file. If no exceptions occur and you reach the end of the `try` statement, a call to the `commit()` function is made, saving changes to the database made by the `domainMatch()` functions. If an exception occurs, the `rollBack()` function is called within the `catch` statement, nullifying any changes made to the database. Finally, the connection is closed by setting the `$pdo` object to `null`.

Next is the code for querying the search engines.

Section 6. Querying the search engines

In this section, you'll write the functions that use SOAP and REST to obtain results from each of the three major search engines.

Google

As mentioned, Google allows only SOAP requests for obtaining positions. The hard part is that they only allow a maximum of 10 positions to be returned, so to hit the top 30, you need to make three requests to Google. Define the `queryGoogle()` function, as shown in Listing 12.

Listing 12. Querying Google via SOAP

```
function queryGoogle($query, $ct){
    $start = ($ct-1)*10;
    $params = array(
        'key' => GOOGLE_KEY,
        'q' => $query,
        'start' => $start,
        'maxResults' => 10,
        'filter' => false,
        'restrict' => '',
        'safeSearch' => false,
        'lr' => '',
        'ie' => '',
        'oe' => ''
    );

    $client = new
        soapclient(null,
            array('location' =>
                "http://api.google.com/search/beta2",
                'uri' =>
                "urn:GoogleSearch"));

    return
        $client->__soapCall('doGoogleSearch',
            $params);
}
```

First, the `$start` variable is calculated (due to the fact that we have to query Google more than once to get 30 positions). The `$params` list has each of the variables in the SOAP request, all of which are required, however, some are ignored. Check out [Resources](#) on the Google API for more information. The items of interest involve the `key` and `maxResults` parameters. The developer token you obtained is the value of `key`, the keywords stored in the CSV file is the value of `q`, and `maxResults` is 10. Next, the `$client` is initialized, and since Google doesn't give a WSDL file, the service endpoint (`location`) and namespace (`uri`) must instead be passed as parameters. Finally, the call is made on the `doGoogleSearch` function. This returns a SOAP object that the corresponding `domainMatch` function can later extract the appropriate URL information from.

MSN

MSN allows SOAP and has no restrictions on how many positions you can get back, so you'll only need to make one request to MSN. Define the `queryMSN()` function, as shown in Listing 13.

Listing 13. Querying MSN via SOAP

```
function queryMSN($query){
    $params = array('Request' => array('Request' => array(
        'AppID' => MSN_KEY,
        'Query' => $query,
        'CultureInfo' => 'en-US',
        'SafeSearch' => 'Moderate', // default parameter
        'Flags' => '',
        'Location' => '',
        'Requests' => array (
            'SourceRequest' => array (
                'Source' =>
                    'Web',
                'Offset' => 0,
                'Count' => 30,
                'ResultFields'
                    => 'All'
            ))));
    $client = new
        soapclient("http://soap.search.msn.com/webservices.asmx?wsdl");
    return $client->__soapCall('Search', $params);
}
```

Similar to the Google request, with a little more complex hierarchy, the parameters are set up with your developer ID stored in the `AppID` variable, the `$query` being stored as `Query`, and the number of retrieved results stored in the `Count` variable as 30. MSN does provide WSDL, so creating the `$client` just requires the WSDL file to be passed in, and the `soapCall` is made to the `Search` function.

Yahoo!

Yahoo! is different in that it only supports REST and returns results in XML. However, like MSN, there is no restriction on the number of positions returned. Define the `queryYahoo()` function, as shown in Listing 14.

Listing 14. Querying Yahoo! via REST

```
function queryYahoo($query){
    $getURL = "http://api.search.yahoo.com/WebSearchService/V1/" .
        "webSearch?appid=".YAHOO_KEY."&query=".$query."&results=".30;
    $fp = fopen($getURL,"rb");
    $results = "";
}
```

```
while(!feof($fp)){
    $results .= fread($fp, 8192);
}
fclose($fp);
return XML_unserialize($results);
}
```

The URL for the REST request is first calculated. The parameters are very similar to MSN and Google. The URL is opened like a file and is read into the `$results` variable. Here's where you use the `xml.php` library obtained from [Download](#), which contains the `XML_unserialize()` function. This function takes the XML file returned from Yahoo! and turns it into a PHP data structure, as you'll see in the next section where you'll begin storing matches.

Section 7. Storing matched domains in Derby

You've laid the foundation for querying the search engines. Now you'll use that code in obtaining the positions from the results and store them in the Derby database.

Helper function

Before getting into the domain matching, you must first define a helper function. This function takes a long URL and extracts only the domain name from it. Define the `getDomain()` function, as shown in Listing 15.

Listing 15. Extracting the domain name from a URL

```
function getDomain($url){
    $str = explode("//", $url);
    $str2 = explode("/", $str[1]);

    $comp1 = $str2[0];
    if(substr($comp1, 0, 4) === "www.")
        $comp2 = substr($comp1, 4,
            strlen($comp1));
    else
        $comp2 = "www." . $comp1;

    return array('d1' => $comp1, 'd2' =>
        $comp2);
}
```

For example, if `http://example.com/index.php` is passed to this function, you'll get two results: `d1` and `d2`. `d1` will have the value of `example.com`, and `d2` will have the value of `www.example.com`.

Next, you'll define the `domainMatch()` function.

Top-level domainMatch function

Recall `domainMatch()` referenced in Listing 10? Here's where you get to define it. Basically, it takes a domain and keyword set from the CSV file and searches each of the search engines for results. Define the `domainMatch()` function, as shown in Listing 16.

Listing 16. Defining the domainMatch() function

```
function domainMatch($domain, $query, $pdo){
    $totalFound = 0;
    $totalFound += domainMatchYahoo($domain, $query, $pdo);
    $totalFound += domainMatchMSN($domain, $query, $pdo);
    $totalFound += domainMatchGoogle($domain, $query, $pdo);
    return $totalFound;
}
```

This functions calls each `domainMatch` function for Yahoo!, MSN, and Google, respectively, and returns the total number of positions found. Let's go over the functions for matching domains with each of the three major search engines, respectively.

Matching domains for Google

Everything is in place, so now you just need to get the results from Google via SOAP, parse out the domain, and store any matched positions in the SEO database. Define the `domainMatchGoogle()` function, as shown in Listing 17.

Listing 17. Querying and matching domains for Google

```
function domainMatchGoogle($domain, $query, $pdo){    $position = 1;
    $found = 0;

    for($i = 1; $i <= 3; $i++){
        usleep(1000);
        $results = queryGoogle($query, $i);
        for($j = 0; $results->resultElements[$j]->URL; $j++,
            $position++){
            $d = getDomain($results->resultElements[$j]->URL);
            if($d['d1'] === $domain || $d['d2'] === $domain){
                $pdo->exec("insert into positions values ('".$domain.
                    "' , '".$position."', 'google')");
                $found++;
            }
        }
    }
    return $found;
}
```

Google requires three requests, so you need a for loop that iterates three times, which will get you 30 results. Each result is then looped through in the second for loop. Then the domain is stripped out of the URL, and if a position matches the given domain, it is stored in the SEO database with google being the matching search engine. Last, the number of \$found matches is incremented.

Matching domains for MSN

MSN is very similar to Google, except you only need to make one request. Define the domainMatchMSN() function, as shown in Listing 18.

Listing 18. Querying and matching domains for MSN

```
function domainMatchMSN($domain, $query, $pdo){
    $results = queryMSN($query);

    $position = 1;
    $found = 0;

    for($i = 0;
    $results->Response->Responses->SourceResponse->
Results->Result[$i]->Url;
    $i++, $position++){
        $d = getDomain($results->Response->Responses->SourceResponse->
Results->Result[$i]->Url);
        if($d['d1'] === $domain || $d['d2'] === $domain){
            $pdo->exec("insert into positions values ('".$domain.
                "', ".$position.", 'msn')");
            $found++;
        }
    }
    return $found;
}
```

The for loop is the same as Google's second for loop, except the returned SOAP structure is different, so the URL must be retrieved from the right location, and msn is stored instead of google as the matching search engine.

Matching domains for Yahoo!

Yahoo! is also similar, except the returned data structure is a little different. Define the domainMatchYahoo() function, as shown in Listing 19.

Listing 19. Querying and matching domains for Yahoo!

```
function domainMatchYahoo($domain, $query, $pdo){
    $results = queryYahoo($query);

    $position = 1;
    $found = 0;
```

```
if($results['ResultSet'])
for($i = 0; $results['ResultSet']['Result'][$i]['Url']; $i++,
    $position++){
    $d = getDomain($results['ResultSet']['Result'][$i]['Url']);
    if($d['d1'] === $domain || $d['d2'] === $domain){
        $pdo->exec("insert into positions values ('".$domain.
            "', ".$position.", 'yahoo')");
        $found++;
    }
}
return $found;
}
```

The data structure used here requires that you first make sure that the result from Yahoo! and the PHP data structure returned from the library contain a `ResultSet` variable (see the code marked in bold in Listing 19). Beside the fact that the URL is being retrieved from the returned data structure in a different way, this Yahoo! code is the same as the MSN code, except that Yahoo! is stored as the matching search engine.

Next, you'll see the whole application in action.

Results

Uploading the example CSV file from Listing 1 will now give you the results shown from the browser output in Figure 5.

Figure 5. Results found in the CSV file



Typing `select * from positions;` at the `ij` prompt yields the output shown in Listing 20.

Listing 20. Viewing the stored positions in Derby

DOMAIN	POSITION	SEARCHENGINE
ibm.com	1	yahoo
ibm.com	2	yahoo
ibm.com	3	yahoo
ibm.com	6	yahoo
ibm.com	7	yahoo
ibm.com	15	yahoo
ibm.com	1	msn
ibm.com	2	msn
ibm.com	1	google
ibm.com	2	google
ibm.com	3	google
ibm.com	4	google
ibm.com	5	google
ibm.com	6	google
ibm.com	7	google
ibm.com	8	google
ibm.com	11	google

```
ibm.com | 12 | google
ibm.com | 13 | google
ibm.com | 14 | google
ibm.com | 15 | google
ibm.com | 16 | google
ibm.com | 17 | google
ibm.com | 18 | google
ibm.com | 19 | google
ibm.com | 21 | google
ibm.com | 22 | google
ibm.com | 23 | google
ibm.com | 24 | google
ibm.com | 25 | google
ibm.com | 26 | google
ibm.com | 27 | google
ibm.com | 28 | google
33 rows selected
```

You can now view the positions of your domains in the three major search engines!

Section 8. Summary

You're on your way to building your own position checking SEO application. Come back for Part 2 where you'll learn to process the results and match them within Derby with billing values, as well as send a CSV file to your clients so they can save a copy of their latest results.

Downloads

Description	Name	Size	Download method
Part 1 source code	os-php-seo1.source.zip	5.4KB	HTTP

[Information about download methods](#)

Resources

Learn

- Read "[Search engine optimization basics, Part 1](#)" to get a foundation in search engine optimization to organically optimize your Web site and create Web pages that are usable, accessible, and friendly to search engines.
- Learn how to get Apache V2 and PHP V4.x working together on Linux with [Apache V2 and PHP Installation](#).
- For a description, examples, and options for the `getcsv` command, see [fgetcsv](#).
- Find out how to install and configure PHP on Windows by reading "[Connecting PHP Applications to Apache Derby](#)."
- [PHP.net](#) has links to documentation, source, mailing lists, and news groups.
- Check out the Search Guild for [SEO forums](#) and articles on the subject.
- Learn the basics of PHP with the "[Learning PHP](#)" series of tutorials from developerWorks.
- [ALT-PHP-FAQ.org](#) provides an excellent resource for all PHP-related issues.
- Browse all of the [PHP content](#) on developerWorks.
- Expand your PHP skills by visiting IBM developerWorks [PHP project resources](#).
- Visit the developerWorks [Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Access a library that allows you to easily parse XML into a PHP data structure and to serialize PHP data structures into XML: [keithdevens.com](#).
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.

Discuss

- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

About the author

Tyler Anderson

Tyler Anderson formerly worked for DPMG.com, an SEO company, for whom he wrote proprietary SEO software. He graduated with a degree in computer science from Brigham Young University in 2004 and has just graduated with a Master of Science degree in Computer Engineering in December 2005, also from Brigham Young University. He is currently an engineer for Stexar Corp., based in Beaverton, Oregon. You can reach Tyler at tyleranderson5@yahoo.com.