

# Create a content management system with PHP and Derby

Use a CMS to submit your site content to Google's index automatically

Skill Level: Intermediate

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

11 Oct 2005

Learn how to build a simple content management system (CMS) that can be used to create, maintain, and submit a sitemap to Google automatically. Page content is managed through a database using PHP and a Web browser. When the system's content changes, the system creates and submits a sitemap to Google, speeding up the process of getting your new and updated content indexed.

## Section 1. Before you start

This tutorial is for those interested in automating the process of creating, editing, and submitting a Google sitemap using a CMS with PHP. This tutorial assumes familiarity with basic PHP concepts, including loops and if-then statements, form handling, accessing a database, and the Document Object Model (DOM). You can learn about these by reading the "[Learning PHP](#)" series. Many other articles and tutorials are available, as well (see [Resources](#)).

### About this tutorial

In this tutorial, you will build a CMS that will house the content of your Web site. It will also allow you to add and edit pages through a form, and save them in the

database by clicking a button. On the main CMS control page, there will be an option to view, edit, and delete each page in your Web site, add a new page, or submit them all to Google using a beta mechanism called Google Sitemaps (see [Resources](#)).

A Google sitemap is an easy and efficient way to maintain information about the content on your Web site. A sitemap contains information like how often content changes, the date of the last modification, and the priority you place on a page that indicates to Google how important it is that the new content get into Google's index quickly.

Normally, a Google sitemap is created and maintained manually in an XML document. The sitemap is also manually submitted to Google. The CMS you will build in this tutorial will automatically create and maintain the sitemap, and on changes to content, the CMS will automatically submit the sitemap to Google for swift indexing.

## Prerequisites

To follow along, install and test the following tools (see [Resources](#) for help and links to documentation):

### Web Server

It doesn't really matter which Web server you use, or whether you use Linux® or Windows®. [Download Apache V2.X](#).

### PHP

PHP V4 and V5 are in use at the time of this writing, but we recommend V5 for the DOM sections in this tutorial.

### Database

The content management system of this tutorial relies on a database. This tutorial uses Derby, which is open source. [Download Derby V10.1](#), the [IBM DB2® JDBC Universal Driver](#), and the [DB2 run-time client](#) from IBM. Make sure that you have set your CLASSPATH appropriately by following the instructions on each page. You can follow the [Linux](#) or [Windows](#) instructions for installing and downloading the DB2 run-time client.

### Cloudscape

This database could 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. Although Derby is used in this tutorial, you can download [Cloudscape V10.1](#) and the [DB2 run-time client](#) from IBM.

## Java™

Derby requires Java technology. In working on a Linux box running Red Hat Fedora Core, I found that gcj provided in the distribution was insufficient.

---

## Section 2. Setting up

In this section, you will set up the database and the directory structure, and create the PHP templates before you jump into creating the CMS later.

### Setting up a Derby database

To set up a Derby database, you'll need to start the Network Server:

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

Now that you have started the Network Server, you can begin communicating to it. Before you can communicate with the Network Server, open the Derby command-line processor by typing the following in a new console window:

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

You should now be at the `ij` prompt. Now you will communicate to the Network Server that you want to create a new database -- CMS -- with user `cmsuser` and password `cmspass`:

```
connect 'jdbc:derby:net://localhost:1527/CMS;
create=true:user=cmsuser;password=cmspass';
```

You have now created the Derby database. The next step is to catalog the database so that when you connect to Derby via PHP, PHP will be able to find the new database. Type the following in a new console window to start the DB2 command-line processor: `db2`.

You should now be at the `db2` prompt. Before you catalog the database, you need to catalog a `tcpip` node (`cns`) by typing the following:

---

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

The command-line processor should then return the following output:

```
DB20000I The CATALOG TCPIP NODE
command completed successfully.
DB21056W Directory changes may not
be effective until the directory cache is refreshed.
```

Now you can catalog your new CMS database:

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

The following output should be returned:

```
DB20000I The CATALOG DATABASE
command completed successfully.
DB21056W Directory changes may not
be effective until the directory cache is refreshed.
```

## Connecting to the database

Now that the database is cataloged, you should be able to connect to it in PHP. For a sanity check, and to make sure that everything is working, go ahead and connect to the database at the db2 prompt by typing `connect to CMS user cmsuser using cmsspass`.

If everything is working, the following should be returned as output to the above command:

### Listing 1. Output from the db CMS command

```
Database Connection Information
Database server      = Apache Derby CSS10011
SQL authorization ID = CMSUSER
Local database alias = CMS
```

You can now start setting up the CMS table via PHP.

## Setting up and testing a table via PHP in the Derby database

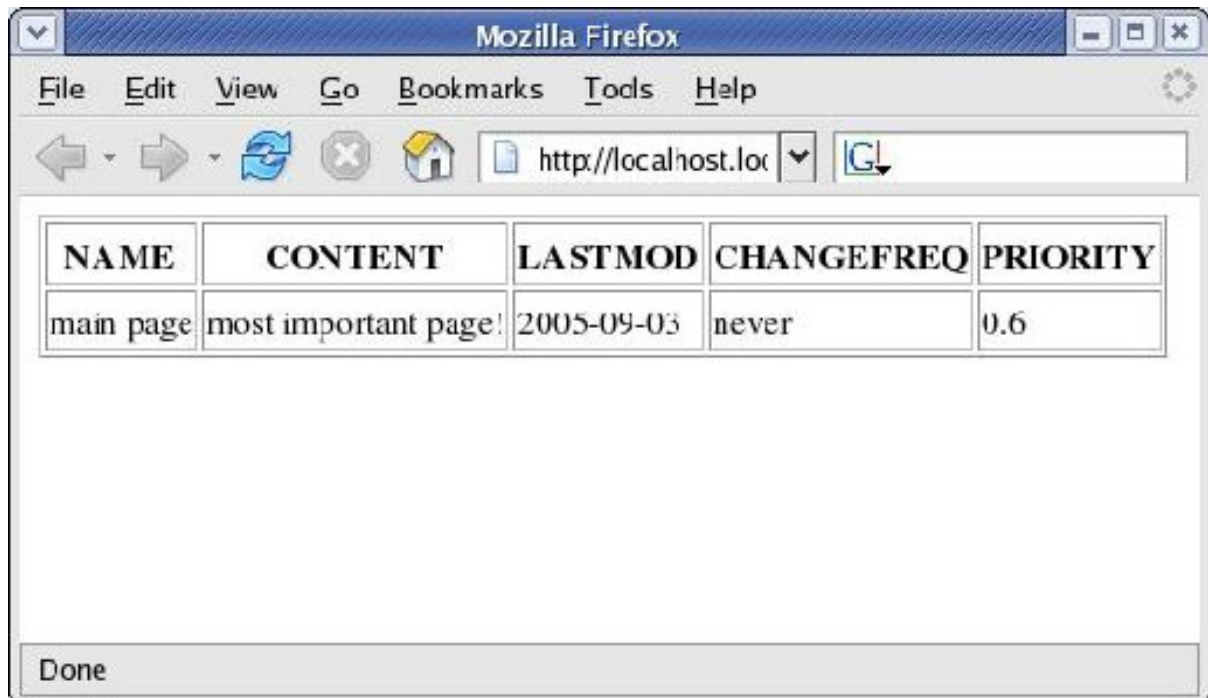
You have created the database. Next, you need to create the table. Create a file, `createTestTable.php`, and add the PHP code to it.

## Listing 2. Creating and testing a table in the CMS database

```
<?php
$username = "cmsuser";
$password = "cmspass";
$dbname = "CMS";
putenv("DB2INSTANCE=tfunk");
$dbconn = odbc_connect($dbname,$username,$password);
odbc_autocommit($dbconn, TRUE);
if($dbconn != 0){
    odbc_exec($dbconn,
        "CREATE TABLE CMS
        (name varchar(30) primary key,
        content varchar(3000),
        lastmod varchar(10),
        changefreq varchar(7),
        priority varchar(3));");
    $result = odbc_exec($dbconn,
        "insert into CMS values
        ('main page', 'most important page!',
        '2005-09-03', 'never', '0.6')");
    if ($result == 0){
        $sqlerror = odbc_errormsg($dbconn);
        die("Error inserting: $sqlerror");
    }
    $result = odbc_do($dbconn,
        "Select * from CMS;");
    odbc_result_all($result, "BORDER=1");
    $result = odbc_do($dbconn,
        "delete from CMS where name='main page';");
}
if(!odbc_commit($dbconn)){
    print "Error on commit\n";
}
odbc_close($dbconn);
?>
```

The code in Listing 2 will connect you to the database, passing in the database name, CMS, username cmsuser, and password cmspass to the database. It will then execute a SQL statement that will create the table you will be using throughout this tutorial to store the content of your Web site. Then it will insert a record into the CMS table, print out the results, delete the added record, and disconnect from the database. See Figure 1 for browser output.

### Figure 1. Browser output from createTestTable.php



Now you can access, delete from, update, and add to the CMS table using `cmsuser` with its password `cmsspass`.

## Setting up the directory structure

The next step is to set up the directory structure. You can place the root directory of your CMS in any directory. You will place most of the files in this directory, except for a couple of files (libraries) that will hold the helper functions. You will place these helper libraries in a directory called `helperfuncs`, a subdirectory to the root directory of your CMS.

The `helperfuncs` directory will contain two files: `dom.php`, which you will get into later in this tutorial, and `database.php`, which will contain four functions, shown in Listing 3.

### Listing 3. Four helper functions contained in `database.php`

```
$dbconn = db_connect();
execute($dbconn, $sql);
getRow($result);
db_disconnect($dbconn);
```

Since you will be using these four functions so often throughout this tutorial, it's best to learn about them here. The first, `db_connect()`, will connect to Derby with `cmsuser` using its password `cmsspass`, return a database identifier, and select the

new CMS database, as demonstrated in `createTestTable.php` you created earlier. The second, `execute($dbconn, $sql)`, takes the database identifier returned from the call to `db_connect()`, and a SQL statement. This function will execute the SQL statement and return the result by executing `odbc_exec`, as demonstrated in `createTestTable.php`. The third, `getRow($result)`, will return the next row in the result by calling `odbc_fetch_array($result)`. The fourth and last function, `db_disconnect($dbconn)`, takes the same database identifier mentioned in the first function above. This function commits all of the changes, then closes the database, as demonstrated in `createTestTable.php`.

Create a `database.php` file, add the above four functions to it, and place it in the `helperfuncs` directory.

## Creating templates for the CMS

Each page displayed by the Web browser in this tutorial, including the generated pages, will reference the `header.php` and `footer.php` templates. This will help keep the PHP scripts clean and easy to follow. You'll learn the usage of these files later in the tutorial.

Next, create a `header.php` file, containing the code shown in Listing 4, and place it in the root directory of the CMS.

### Listing 4. Header template file

```
<html>
<title><?php echo $title; ?></title>
<body>
<center>
<h1><?php echo $title; ?></h1></center>
<table width="750px" align="center"><tr><td>
```

Now create a `footer.php` file, containing the code shown in Listing 5, and place it in the root directory of the CMS.

### Listing 5. Footer template file

```
</td></tr>
<tr><td align="center">
<font size="2px"><br>
<center>Copyright 2005, TylerCo.</center>
</font>
</td></tr></table>
</body></html>
```

The `header.php` and `footer.php` files will be imported to all files that submit output to the browser by calling `require ( 'header.php' )` before the page

contents and `require( 'footer.php' )` after the page contents.

---

## Section 3. Managing content

Now that you have your database and database helper functions set up, you are ready to start coding the CMS. In this section, you'll learn how to create the forms that will be used as input to add new content and edit the CMS. Then you'll go over some error detection to avoid losing previously saved content and to delete unwanted pages. Once you can add, edit, and delete content, you'll learn how to view your new content, as well as create the central CMS control page.

### Adding content

Let's start by creating the "Add New Page" page to the CMS. Create a `addPage.php` file, and place it in the root directory of the CMS. Now you need to create the appropriate form with required fields to update the database, as shown in Listing 6.

#### Listing 6. Form used to collect the data needed to add an entry to the database

```
<?php
$title = "Add New Page";
require('header.php');
?>
<form action="savePage.php" method="post">

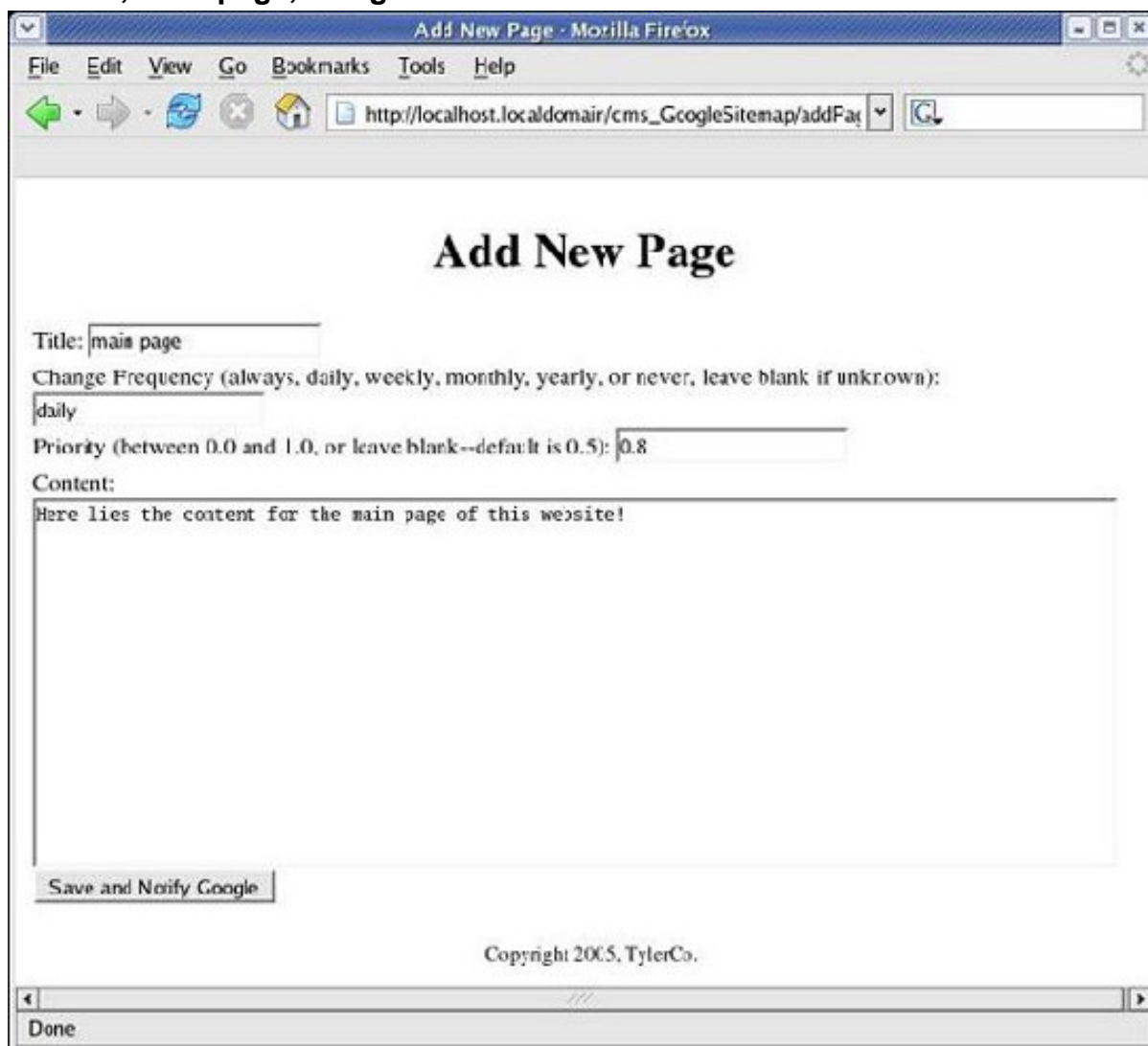
Title: <input name="name"><br>
Change Frequency (daily, weekly, monthly, yearly, or never,
leave blank if unknown): <input name="changefreq"><br>
Priority (between 0.0 and 1.0, or leave
blank--default is 0.5): <input name="priority"><br>
Content:<br>
<textarea name="content" cols="100" rows="15">
</textarea>
<input type="submit" value="Save and Notify Google" \>
<input type="hidden" name="origin" value="add" />
</form>
<?php
require('footer.php');
?>
```

Notice how the `$title` variable gets set before `require( 'header.php' )` gets called, allowing you to dynamically change the `<title>` and `<h1>` elements of the header. Also, take a look at the hidden input field, `origin`. This will notify `savePage.php` which script called it, so that `savePage.php` will handle the data appropriately.

You now have a form that will allow you to enter a title, which will be used as the page name and index into the database. Change Frequency and Priority have to do with the sitemap, which you will learn about later in the tutorial, and notice also Content on the form, which will be the content of the Web page.

The form, as displayed by the Web browser, is shown in Figure 2.

**Figure 2. The Add New Page form as displayed by the Web browser, with a new file, main page, being added to the CMS**



When the button gets pressed, the form data will be submitted to `savePage.php`.

## Saving new content

Now you need to take the data submitted from `addPage.php` and store it in the

database. You will do this through `savePage.php`, which will reside at the root directory of the CMS.

### Listing 7. Takes the data submitted and inserts a new record to the CMS table

```
<?php
include('helperfuncs/database.php');
...
$dbconn = $db_connect();
...
$result = execute($dbconn,
    "insert into CMS
    values ( '". $_POST['name'] ."' ,
            '". $_POST['content'] ."' ,
            '". $_POST['lastmod'] ."' ,
            '". $_POST['changefreq'] ."' ,
            '". $_POST['priority'] ."' );");
if(!$result) die("error adding page");
...
```

This script will connect to the database and insert a new record based on the data sent via POST from `addPage.php`.

The new page, "main page," has now been successfully added to the CMS.

## Editing content

Now that you can add new content to the Web site, you will need a way to edit it. Create a `editPage.php` file and place it in the root directory of the CMS. This form is similar to `addPage.php`, but it first has to check if the page exists.

### Listing 8. Determines whether the page selected for editing exists

```
<?php
include('helperfuncs/database.php');
$dbconn = db_connect();
$result = execute($dbconn,
    "select * from CMS where name='". $_GET['name'] ."'");

if($row = getRow($result)){
    $title = $_GET['name'];
}
else{
    $title = "Page doesn't exist";
    require('header.php');
    echo "The requested page doesn't exist";
    require('footer.php');
    exit;
}
...
db_disconnect($dbconn);
...
```

You have now determined whether the page exists by making sure that the result of

selecting, based on the name of the page you want to edit, contains rows. If not, the Web browser will tell you. If so, the form, as shown in Listing 9, will be displayed, instead.

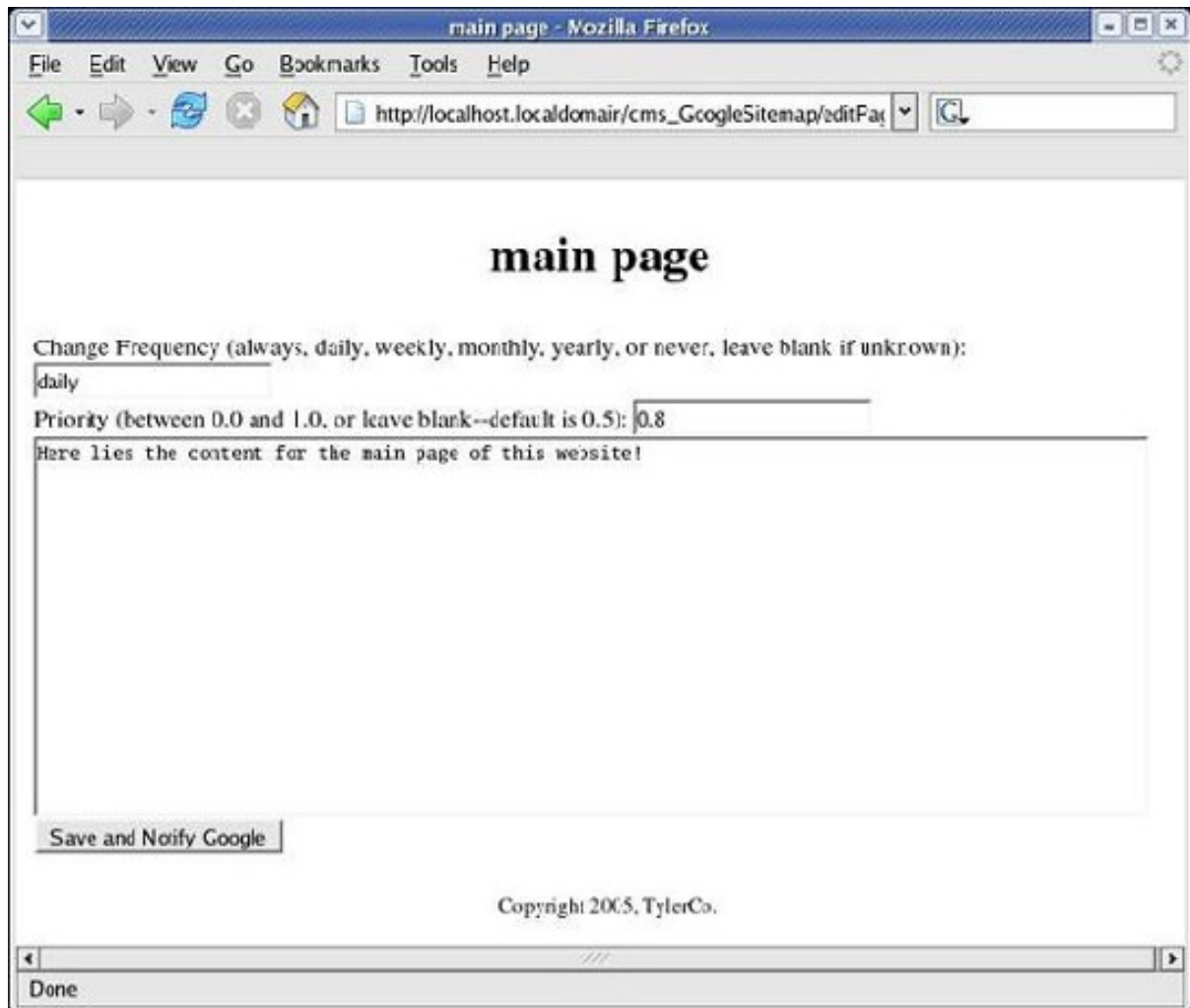
### Listing 9. Displays a form used to edit the content

```
... $content = $row['content'];
}

require('header.php');
?>
<form action="savePage.php" method="post">
Change Frequency (daily, weekly, monthly, yearly, never,
leave blank if unknown): <input name="changefreq" value="<?php echo
$row['changefreq'];?>"><br>
Priority (between 0.0 and 1.0, or leave blank--default is 0.5):
<input name="priority" value="<?php echo $row['priority'];?>"><br>
<input type="hidden" name="name" value="<?php echo $title?>" />
<input type="hidden" name="origin" value="edit" />
<textarea name="content" cols="100" rows="15">
<?php echo $content; ?>
</textarea>
<input type="submit" value="Save and Notify Google" \>
</form>
<?php
require('footer.php');
?>
```

If the page exists, you now have the ability to edit its content. The output of the form from Listing 9 is shown in Figure 3.

### Figure 3. Output from the browser, displaying the form used to edit contents of "main page"



You now have a method to edit contents of "main page" and other pages in your CMS. Notice that the difference between `addPage.php` and `editPage.php` is that the name of the page is transferred using a `<input type="hidden" ... />` element. Note also that the value of the origin input element is `edit`, instead of `add`, which will communicate to `savePage.php` that the given data should be used to update the database.

## Saving edited content

This snippet of `savePage.php` will be used to take the given information and use it to update the database, as shown by Listing 10.

### Listing 10. Modifies the content of the desired page

```
<?php
include( 'helperfuncs/database.php' );
```

```

...
$dbconn = db_connect();
...
$title = "Successful Edit";
$result = execute($dbconn,
    "update CMS
    set content='". $_POST['content'] ."',
    lastmod='". $_POST['lastmod'] ."',
    changefreq='". $_POST['changefreq'] ."',
    priority='". $_POST['priority'] ."'
    where name='". $_POST['name'] ."'");
if(!$result) die("error saving page");
...
db_disconnect($dbconn);
...

```

Now that you have the data from `editPage.php`, which verified that the page already exists, you can connect to the database and update the row using an update statement.

You now have the ability to edit the content of a page in the CMS.

## Avoid adding duplicates

In order to not mistakenly erase previously made pages, you need a way to detect whether a page already exists before you try to add it.

First you will check if you are adding a page that already exists, as shown in Listing 11.

### Listing 11. Make sure a page that already exists doesn't get overwritten

```

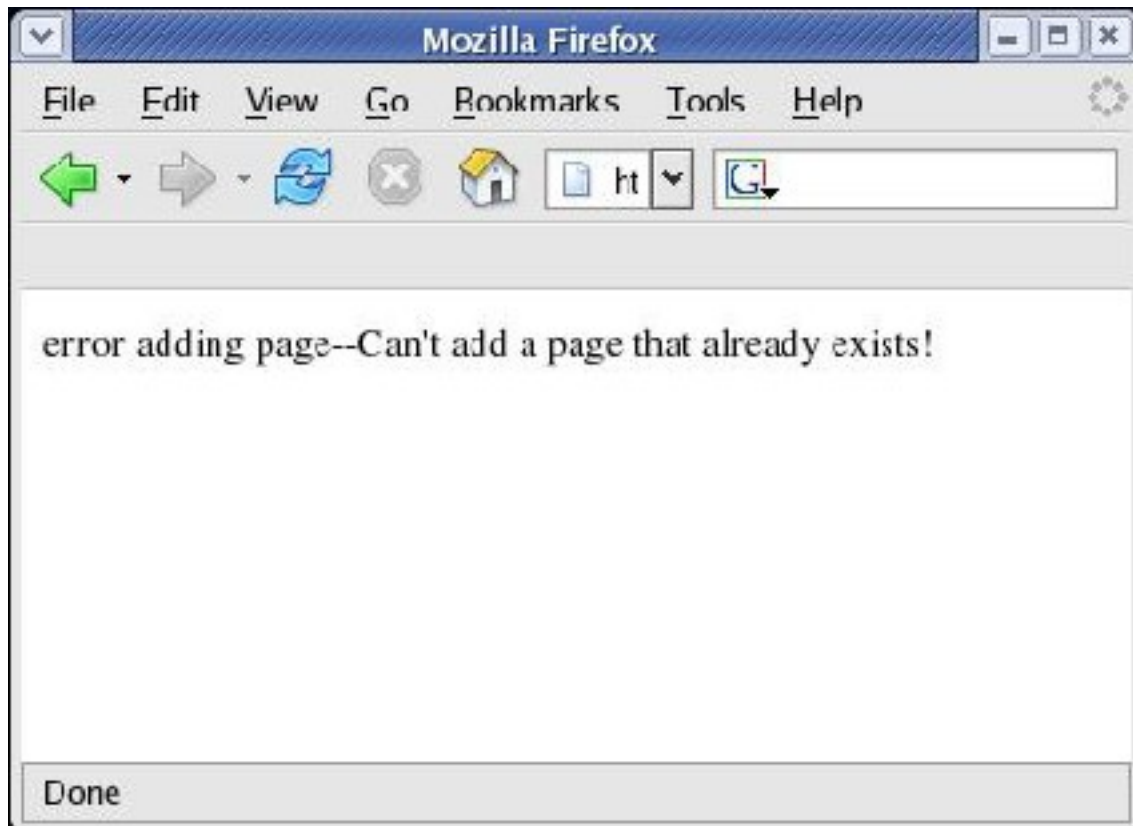
...
$result = execute($dbconn,
    "select * from CMS
    where name='". $_POST['name'] ."'");

if($row = getRow($result)){
    if($_POST['origin'] == "edit"){
        $title = "Successful Edit";
    ...
    }
    else die("error adding page--Can't add a page that already exists!");
}
...
db_disconnect($dbconn);
...

```

This is where you will use the hidden input elements from [Adding Content](#). If the page exists and the originating script was `addPage.php`, the script will die, giving the output shown in Figure 4.

### Figure 4. Output from the script where the CMS attempted to overwrite an existing page



The CMS communicates that a duplicate page cannot be added to the CMS. This ensures that you cannot lose content by mistakenly adding a duplicate page. From here, simply select Back on the browser and rename the page.

## Avoid editing a page that doesn't exist

On the other hand, it's undesirable to edit a page that doesn't exist. This can happen if you edit a page in one browser, open a new browser and delete the same page, go back to the other browser and try to edit it. See Listing 12 for how to detect and prevent this.

### Listing 12. Detecting when the CMS tries to edit a page that doesn't exist

```
...
$result = execute($dbconn,
    "select * from CMS
    where name='". $_POST['name'] . "'");

if($row = getRow($result)){
    if($_POST['origin'] == "edit"){
        $title = "Successful Edit";
    }
}
else{
    if($_POST['origin'] == "add"){
        $title = "Successfully Added New Content";
    }
}
```

```
...
    }
    else die("error editing page--Can't edit a page that doesn't exist!");
...
db_disconnect($dbconn);
...
```

Here's where you use the other hidden input element from [Editing content](#). If the page doesn't exist, the script will die, giving output similar to that shown in Figure 4.

## Deleting content

You can now add and edit content, and sometimes it's necessary to delete content. Create another file, `deletePage.php`, and place it at the root directory of the CMS.

### Listing 13. Deleting a page in the CMS

```
<?php
include('helperfuncs/database.php');
...
$dbconn = db_connect();

$result = execute($dbconn,
                 "delete from CMS
                 where name='".$_GET['name']."'");
if(!$result) die("error deleting page");
...
db_disconnect($dbconn);
...
```

To delete a page in the CMS, connect to the database and remove it using a delete statement. The delete statement in Listing 13 will delete the record matching the name of the page.

## Serving a page

Now that you have pages for your Web site, how will you view them? You will retrieve them first by creating another file, `page.php`. Place this in the root directory of the CMS. This script will be the means for viewing pages in the Web site.

### Listing 14. Retrieving the page content from the database and displaying it to the browser

```
<?php
include('helperfuncs/database.php');
$dbconn = db_connect();
$result = execute($dbconn,
                 "select content from CMS
                 where name='".$_GET['name']."'");
```

```

if($row = getRow($result)){
    $title = $_GET['name'];
    $content = $row['CONTENT'];
}
else{
    $content = "The requested page doesn't exist";
    $title = "Page doesn't exist";
}
require('header.php');
print($content);
require('footer.php');

db_disconnect($dbconn);
?>

```

To view the page, you'll need to retrieve the row associated with the page name received from the `GET` array. If it exists, extract the content from the result of the query. Tailor the title based on the name of the page and output the content to the database, in between the `header.php` and `footer.php` files. See Figure 5 for the output of viewing the "main page."

**Figure 5. Browser output from viewing the contents of "main page" from the CMS**



Great! Now you can view the contents of the sample "main page" in your CMS.

## Pulling it together: The CMS control page

Next, you'll need a CMS control page that will streamline everything for you. This page will connect to the database and provide links to `addPage.php`. It will also provide links for each page in the CMS that will allow you to view, edit, or delete each page. Create a `cmsPages.php` file and place it at the root directory of your CMS. See Listing 15 for its contents.

**Listing 15. Provide the links to all the commands and PHP scripts created thus far**

```

<?php
include('helperfuncs/database.php');
$dbconn = db_connect();

$title = "View Pages in CMS";
require('header.php');
echo "<h4>Click the link containing the command you
would like to execute</h4>";

$result = execute($dbconn,
                 "select name from CMS;");

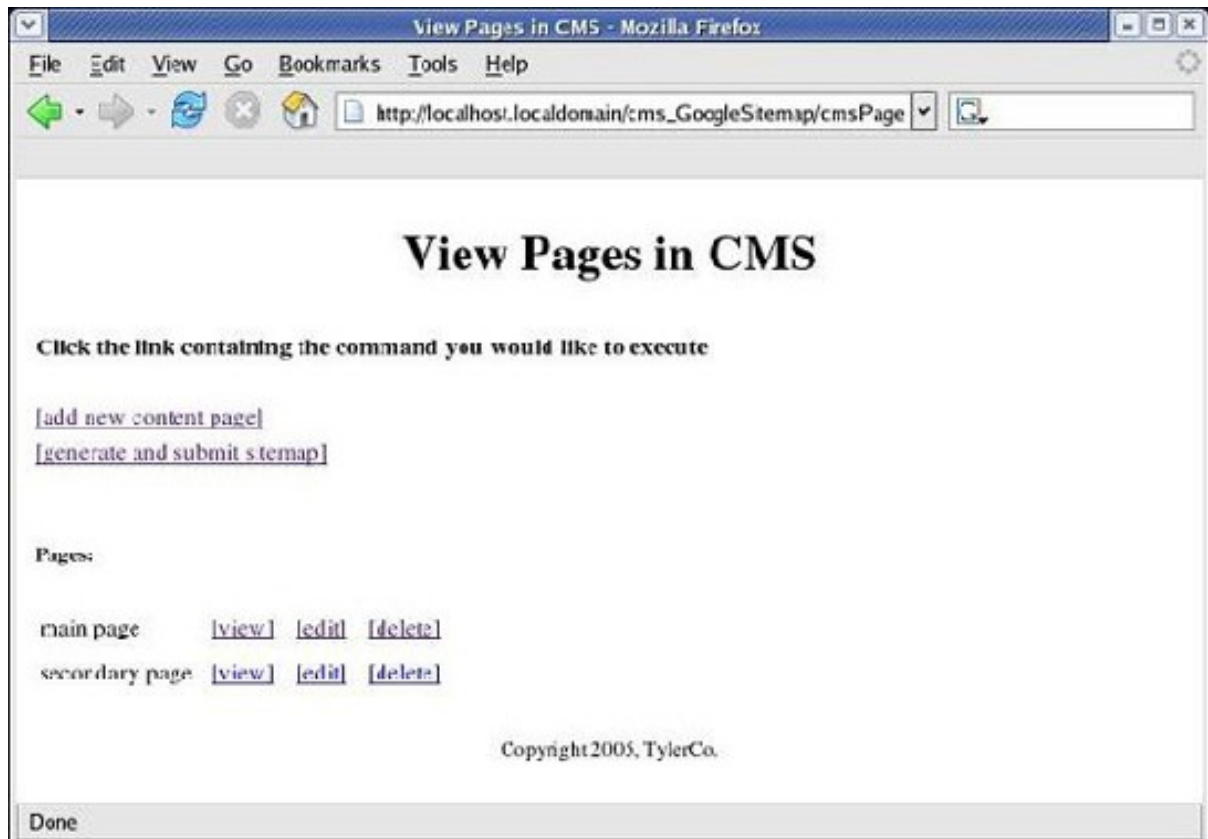
print("<a href='addPage.php'>[add new content
page]</a><br>");
print("<a href='submitSitemap.php'>[generate and
submit sitemap]</a><br>");
if($row = getRow($result)){
    print("<br><h5>Pages:</h5>");
    print("<table>");
    for(; $row; $row = getRow($result)){
        $name = $row['NAME'];
        print("
            <tr><td>$name&nbsp;&nbsp;&nbsp;</td>
            <td><a href='page.php?name=$name'>[view]</a>
&nbsp;</td>
            <td><a href='editPage.php?name=$name'>[edit]</a>
&nbsp;</td>
            <td><a href='deletePage.php?name=$name'>[delete]</a>
</td><tr>");
    }
    print("</table>");
}
require('footer.php');

db_disconnect($dbconn);
?>

```

Note how the commands to view, edit, or delete each page are created by querying the database, then display each entry, row by row, until each file in the CMS is displayed.

**Figure 6. Browser output of the CMS control page**



Now you see the entire CMS pulled together in one control page where you can add a new page, submit to Google Sitemaps, or view, edit, or delete a page in your CMS.

---

## Section 4. Google Sitemaps

This section will describe Google Sitemaps and its syntax. It will also describe how to create the initial sitemap and load it using the Document Object Model (DOM).

You've got a CMS with your whole Web site in it, and now you need to efficiently get it to Google for indexing. Not only do you need the main page indexed but you need all of your Web pages indexed, even if they aren't linked together. A Google sitemap is just the way to do it.

A Google sitemap is an XML document that contains all the URLs of your Web site, or all the links to all the pages in your CMS, along with basic information about when it was last modified, change frequency, and priority (how fast you want it indexed -- for example, your index or main page should be the highest priority).

## Listing 16. Sample sitemap XML document

```
<urlset xmlns="http://www.google.com/schemas/sitemap/0.84"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.google.com/schemas/sitemap/0.84
http://www.google.com/schemas/sitemap/0.84/sitemap.xsd">
  <url>
    <loc>http://localhost.localdomain/cms_GoogleSitemap/
page.php?name=main_page</loc>
    <lastmod>2005-09-16</lastmod>
    <changefreq>daily</changefreq>
    <priority>1.0</priority>
  </url>
</urlset>
```

Before working with Google Sitemaps, it is important to first examine all of the XML elements of a sitemap, which you will do in the remainder of this section:

- `urlset`
- `url`
- `loc`
- `changefreq`
- `priority`

## Syntax

The root XML element of a sitemap is `<urlset ... >`. The `urlset` element will enclose all of your `url` elements, which hold basic information about that URL for Google. This is how Google will know you are submitting a correct sitemap that will be parsable by Google's XML parsing technology. The `urlset` element has three attributes whose values are constant: `xmlns`, `xmlns:xsi`, and `xsi:schemaLocation`. Their values are `http://www.google.com/schemas/sitemap/0.84`, `http://www.w3.org/2001/XMLSchema-instance`, and `http://www.google.com/schemas/sitemap/0.84` `http://www.google.com/schemas/sitemap/0.84/sitemap.xsd`, respectively.

The `url` element is the child of the `urlset` element, and there can be any number of child `url` elements. The `url` elements will hold the URLs of your entire Web site. A `url` element has no attributes, and encloses four elements, one of which is required: `loc`. The `lastmod`, `changefreq`, and `priority` elements are optional.

## Location element

The location element (`loc` element) has no attributes and just contains a single URL for the page. There is no restriction on the format of the URL, except that characters such as the ampersand (&) be encoded as `&amp;`.

## Last modified element

The last modified element (`lastmod` element) holds the date and, optionally, the time the URL was last modified. The format of the date is YYYY-MM-DD, where YYYY is the year, MM is the month, padded with zeros, and DD is the day, padded with zeros. An example would be 2005-09-16. In your CMS, this will be entered automatically by calling the `date($str)` function. You can retrieve today's date with the following PHP code: `$lastmod = date("Y-m-d");`.

The `lastmod` element is similar to the `loc` element in that it, too, has no attributes and contains only the date. In fact, this is how you set the `lastmod` element in the `savePage.php` script. See Listing 17 for placing of this code.

## Change frequency element

The change frequency element (`changefreq` element) specifies how often the URL gets modified. In your CMS, you can enter it in yourself or create an algorithm to create it automatically. The `changefreq` element has six possible values: always (pages that always change), daily, weekly, monthly, yearly, and never.

Errors for this element (once submitted to `savePage.php` from `addPage.php` or `editPage.php`) can be corrected.

### Listing 17. The `changefreq` element

```
...
$dbconn = db_connect();

$lastmod = date("Y-m-d");
switch($_POST['changefreq']){
  case 'always':
  case 'daily':
  case 'weekly':
  case 'monthly':
  case 'yearly':
  case 'never':
    $changefreq = $_POST['changefreq'];
    break;
  default:
    $changefreq = '';
}
...
$result = execute($dbconn,
                  "insert into CMS
                  values ( '".$_POST['name']."' ,
...
}
```

The method that creates the sitemap can leave out the optional `changefreq` element if it has the empty string value `''`.

## Priority element

The priority element tells Google how much of a hurry you are in to get your new content indexed in its system. The default value of this element (if left blank), is 0.5, with a highest possible value of 1.0, and a lowest possible value of 0.

Errors for this element, before sending the sitemap off to Google, can be corrected.

### Listing 18. The priority element

```
if($_POST['priority'] < 0 || $_POST['priority'] > 1)
    $priority = '';
else
    $priority = $_POST['priority'];
```

If the value is not in its expected range, set to the empty string, `''` and leave it out of the sitemap. Google will, therefore, interpret it as being its default value of 0.5.

## Creating the initial sitemap file

Now you need to create the sitemap in PHP. Create a `dom.php` file and place it in the `helperfuncs` directory, where `database.php` resides. You'll be doing this using the DOM.

### Listing 19. Create the initial sitemap file using the DOM

```
define('SITEMAP', '/var/www/html/sitemap.txt.xml');
...
function create_edit_sitemap($loc, $lastmod,
    $changefreq='', $priority='') {
    ...
    $dom = new DOMDocument('1.0');
    $root = $dom->createElement('urlset');
    $root->setAttribute('xmlns',
        'http://www.google.com/schemas/sitemap/0.84');
    $root->setAttribute('xmlns:xsi',
        'http://www.w3.org/2001/XMLSchema-instance');
    $root->setAttribute('xsi:schemaLocation',
        'http://www.google.com/schemas/sitemap/0.84
http://www.google.com/schemas/sitemap/0.84/sitemap.xsd');
    $dom->appendChild($root);
    ...
    $dom->save(SITEMAP);
}
```

To create the sitemap XML document, you'll need to first create a new

DOMDocument. Create a `urlset` element, and set the three attributes to its defined values as shown in Listing 19. The `urlset` element must then be appended to the DOMDocument object as the top-level element that will enclose all the `url` elements. The DOMDocument is then saved to a Web-accessible location so Google can read it.

## Loading the sitemap file

Once you have a sitemap created, you will want to add `url` elements to it, or edit or remove existing ones. See Listing 20 for loading a previously created DOMDocument.

### Listing 20. Load an existing DOMDocument

```
define('SITEMAP', '/var/www/html/sitemap.txt.xml');
...
function create_edit_sitemap($loc, $lastmod,
    $changefreq='', $priority='') {
    if(is_file(SITEMAP)){
        $dom = DOMDocument::load(SITEMAP);
        $root = $dom->getElementsByTagName('urlset')->item(0);
    }
    else{
        $dom = new DOMDocument('1.0');
        $root = $dom->createElement('urlset');
    }
    ...
}
```

First, you need to make sure the the `SITEMAP` file exists, load it, and retrieve the top-level `urlset` element, as shown in Listing 20.

---

## Section 5. Using Google Sitemaps from CMS

In this section, you will learn how to edit, modify, and delete a `url` element in a sitemap, and submit the sitemap to Google -- all automatically through the CMS.

### Adding a sitemap URL

Now that you've created a sitemap, you need to be able to add `url` elements to it. Add the following, as shown in Listing 21, to the `create_edit_sitemap` function you started on in the `dom.php` file.

### Listing 21. Add a url element to a sitemap

```

define('URLBASE',
'http://localhost.localdomain/page?name=');
...
function create_edit_sitemap($loc,
$lastmod, $changefreq='', $priority='') {
    if(is_file(SITEMAP)){
        $dom = DOMDocument::load(SITEMAP);
        $root = $dom->getElementsByTagName('urlset')->item(0);
    } else {
        $dom = new DOMDocument('1.0');
    }
    ...
    $dom->appendChild($root);
    ...
    $loc = URLBASE . $loc;
    ...
    $node = $dom->createElement('url');
    $root->appendChild($node);

    $node->appendChild($textEle = $dom->createElement('loc'));
    $textEle->appendChild($dom->createTextNode($loc));

    $node->appendChild($textEle =
$dom->createElement('lastmod'));
    $textEle->appendChild($dom->createTextNode($lastmod));

    if($changefreq != '') {
        $node->appendChild($textEle =
$dom->createElement('changefreq'));
        $textEle->appendChild($dom->createTextNode($changefreq));
    }
    if($priority != '') {
        $node->appendChild($textEle =
$dom->createElement('priority'));
        $textEle->appendChild($dom->createTextNode($priority));
    }

    $dom->save(SITEMAP);
}

```

You will need to create a new element name: `url`. Note that every element you create has to be appended as a child to the element it is a child to, hierarchically. For example, the `url` element is added to the `$root`, and the `loc`, `lastmod`, `changefreq` and `priority` elements are added to the `url` element, `$node`.

Also notice how you changed the `$loc` value to contain the full URL by concatenating `URLBASE` to `$loc`, making the value contained in `$loc` the full URL to the Web page.

If `$changefreq` or `$priority` are the empty string, `'`, they will be left out and not be added as children to the `url` element, `$node`. This is OK because these elements are optional.

## Retrieving a sitemap URL

Now that you can add `url` elements to the sitemap, you will need the ability to

retrieve them, so that they can be modified or removed later.

### Listing 22. Loop through the url elements of a sitemap to find the matching element

```
...
function create_edit_sitemap($loc, $lastmod, $changefreq='', $priority='') {
    if (is_file(SITEMAP)) {
        $dom = DOMDocument::load(SITEMAP);
        $root = $dom->getElementsByTagName('urlset')->item(0);
    } else {
        $dom = new DOMDocument('1.0');
    }
    $dom->appendChild($root);

    $children = $dom->getElementsByTagName('url');
    $i = 0;
    $loc = URLBASE . $loc;
    $locCompare = "<loc>".$loc."</loc>";
    while ($i < $children->length) {
        $find = $dom->saveXML(
            $children->item($i)->getElementsByTagName('loc')->item(0));
        if ($find == $locCompare)
            break;
        $i++;
    }

    if ($i < $children->length) {
    ...
}
```

Once the XML document has been loaded into a DOM object, you need to loop through all the `url` elements, as shown in Listing 22. Notice how you retrieve all the `url` elements from the global `$dom` object, and set up the `$locCompare` variable, turning the `$loc` variable into the XML format needed to compare to the `loc` element children of the `url` elements. You then loop through the children until you have exhausted all the `url` elements, or the condition `$find == $locCompare` is true, which means the condition, `$i < $children->length`, will be true also, indicating that a match has been found.

## Modifying a sitemap URL

Now that you have found a matching `url` element, chances are it will need to be modified in time.

### Listing 23. Modify a url element

```
...
if ($i < $children->length) {
    $node = $children->item($i);
    ...
    $root->removeChild($node);
}
}
```

```
$node = $dom->createElement('url');  
$root->appendChild($node);  
...
```

A match has been found, so remove the old one and create a new one, effectively modifying the `url` element.

## Deleting a sitemap URL

When you delete a page from your CMS, you will also need to delete it from the sitemap. Add this method to your `dom.php` file, as shown in Listing 24.

### Listing 24. Delete a url element entry from a sitemap

```
function delete_sitemap_entry($loc){  
    if(!is_file(SITEMAP))  
        return;  
    $dom = DOMDocument::load(SITEMAP);  
    $root = $dom->getElementsByTagName('urlset')->item(0);  
  
    $children = $dom->getElementsByTagName('url');  
    $i = 0;  
    $loc = URLBASE . $loc;  
    $locCompare = "<loc>".$loc."</loc>";  
    while($i < $children->length){  
        $find = $dom->saveXML(  
            $children->item($i)->getElementsByTagName('loc')->item(0));  
        if($find == $locCompare)  
            break;  
        $i++;  
    }  
  
    if($i >= $children->length)  
        return;  
  
    $node = $children->item($i);  
    $root->removeChild($node);  
  
    $dom->save(SITEMAP);  
}
```

The method `delete_sitemap_entry()` takes the name of the file to be removed. The XML document is loaded and the child `url` elements get looped through. If a match is found, that element is removed, and the new XML document is saved. If the sitemap file doesn't exist or a match isn't found, nothing happens and the function returns.

## Submitting sitemap to Google

Now that the sitemap is complete, it's time to submit it to Google. Add the function shown in Listing 25 to your `dom.php` file.

## Listing 25. Submitting the sitemap to Google

```
define('SITEMAP_URL',
    'http%3A%2F%2Flocalhost.localdomain%2Fsitemap.txt.xml');

function notify_google(){
    $url =
        "http://www.google.com/webmasters/sitemaps/ping?sitemap=" .
        SITEMAP_URL;
    $fd = fopen($url, "rb");
    ...
}
```

First you need to calculate the URL that you want to open with the function `fopen($url, "rb")`. This is done by concatenating the Google submission string `http://www.google.com/webmasters/sitemaps/ping?sitemap=` to `SITEMAP_URL`, as shown in Listing 25.

## Verifying submission of sitemap to Google

The call to `fopen` returns a handle to the confirmation Web page. To verify that the submission was successful, you need to read that confirmation page.

## Listing 26. Reading the confirmation page -- success or failure

```
function notify_google(){
    ...
    $fd = fopen($url, "rb");
    $buffer = '';
    while(!feof($fd))
        $buffer .= fgets($fd, 4096);
    fclose($fd);
    if(strpos($buffer, "Sitemap Notification Received"))
        return "true";
    return "false";
}
```

The page will be read into the `$buffer` variable in the `while` loop. When the loop completes and `$fd` is closed using `fclose()`, the contents of the Google confirmation page will be in the variable `$buffer`. If `$buffer` contains the string `Sitemap Notification Received`, as shown in Listing 26, then the sitemap submission was successful; return `"true"`. Otherwise, it was a failure; return `"false"`.

## Deleting a sitemap URL from CMS

Now that you have the methods to remove sitemaps in your helper library, `dom.php`, you need to know where to add these in the CMS. You will use the `delete_sitemap_entry()` function in `deletePage.php`.

## Listing 27. Deleting the corresponding url element from the sitemap

```
<?php
include( 'helperfuncs/database.php' );
include( 'helperfuncs/dom.php' );
...
if(!$result) die("error deleting page");

delete_sitemap_entry($_GET['name']);
...
```

Once you have verified that the page has been deleted in the CMS, you can go ahead and delete the `url` element in the sitemap.

## Creating or editing a sitemap URL from CMS

To create or edit a `url` element in the sitemap, revisit `savePage.php`.

## Listing 28. Creating or editing the sitemap

```
<?php
include( 'helperfuncs/database.php' );
include( 'helperfuncs/dom.php' );
...
} else die("error adding page--Can't add a page that already exists!");
}

require( 'header.php' );

echo "Command was successful.<br>";
create_edit_sitemap($_POST['name'], $lastmod, $changefreq, $priority);
...
require( 'footer.php' );

db_disconnect($dbconn);
?>
```

In the Google Sitemaps section, you calculated the value of `$lastmod` using the `date()` function, and you did error detection and correction for `$changefreq` and `$priority`. Now you pass these values, along with the name of the page to the `create_edit_sitemap()` function, which will create or edit a `url` element for you.

## Submitting sitemap to Google from CMS

After you create or edit the `url` element for the page, you need to notify Google of the newly updated sitemap.

## Listing 29. Submitting the sitemap to Google

```
...
echo "Command was successful.<br>";
create_edit_sitemap($_POST['name'], $lastmod, $changefreq, $priority);
if(notify_google() == "true")
...
require('footer.php');
db_disconnect($dbconn);
?>
```

A simple call to `notify_google()` will notify Google of the newly updated sitemap.

## Verifying submission of sitemap to Google from CMS

It's important to verify a valid submission so your pages get indexed into Google without any problems.

### Listing 30. Verifying a valid submission

```
...
if(notify_google() == "true")
    echo "Google was successfully notified of the changes.<br>";
else
    echo "There was an error notifying Google of changes.<br>";
echo 'Return to <a href="cmsPages.php">View CMS</a> page.';

require('footer.php');
db_disconnect($dbconn);
?>
```

If Google received the submission successfully, the `notify_google()` function will return `true`; otherwise, it will return `false` on failure. Simply display to the browser whether the submission was successful, so you will know to resubmit later or debug your code. See Figure 7 for browser output of a successful CMS edit and Google sitemap submission.

### Figure 7. Browser output showing successful edit and Google sitemap submission



The CMS now automatically notifies Google of changed content.

## Updating and submitting sitemap from the CMS control page

Let's move on to updating the entire sitemap and submitting it fresh to Google. Create the last file, `submitSitemap.php`, and place it at the root directory of your CMS.

### Listing 31. Update the entire sitemap and submit to Google

```
<?php
include('helperfuncs/database.php');
include('helperfuncs/dom.php');
$dbconn = db_connect();

$result = execute($dbconn,
    "select * from CMS;");
for($row = getRow($result); $row; $row = getRow($result)){
    create_edit_sitemap($row['NAME'], $row['LASTMOD'],
        $row['CHANGEFREQ'], $row['PRIORITY']);
}

if(notify_google() == "true"){
    $content = "New Sitemap was successfully submitted to Google.<br>";
    $title = "Submission Successful";
}
else{
    $content =
        "There was an error submitting the new sitemap to Google.<br>";
    $title = "Submission Error";
}
require('header.php');

echo "$content Return to <a href='cmsPages.php'>View CMS</a> page.";

require('footer.php');
db_disconnect($dbconn);
```

```
?>
```

First you will need to iterate over all the page names in your database. For every page, you will need to call `create_edit_sitemap()`, passing in the data from the database. Next, you'll notify Google to verify that Google received it successfully. See Figure 8 for browser output.

**Figure 8. Browser output showing successful creation and submission to Google of a sitemap**



Now you can create, edit and submit the entire Google sitemap with one button click.

---

## Section 6. Summary

In this tutorial, you successfully created a CMS to automate the creation and submission of a sitemap to Google. You also enabled the CMS to create and edit your Web pages.

Along the way, you learned the syntax of Google Sitemaps, so you could integrate the CMS and the sitemap, allowing simultaneous updates and submits to Google Sitemaps behind the scenes.

## Downloads

Description	Name	Size	Download method
Source code for CMS	os-cms.source.zip	8 KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- According to the [About Google Sitemaps page](#), it "is an experiment in Web crawling. By using Sitemaps to inform and direct our crawlers, we hope to expand our coverage of the Web and speed up the discovery and addition of pages to our index."
- For a series of tutorials designed to broaden your PHP skills, see "[Learning PHP](#)."
- "Getting started with an open source CMS" explains how to build an open source content management system with Tomcat and Jakarta Slide. [Part 1](#) shows how to download, install, and start working with some of the tools you will need. [Part 2](#) shows how to download the sources, set up Eclipse, and build the Jakarta Slide open source CMS. And [Part 3](#) describes how to customize Jakarta Slide with Eclipse.
- Find more resources for PHP developers in [Top project resources](#) on developerWorks.
- Learn how to [configure Apache and PHP](#). Both install PHP V4; install V5 instead, and you shouldn't run into problems.
- Learn how to install and configure PHP on Windows with the developerWorks article "[Connecting PHP applications to Apache Derby](#)."
- "[Develop IBM Cloudscape and DB2 Universal Database applications with PHP](#)" shows how to configure IBM Cloudscape V10.0 and IBM DB2 Universal Database V8.2 servers for access from PHP V4.x and V5.x.
- For information about creating PHP forms, visit [W3schools: PHP Forms](#).
- For a great DOM reference, see the [PHP Manual: DOM Functions](#).
- "[SAX-like apps in PHP](#)" explains how to work with XML files in PHP by building and setting handler functions and creating a parser.

## Get products and technologies

- [Download](#) the [Apache Web Server](#) at the [Apache Foundation](#).
- [Download](#) the latest version of PHP from [PHP](#).
- Learn more about [IBM Cloudscape](#) and [download](#) the latest version of the open-standards, small-footprint database.
- Learn the system requirements and where to [download](#) the free [ODBC for Cloudscape](#) driver.

## About the author

### Tyler Anderson

Tyler Anderson graduated with a degree in computer science from Brigham Young University in 2004 and is currently in his last semester as a master's of science student in computer engineering. In the past, he worked as a database programmer for DPMG.COM, and he is currently an engineer for Stexar Corp., based in Beaverton, Ore.