SiteClass

SiteClass PHP class mini framework for simple, small websites.

This project has several parts that can function standalone or combined.

- siteautoload.class.php : Autoload classes and reads a .sitemap.php file to initialize the system.
- Database.class.php : provides a wrapper for several different database engines.
- dbTables.class.php : uses the functionality of Database.class.php to make creating tables easy.
- Error and Exception classes
- SiteClass.class.php : tools for making creating a site a little easier. The class provides methods to help with headers, banners, footers and more.

The following database engines are provided:

1. Mysql (depreciated)
2. Mysqli (most rigorously tested)
3. sqlite3 (used for the examples)
4. POD (least tested)

Disclamer

To start, this framework is meant for Linux not Windows. I don't use Windows, like it or have it, so nothing has been tried on Windows.

I use Linux Mint which is an Ubuntu derivative which is a Debian derivative. I have not tried this package on any distributions that do not evolve from Debian.

Install

There are several ways to install this project.

Download The ZIP File

Download the ZIP file from GitHub. Expand it and move the 'includes' directory somewhere.
On a system with Apache2 I usually put the 'includes' directory in the /var/www directory
that Apache creates. Apache also usually creates /var/www/html and makes this the default DocumentRoot.
I put the 'includes' directory just outside of the DocumentRoot. In my servers I have /var/www and then
have my virtual hosts off that directory. That way the 'includes' directory is easily available to all
of my virtual hosts.

If you are testing with the PHP server I put a www directory off my $HOME and put the
'includes' directory there.
I then make my test DocumentRoot off ~/www like ~/www/test.
I `cd` to the test directory and do
`php -S localhost:8080`. I can then use my browser and goto `localhost:8080` and see my
'index.php' file.

Use Composer

If you have Apache or Nginx installed then you should made your project root somewhere
withing your DocumentRoot (/var/www/html for Apache2 on Ubuntu).

Create a directory `mkdir myproject; cd myproject`, this is your project root directory.
Add the following to 'composer.json', just cut and past:

```
{
  "require": {
      "php": ">=5.4",
      "bartonlp/site-class": "dev-master"
  }
}
```

Then run `composer install`.

OR you can just run `composer require bartonlp/site-class:dev-master` which will create
the 'composer.json' for you and load the package like 'install'.

In your PHP file add `require_once($PATH . 'vendor/autoload.php');` where '$PATH'
is the path to the 'vendor' directory like './' or '../' etc.

There are some example files in the 'examples' directory at
'/vendor/bartonlp/site-class/examples' or 'examples' in your project root if you copied
the it there.

Examples

The code shown below can be found in the 'examples' directory at
http://github.com/bartonlp/site-class or from your project root
at 'vendor/bartonlp/site-class/examples' along with
more documentation.

You may want to copy (or move)
the examples and the 'README.html' to your project root directory before running them.

From your project root do:

```
cp -r vendor/bartonlp/site-class/examples .
cp vendor/bartonlp/site-class/README.html .
```

If you have Apache or Nginx installed then you should have made your project root somewhere
withing your DocumentRoot, for example /var/www/html/myproject.

If you don't have Apache or Nginx installed on your computer you can use the PHP server.
Do the following from your project root:

```
php -S localhost:8080
```

Then use your browser by entering `http://localhost:8080/README.html` in the browsers
location bar.

The 'examples' code uses the **sqlite3** engine.
There should be a 'test.sdb' database file in the 'examples' directory already.

I have included a 'sqlite.sql' file that can be run from the command line if you want to
recreate the 'members' table.
You will need to get sqlite3 and get the PHP sqlite packages along with mysql etc.
From the command line in the directory where the SiteClass was downloaded:

```
$ cd examples
$ sqlite3 test.sdb
sqlite> drop table members;
sqlite> .read sqlite.sql
sqlite> .table
members
sqlite> select rowid,* from members;
1|Big|Joe
2|Little|Joe
3|Barton|Phillips
4|Someone|Else
sqlite> .quit
$
```

This should create a new 'members' table in the 'test.sdb' database.

These examples assume that the 'includes' directory is at /var/www as I suggested.
The files in the 'examples' directory use either '../vendor/bartonlp/site-class/includes' or
'../vendor/autoload.php' for the 'composer-*' variants.

There are a number of ways to use the framework:

First you can just use the SiteClass all by itself.

```php
<?php
// test1.php

require-once('/var/www/includes/SiteClass.class.php'); // path to your SiteClass.class.php

$siteinfo = array(
  'siteDomain' => "localhost",
  'siteName' => "Vbox Localhost",
  'copyright' => "2015 Barton L. Phillips",
);

$S = new SiteClass($siteinfo);

list($top, $footer) = $S->getPageTopBottom();
echo <<<EOF
$top
<h1>Test 1</h1>
<p>Stuff</p>
$footer
EOF;
```

That is the simplest usage. You get a generic <head> a blank <header> and a generic
<footer>.
No database or other stuff.

You can extend this by adding a database either by instantiating the Database class directly or
indirectly.

```php
<?php
// test2.php

require-once('/var/www/includes/SiteClass.class.php'); // path to your SiteClass.class.php

$siteinfo = array(
  'siteDomain' => "localhost",
  'siteName' => "Vbox Localhost",
  'copyright' => "2015 Barton L. Phillips",
  'memberTable' => 'members',
  // Add dbinfo to the $siteinfo and SiteClass will instantiate the Database
  'dbinfo' => array(
    'database' => 'test.sdb',
    'engine' => 'sqlite3'
  ),
);

$S = new SiteClass($siteinfo);

list($top, $footer) = $S->getPageTopBottom();

// Do some database operations

$S->query("select fname||' '||lname) from {$siteinfo['memberTable']}";
$names = '';

while(list($name) = $S->fetchrow('num')) {
  $names .= "$name<br>";
}

echo <<<EOF
$top
<h1>Test 2</h1>
<p>$names</p>
$footer
EOF;
```

The above example uses the 'query' and 'fetchrow' methods to do some database operations.

The database could also be instantiated explicitly as follows:

```
<?php
// test3.php

require-once('/var/www/includes/SiteClass.class.php'); // path to your SiteClass.class.php

$siteinfo = array(
  'siteDomain' => "localhost",
  'siteName' => "Vbox Localhost",
  'copyright' => "2015 Barton L. Phillips",
  'memberTable' => 'members',
);

$dbinfo = array(
  'database' => 'test.sdb',
  'engine' => 'mysqli'
);

$siteifno['databaseClass'] = new Database($dbinfo);

// by adding to the $siteinfo arrays 'databaseClass' element we let SiteClass
// know that the database is active.

$S = new SiteClass($siteinfo);

list($top, $footer) = $S->getPageTopBottom();

// Do some database operations

$S->query("select fname||' '||lname from {$siteinfo['memberTable']}");

$names = '';

while(list($name) = $S->fetchrow('num')) {
  $names .= "$name<br>";
}

echo <<<EOF
$top
<h1>Test 3</h1>
<p>$names</p>
<hr>
```

```
$footer
EOF;
```

You can also use the siteautoload.class.php and .sitemap.php to further automate working with the
framework. There is a 'dot-sitemap.php.example' file that is well commented. Copy the file to your
project directory as '.sitemap.php' and edit it to match your needs. There is already a .sitemap.php
file in the 'examples' directory to see it you need to do `ls -a`.

```
<?php
// test4.php

require_once('/var/www/includes/siteautoload.class.php'); // path to siteautoload.class.php
// the siteautoload.class.php first looks for the .sitemap.php file and then sets up class
// Now the class autoloader finds the classes that are required. The .sitemap.php has all th
// information needed to instanntiate the Database class. The $siteinfo array is available
// SiteClass etc.

$S = new SiteClass($siteinfo);

list($top, $footer) = $S->getPageTopBottom();

// Do some database operations

$S->query("select concat(fname, ' ', lname) from {$siteinfo['memberTable']}";

$names = '';

while(list($name) = $S->fetchrow('num')) {
  $names .= "$name<br>";
}

echo <<<EOF
$top
<h1>Test 4</h1>
<p>$names</p>
$footer
EOF;
```

In addition to the SiteClass and Database classes there are several others classes:

- Error

- SqlException
- dbMysql
- dbMysqli
- dbSqlite
- dbPod
- dbTables
- and a file with helper functions.

The dbTables class uses the Database class to make creating tables simple. For example:

```
<?php
require_once('/var/www/includes/siteautoload.class.php'); // path to siteautoload.class.php

$S = new SiteClass($siteinfo);
$T = new dbTables($S);

// Pass some info to getPageTopBottom method
$h->title = "Table Test 5"; // Goes in the <title></title>
$h->banner = "<h1>Test 5</h1>"; // becomes the <header> section
// Add some local css to but a border and padding on the table
$h->css = <<<EOF
  <style>
main table * {
  padding: .5em;
  border: 1px solid black;
}
  </style>
EOF;

list($top, $footer) = $S->getPageTopBottom($h);

// create a table from the memberTable
$sql = "select * from {$siteinfo['memberTable']}";
list($tbl) = $T->maketable($sql);
echo <<<EOF
$top
<main>
<h3>Create a table from the members database table</h3>
```

```
<p>The members table follows:</p>
$tbl
</main>
<hr>
$footer
EOF;
```

The maketable method takes several optional options to help setup the table. Using the options you can
give your table an id or class or set any other attributes. You can also pass 'callback' function
which can modify the rows as they are selected.

The dot sitemap File

The .sitemap.php file is the site configuration file. siteautoload.clas.php looks for the .sitemap.php
file in the current directory. If it does not find it there it moves up to the parent directory. This
continues until the DocumentRoot of the domain is reached. If a .sitemap.php file is not found an
exception is thrown.

Once a .sitemap.php file is found the information in it is read in via 'require_once'. There are several
section to the .sitemap.php file. The first section defines the layout of the directorys. The second
section defines some email information used to send error messages. The third section sets up the
$siteinfo array. This array describes your site or page.

My usual directory structure starts under a www subdirectory. On an Apache2 host the structure looks
like this:

```
/var/www/includes        // this is where the SiteClass resides
/var/www/html            // this is where your php files and js, css etc. directories live
/var/www/html/includes   // this is where 'headFile', 'bannerFile', 'footerFile' and child c
/var/www/html/otherstuff // other directories. These directories can have their own .sitemap
```

If I have multiple virtual hosts they are all off the /var/www directory instead of a single html
directory.

How the xxxFile files look

In the .sitemap.php's $siteinfo array there can be three elements that describe the location of
special files. These files are 1) 'headFile', 2) 'bannerFile' and 3) 'footerFile'.

I put the three special file in my /var/www/html/includes directory.
Here is an example of my 'headFile':

```
<?php
// head.i.php for bartonphillips.com

$pageHeadText = <<<EOF
<head>
  <title>{$arg['title']}</title>
  <!-- METAs -->
  <meta name=viewport content="width=device-width, initial-scale=1">
  <meta charset='utf-8'/>
  <meta name="copyright" content="$this->copyright">
  <meta name="Author"
    content="Barton L. Phillips, mailto:bartonphillips@gmail.com"/>
  <meta name="description"
    content="{$arg['desc']}"/>
  <meta name="keywords"
    content="Barton Phillips, Granby, Applitec Inc., Rotary, Programming, Tips and tricks, b
  <!-- ICONS, RSS -->
  <link rel="shortcut icon" href="http://www.bartonphillips.org/favicon.ico" />
  <link rel="alternate" type="application/rss+xml" title="RSS" href="/rssfeed.xml" />
  <!-- Custom CSS -->
  <link rel="stylesheet" href="css/blp.css" title="blp default" />
{$arg['link']}
  <!-- jQuery -->
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
  <script async src="js/tracker.js"></script>
{$arg['extra']}
{$arg['script']}
{$arg['css']}
</head>
EOF;
```

The $pageHeadText variable gets all of the <head> section code. The $arg array is created
form the argument passed to the getPageTopBottom method. The getPageTop-Bottom method also has access to
the SiteClass $this property.

You will see as you delve into the SiteClass code that many things can be passed to the
getPageTopBottom method and the various sub-methods but the standard things are:

- title
- desc
- link
- extra
- script
- css

As you saw in example 5 above (test5.php in the 'examples' directory)
I passed a $h object to the SiteClass. For example:

```
$h->title = 'my title';
$h->desc = 'This is the description';
$h->link = '<link rel="stylesheet" href="test.css">';
$h->extra = '<!-- this can be anything from a meta, link, script etc -->';
$h->script = '<script> var a="test"; </script>';
$h->css = '<style> /* some css */ #test { width: 10px; } </style>';
$S = new SiteClass($h);
```

As you can see in the 'headFile' example the $this can also be used as in $this->copyright.

The other special files have similarities and have their own file variable:

- 'bannerFile' : $pageBannerText
- 'footerFile' : $pageFooterText

As these special files are PHP files you can do anything else that you need to including database
queries.

I usually call these files 'head.i.php', 'banner.i.php' and 'footer.i.php' but you can name them anything
you like. In the .sitemap.php $siteinfo array just add the full path to the file. For example:

```
$siteinfo = array('siteDomain' => "bartonphillips.com",
                  'siteName' => "My Home Page",
                  'copyright' => "2015 Barton L. Phillips",
                  //'className' => "", // if you have sub-classed SiteClass
                  'memberTable' => "members",
                  'headFile' => SITE_INCLUDES."/head.i.php",
                  'bannerFile' => SITE_INCLUDES."/banner.i.php",
                  footerFile => SITE_INCLUDES."/footer.i.php",
                  'dbinfo' => array('database' => 'test.sdb', 'engine' => 'sqlite3'),
                  'count' => false,
                  'countMe' => true, // Count BLP
                  'myUri' => "bartonphillips.dyndns.org"
                 );
```

In the abovd example I have used the SITE_INCLUDES define from the .sitemap.php's first section.

There is a default for the <head>, banner and footer section if you do not have special files.
The DOCTYPE is by default but that can be altered via an argument to the getPageTopBottom method ($h->doctype='xxx').

Creating the special files make the tedious boiler plate simple and yet configureable via the $arg
array.

Class Methods

While there are a number of methods for each of the major classes there are really only a small
handful you will use on a regular bases. The ones must used have some documentation with them.

SiteClass methods:

- constructor
- public function setSiteCookie($cookie, $value, $expire, $path="/")
- public function setIdCookie($id, $cookie=null)
- public function checkId($mid=null, $cookie=null) // if a memberTable

- public function getId() // if a memberTable
- public function setId($id) // if a memberTable
- public function getIp()
- public function getEmail() // if a memberTable
- public function setEmail($email) // if a memberTable
- public function getWhosBeenHereToday() // if a memberTable
- public function getPageTopBottom($h, $b=null)
  This is the most used method. It takes one or two arguments which can be string|array|object.
  $h can have 'title', 'desc', 'banner' and a couple of other less used options. $b is for the footer or bottom. I usually just pass a <hr> but you can also pass a 'msg', 'msg1',
  'msg2' etc. I usually put things into the 'footerFile' but on occasions a page needs something extra.
  This method calls getPageHead(), getBanner(), getFooter().
- public function getPageTop($header, $banner=null, $bodytag=null)
- public function getDoctype()
- public function getPageHead(/*$title, $desc=null, $extra=null, $doctype, $lang*/)
- public function getBanner($mainTitle, $nonav=false, $bodytag=null)
- public function getFooter(/* mixed */)
- public function ___toString()
- A number of 'protected' methods and properties that can be used in a child class.

Database methods:

- constructor
- public function getDb()

- public function query($query)
  This is the workhourse of the database. It is used for 'select', 'update', 'insert' and basically
  anything you need to do like 'drop', 'alter' etc. $query is just sql.

- public function fetchrow($result=null, $type="both")
  Probably the second most used method. If it follows the 'query' the $result is not needed. The only
  time $result is needed is is there are other queries in a while loop. In that case you need to get
  the result of the query by calling the getResult() method before running the while loop.
  The $type can be 'assoc', 'num' or default 'both'. 'assoc' returns only an associative array, while 'num'
  return a numeric array. I usually use a numeric array with

  ```
  while(list($name, $email) = $S->fetchrow('num')) { ... }
  ```

- public function queryfetch($query, $retarray=false)

- public function getLastInsertId()
  After an 'insert' this method returns the new row primary key id.

- public function getResult()
  Returns the result object from the last 'query'. Usually not needed.

- public function escape($string)

- public function escapeDeep($value)

- public function getNumRows($result=null)

- public function prepare($query)
  Hardly ever use prepare(), bindParam(), bindResults() or execute() so they are not as well tested as
  the other methods.

- public function bindParam($format)

- public function bindResults($format)
- public function execute()
- public function getErrorInfo()

The database methods are implemented for all supported engines.

dbTables methods:

- constructor
- public function makeresultrows($query, $rowdesc, array $extra=array())
- public function maketable($query, array $extra=null)
  $extra is an optional assoc array: $extra['callback'], $extra['callback2'], $extra['footer']
  and $extra['attr'].
  $extra['attr'] is an assoc array that can have attributes for the tag, like 'id',
  'title', 'class', 'style' etc.
  $extra['callback'] function that can modify the header after it is filled in.
  $extra['footer'] a footer string
  @return array [{string table}, {result}, {num}, {hdr}, table=>{string}, result=>{result},
  num=>{num rows}, header=>{hdr}]
  or === false

Contact Me

Barton Phillips : bartonphillips@gmail.com