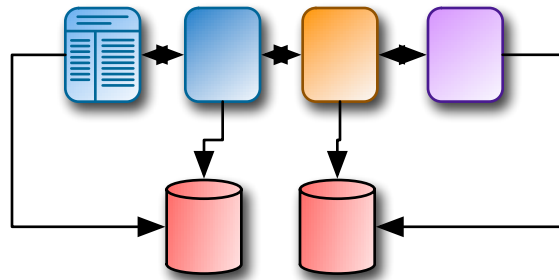


phpFramework

An Object Oriented Class For PHP To Simplify Application Development



Version 8.3 / 2 August 2008
Michael Heuss

Foreward	3
Setting Up phpFramework	5
Configuring and Accessing Functionality	7
<i>Configuration</i>	7
<i>Authentication</i>	7
<i>Caching</i>	9
<i>Database Access</i>	11
<i>File Uploads</i>	12
<i>Forms Management</i>	13
<i>Javascript</i>	14
<i>Logging</i>	15
<i>Email</i>	17
<i>Navigation</i>	18
<i>Debugging</i>	19
<i>Templating</i>	21
<i>User Input</i>	22
<i>Miscellaneous Utilities</i>	23

Foreward

phpFramework is a class that contains commonly used code organized in a way to quickly allow you to develop websites. With the functionality wrapped in this and a few other support classes, you can greatly cut down on the workload required to get a dynamic website up and running.

phpFramework does the following:

- A. Start Sessions
 - 1. Save session data into a database
 - 2. Tracks user path through the web application
 - 3. Tracks referrers
 - 4. Offers a method to enforce authentication
- B. Turns on Output Buffering
- C. Creates a benchmarking object through PEAR::Benchmark
- D. Provide a safe interface to user provided data
- E. Handles Debugging chores
 - 1. Provides a mechanism to log data through PEAR::Log
 - 2. Provides a method to dump variables through PEAR::Var_Dump
 - 3. Provides a method to debug sql through DBClass
 - 4. Allows you to set the application state to Debug or Production
 - 5. Gives you a standard place to output debugging messages. Without having it sprinkled throughout the website
- F. Starts and manages a database connection through Pear::DB and DBClass
- G. Provides an interface to the Savant templating engine
- H. Provides an interface to send rich text email
- I. Create and Validate Forms
 - 1. Can save the results of validated forms to a table or send through email
- J. Provides some commonly used javascript functionality
 - 1. Rich Text Editor

2. Select Box Navigation
 3. Pass values between parent and child windows
 4. Pop Up windows
 5. Tabbed Interface
- K. Handles the upload of images and files
- L. Provides caching of dynamic pages
- M. Tracks click on embedded links
- N. Quickly generate and display reports, complete with filtering and sorting

Setting Up phpFramework

Let's start with the necessary foundation classes. The following should be executed from the command line:

```
pear install http://phpsavant.com/Savant2-2.3.3.tgz Benchmark Log Var_Dump mail_mime  
http://mikeheuss.com/scripts/DBClass/DBClass.tgz
```

Now, we need to get the required tables going. Create the database for the site, and put in the following SQL (For MySQL - this may need to be modified for others).

```
CREATE TABLE `log_site_tracking` (`id` mediumint(9) NOT NULL default '0', `ts` timestamp(14) NOT NULL, `session_id` varchar(64) default NULL, `ip_address` varchar(25) NOT NULL default '', `path` varchar(100) NOT NULL default '', `domain` varchar(50) NOT NULL default '', `referer` varchar(100) default NULL, `user_agent` varchar(100) NOT NULL default '', PRIMARY KEY (`id`), KEY `referer` (`referer`));
```

```
CREATE TABLE log_table (id INT NOT NULL, logtime TIMESTAMP NOT NULL, ident CHAR(16) NOT NULL, priority INT NOT NULL, message VARCHAR(200), PRIMARY KEY (id));
```

```
CREATE TABLE `caching` (`path` varchar(255) NOT NULL default '', `ts` timestamp(14) NOT NULL, `num_pulled` mediumint(9) NOT NULL default '0', `id` mediumint(9) NOT NULL default '0', PRIMARY KEY (`id`), KEY `path` (`path`));
```

```
CREATE TABLE `session_data` (`id` VARCHAR( 32 ) NOT NULL , `data` INT( 10 ) NOT NULL `access` TEXT NOT NULL , PRIMARY KEY ( `id` ) );
```

Now that that is done, you need to create a configuration file that requires the phpFramework script and outlines the services you want to use. For each and every page that needs the functionality, you just have to include the configuration file, call `phpFramework::start()` at the beginning of the page and `phpFramework::stop()` at the end.

As you might have picked up from the above, phpFramework is a operates as a global, static class. It is accessible from anywhere in the application.

Here is an example configuration file that I am going to name `SystemVars.php`:

```
require_once ($framework_path."phpFramework.php");  
  
//Now we are setting the various options  
  
phpFramework::setOption("site_path", "/var/www/html/mikeheuss.com/");  
phpFramework::setOption("site_url", "http://mikeheuss.com");  
phpFramework::setOption("framework_url", "http://mikeheuss.com/phpFramework/");  
  
//Debug console dumps values to the screen. Production does not  
  
//phpFramework::setOption('state', "debug_console");  
  
phpFramework::setOption('state', "production");  
phpFramework::setOption("db_host", "localhost");  
phpFramework::setOption("db_password", "password");  
phpFramework::setOption("db_user", "mikeheuss");  
phpFramework::setOption("db_database", "mikeheuss_db");  
phpFramework::setOption("db_dbtype", "mysql");  
  
////Caching
```

```
//Whether or not to load the caching support classes
phpFramework::setOption("caching", false);

////TIMER

//Creates a timer if set to 1. Otherwise, no timer is created.
phpFramework::setOption("timer", 1);

////Logging

//Creates a log if set to one. Otherwise, no log is created
phpFramework::setOption ("log", 1);

//or

//phpFramework::setOption("log_up_to_level", "PEAR_LOG_DEBUG");

//Database

//Creates a DB connection if set to one. Otherwise, no db connection is created
phpFramework::setOption ("db", 1);

//Site Logging

//Logs path through site. If 1, it works. Otherwise, no tracking is enabled
phpFramework::setOption ("tracking",1);

//Authentication

phpFramework::setOption ("auth", 1); //Turns on authorization

//Lets set up a zone
phpFramework::authConfigureZone ("Test Zone", "~/mrheuss/mikeheuss/scripts/",
"http://mikeheuss.com/login.php");

//Templating

phpFramework::setOption ("template", 1); //Turns on templating
phpFramework::setOption ("template_path", "/var/www/html/mikeheuss.com/templates");
```

Configuring and Accessing Functionality

Configuration

Authentication

Configuring

The authentication functions work by comparing the path the visitor to the website is currently at, and seeing if visitor is attempt to access any “secure” zones. If so, phpFramework looks to see if the visitor has permission for that zone. If not, it redirects the visitor to the corresponding login page.

The authentication functions also handle the visitors session and corresponding variables.

To configure secure zones, you just need to turn authentication on, and then specify the secure zones. In the example below, we configure two such zones, one for administrators and one for users.

```
phpFramework::setOption (“auth”, 1);  
phpFramework::authConfigureZone(“Admin Zone”, “/administration”, “/admin_login.php”);  
phpFramework::authConfigureZone (“User Zone”, “/backoffice”, “/user_login.php”);
```

That is all that is done.

Functions

authStart()

This function is called automatically when you start phpFramework. This is also the function that enforces secure zones.

```
authConfigureZone ($arg_zone_name, $arg_path, $arg_login_path)
```

This function creates the zones that phpFramework will be enforcing. The zone name should be unique, the path is the absolute URL to the directory being secured (i.e. /administration), and the login path is the absolute URL to the login page. Note, the page should not be in the directory that is to be protected.

```
authCreateSession ($arg_zone, $arg_array)
```

Establishes a session for a visitor and sets the initial variables to do with that session. For instance, to create a session for the user and store the values user_id, user_type and login_time, you would do the following:

```
phpFramework::authCreateSession (“Admin Zone”, array (“user_id”=>$user_id, “user_type”=>“administrator”, “login_time”=>time()));
```

```
authEndSession ($arg_zone)
```

Logs the visitor out of a zone, completely destroying the session variables in the process.

```
authGetSessionValue ($arg_zone, $arg_variable)
```

Retrieves a single session variable for a single zone. For instance, to get the user id previously saved in the authCreateSession section above, you'd do the following:

```
phpFramework::authGetSessionValue ("Admin Zone", "user_id");
```

```
authRedirect ($arg_redirect_path)
```

This function uses javascript to redirect a visitor to a different page. The javascript is returned as a string.

```
authSetSessionValue ($arg_zone, $arg_variable, $arg_value)
```

Sets a session variable for a visitor in a single zone. For instance, to set the session variable "user_edit_flag" to true in a zone entitled "Admin Zone" you'd do the following:

```
phpFramework::authSetSessionValue ("Admin Zone", "user_edit_flag", true);
```

Caching

Configuring

Caching relies on the database option being configured.

Possible configuration values include *caching*, *cache_limit_to*, *cache_time_limit*, *cache_always_on*, *cache_exclude* and *cache_directory*. Those listed in bold are required for proper operation.

Setting *cache_always_on* to 1 means that when `phpFramework::start` is executed, we automatically check for a cached version of this page. By setting *cache_limit_to*, we are restricting caching to only pages within that directory (and subdirectories). Setting *cache_exclude* means that pages within that directory will never be cached.

The *cache_time_limit* option can be any string that can be used in the `php` `strtotime` function. When setting the *cache_directory*, it is important that the webserver have the ability to write to that directory.

There is one more configuration option for caching - *cache_calc_path*. When checking for cached files in certain configurations, you may want to determine that the structure of the URL be a certain way. You would set *cache_calc_path* for the page before `phpFramework` is started - and caching would then not look at the `HTTP_HOST` and `REQUEST_URI` when determining what page we are attempting to retrieve and cache- it would go solely on the value you specify. This is handy when you are working in third level domains and such- so that you don't create a panolpy of identical cached pages.

```
phpFramework::setOption ('caching', 1); //Turn on page caching
phpFramework::setOption ("db", 1);
phpFramework::setOption ('cache_always_on',1)
phpFramework::setOption ('cache_exclude', 'http://www.mydomain.com/backoffice/');
phpFramework::setOption ('cache_exclude', 'http://mydomain.com/backoffice/');
//phpFramework::setOption ('cache_limit_to', 'http://www.mydomain.com/static/pages');
phpFramework::setOption ('cache_time_limit', "30 minutes");
phpFramework::setOption ('cache_directory', '/var/www/html/cached_pages/');
```

Functions

`cacheOff ()`

Turns off caching for a single page.

`cachePage()`

If the framework attempted to retrieve a cache, and the page isn't in violation of the *cache_exclude* or *cache_limit_to* configuration option, then this function will execute when

phpFramework::stop is called. This function updates the database table and writes the cache file to the caching directory.

cachePurge()

This function empties the cache- both from the database table and from the file system.

cacheRetrieve()

This function attempts to retrieve a previously cached file. It also checks to ensure we aren't attempting to retrieve a file from an excluded path, and that is cache_limit_to was specified, then we are only retrieving pages from the limited url.

cacheStart()

cacheStart() is automatically called when the phpFramework is started. If cache_always_on is set, then we automatically attempt to retrieve a previously cached page.

cacheStop()

This function is automatically called by stop. If we had attempted to retrieve a cached page, then this function will automatically attempt to cache the current page.

cacheDeletePage(\$arg_id, \$arg_loaded=false)

Calling this function will delete the cached page from the database table and file system. The id contains the value of the primary key for the row from the caching table that you want deleted. If \$arg_loaded is true, then the page we are deleting is assumed to be the page that was loaded from a subsequent cacheRetrieve().

Database Access

Database Access is provided by Pear::DB and DBClass.

Configuring

```
phpFramework::setOption("db_host", "localhost");
phpFramework::setOption("db_password", "password");
phpFramework::setOption("db_user", "mikeheuss");
phpFramework::setOption("db_database", "mikeheuss_db");
phpFramework::setOption("db_dbtype", "mysql");

//Creates a DB connection if set to one. Otherwise, no db connection is created
phpFramework::setOption("db", "1");
```

Functions

dbStart()

This function is called automatically when you start phpFramework. It creates the first instance of DBClass.

dbReturnPointer(\$arg_table, \$arg_primary_key)

Returns a DBClass object, ready for manipulation. Please refer to the DBClass documentation for instructions on methods and properties. That documentation can be found on the mikeheuss.com website.

displayDataTable()

Displays the results of a query into an attractive HTML table. Each row can be assigned a different CSS class. Each row can have a link, or collection of links, that allow actions to be taken on just that row of data. This is achieved by placing the column name in between double hashes - for example - if you were wanting column id to be rendered for that link, you would put it in the string as such: ##id##.

For example - in the following command we are producing a table that has a width=100% and a cellpadding of 3. The TH row is using the tablehead2 style. Every other row uses the alternate_background style. Each row has a link to allow the data to be edited.

```
echo phpFramework::displayDataTable("<table border=0 width=100% cellpadding=3>".
    "<tr class=tablehead2>". //This is for the TH row
    "class=alternate_background". //This is for every other row
    "<a href=\"index.php?action=edit&id=##primary_id##\">Edit This</a>"
    );
```

File Uploads

Configuring

None necessary.

Functions

`fileHandleUpload ($arg_name, $arg_destination_directory)`

When given the name of the form file upload field and the destination directory you want the file to go to, this function handles the rest. It will return an error code if there is a problem - the codes are: `FILE_UPLOAD_FAILED_TO_MOVE_FILE`, `FILE_UPLOAD_NO_FILE`, `FILE_UPLOAD_NO_WRITE_PERMISSION`, `FILE_UPLOAD_NOT_DIRECTORY`.

The first two refer to problems with the uploaded file, the second two refer to problems with the destination directory.

If successful, a string is returned with the file name of the file as it is named in the destination directory.

Forms Management

JavaScript

Overview

No configuration is required. These functions return the complete javascript to be inserted into a page. If the function has already been called, then the function will return nothing.

These functions were added to the framework because I seem to use them on every single web page. The functions are as follows:

1. Javascript forward to a different page. (using javascript location)
2. Hide and Redisplay a div based on a link
3. Select box navigation
4. Emails web page URL
5. Pass back values from a child window to a parent window
6. Rich Text Editor
7. Sets the focus to the topmost form item on the page.
8. Opens a pop up window based on provided content.
9. Opens a pop up window based on provided url.
10. Creates a tabbed interface.

Functions

`jsJumpMeStart ()`

This function returns a javascript routine that will redirect a person to a different URL. If the new URL starts with `http://`, a new page is opened. Otherwise, the page opens in the same window.

After calling this function, you load the individual items by using `jsJumpMeAdd`, and when complete, load `jsJumpMeClose`.

`jsJumpMeAdd($arg_display, $arg_url, $arg_selected=false)`

Display is the text that will be in the select box, url is where it is going. If you set `arg_selected` to true, then that option will be selected.

`jsJumpMeClose`

Closes the select box.

`jsHideDiv`

returns a javascript function that hides and displays a div.

jsForward (\$arg_url)

Forwards to the page specified in arg_url. This is not a function, but script that will execute automatically when the page loads.

jsMailPage

Returns javascript that will allow a site visitor to mail a page URL to a friend.

jsPassBackValue (arg_attribute, arg_form_name, arg_value)

Returns a function that communicates back to the parent window specified values, and closes the child window.

Should be used in conjunction with the jsWindowOpen function.

jsSetFormFocus

Returns automatically execute javascript that will set the window's focus to the top most text box.

jsRichText (\$arg_content, arg_height, arg_width, arg_instance_name)

Creates a rich text editor, as specified by the function arguments.

jsTabber

Returns a javascript function that creates a tabbed interface.

jsWindowOpen

Returns a javascript function that opens a window from a given url.

jsWindowContentOpen

Returns a javascript function that will open a window based on content provided.

Logging Configuring

Logging is simple to set up. Make sure you have Database Access configured correctly, and the logging table has been created in the current database.

There are six different levels of urgency for logging. You specify which level is applicable at the time you log the event. They are, in order of urgency, FW_LOG_CRITICAL, FW_LOG_EMERGENCY, FW_LOG_ERR, FW_LOG_ALERT, FW_LOG_WARNING, FW_LOG_INFO, FW_LOG_NOTICE, FW_LOG_DEBUG

There are four different options you can set. The first option, *log_from_level*, is the level of urgency you want to start logging events. For instance, setting this to FW_LOG_ERR means that any event less than that is ignored.

The second option, *log_email_level*, is the level at which emails start being sent out to alert you of a problem in addition to the regular logging into the database.

To use the email options, you need to specify *log_email*, which is the address you want email to go to, and *log_email_from*, which is the return address you want the emails to have once sent out.

So, a sample configuration looks like this:

```
phpFramework::setOption ("log_from_level", FW_LOG_DEBUG);  
phpFramework::setOption ("log_email_level", FW_LOG_EMERGENCY);  
phpFramework::setOption ("log_email", "mike@rapidstability.com");  
phpFramework::setOption ("log_email_from", "server@eztradepro.com");
```

Functions

`logEvent ($arg_level, $arg_function, $arg_message="", $arg_data="")`

This function logs to the table, and if configured and appropriate, sends out emails using the standard php mail command.

\$arg_level is the level of this event. The possible levels are FW_LOG_CRITICAL, FW_LOG_EMERGENCY, FW_LOG_ERR, FW_LOG_ALERT, FW_LOG_WARNING, FW_LOG_INFO, FW_LOG_NOTICE, FW_LOG_DEBUG

\$arg_function is either the function, name or subject of the event you are logging.

\$arg_message is any message you want to accompany the event

\$arg_data is any data you want to accompany the event.

Email

Configuring

Email support is based on PEAR Mail & Mail_mime.

All that needs to be configured is mail needs to be enabled. This is done as follows:

```
phpFramework::setOption ("mail", 1);
```

However - you can also configure phpFramework to send mail using SMTP - to do that, you'll need to specify a few extra options:

```
phpFramework::setOption ("smtp_host", "mail.mikeheuss.com");  
phpFramework::setOption ("smtp_port", "25");  
phpFramework::setOption ("smtp_auth", true);  
phpFramework::setOption ("smtp_username", "mike@mikeheuss.com");  
phpFramework::setOption ('smtp_password', "somepassword");
```

To use SMTP with SSL, just change as follows:

```
phpFramework::setOption ("smtp_host", "ssl://mail.mikeheuss.com");  
phpFramework::setOption ("smtp_port", "465");
```

Additionally, you can just call `phpFramework::mailStart()` in your application

Functions

```
mailSimpleSend ($arg_recipient, $arg_headers, $arg_text, $arg_body='')
```

Attempts to send a mail message. `arg_recipient` is the email address to send the message to. Headers can be an array of any common mail header, but should contain at least `array('subject'=>'some subject')`. `arg_text` is the plain text version of your email, `$arg_body` is the rich version. If the `arg_body` is omitted, then we send a plain text message.

This function returns true if the message is sent, otherwise a `PEAR::isError` is sent. Here is an example on how to test if the message was sent successfully:

```
$mail = phpFramework::mailSimpleSend ("test@test.com",  
    array("FROM"=>'mike@mikeheuss.com', "SUBJECT"=>"This is a test"),  
    "This is a message body");  
  
if (PEAR::isError($mail)) {  
    echo("<p>" . $mail->getMessage() . "</p>");  
} else {  
    echo("<p>Message successfully sent!</p>");  
}
```

Navigation

Configuring

None required.

Functions

One navigation function was described in the javascript section - the jsJumpMe script. Another is defined in jsTabber.

navigateBreadCrumbs (\$arg_array, \$arg_seperator, \$arg_link_class)

Bread crumbs refers to the navigation system you see at the top of many search engines and deeply nested pages. Since I can't seem to think of the words to describe one, let an example suffice: If you were on a site within the United States section, reading a page dedicated to Alabama, the bread crumb at the top of the page might read Home >> United States >> Alabama, with each segment being a hyper link to each level's home page.

Arg_array contains the path information. The key of the array is the text to be displayed, the value is the URL that the person goes to if they click on the link. The separator is, in the above example, >> - that can be changed by specifying a different separator in arg_seperator. The link class is the class each link should belong to.

Debugging

Configuring

Most debugging functions need the option *state* to be set to `debug_console`, instead of `production`, to have any effect.

To use the Timer functions, *timer* needs to be set to 1.

The configuration code would look like this:

```
phpFramework::setOption ("timer", 1);  
phpFramework::setOption ("state", "debug_console");
```

Functions

getStdErr()

`stdErr` is a concept familiar to other programming languages - it is a particular port where you can send error messages, and is kept separate from the regular output.

The concept is carried through in `phpFramework` - by sending your data to `stdErr`, you can retrieve that data later as a string and display it separately.

By default, when you execute `stop()`, the framework will output a hidden collapsible div that is exposed by clicking on the `stdErr` link at the bottom of your page. That link will not be present if there is no data to output;

```
outputVariable(&$arg_mixed, $arg_forced=false)
```

This function will print the contents of the variable passed into *arg_mixed* using the `var_dump` PEAR class. If `forced` is set to true, this function will operate even if we are in production mode.

```
stdErr(&$arg_output)
```

This will add a newline and the contents of `output` to the `stdErr` buffer.

```
stdErrHTML(&$arg_output)
```

This will add a `
` and the contents of `output` to the `stdErr` buffer.

```
timerElapsed ($arg_start_mark, $arg_end_mark)
```

This function will return the time elapsed between two marks. For more on timers, see the Benchmark package in PEAR at <http://pear.php.net/package/Benchmark>

```
timerSetMarker ($arg_name)
```

This function allows a marker called *arg_name* to be set.

`timerStart ()`

Called automatically by `start()`, this function creates the first marker.

`timerStop ()`

Called automatically by `stop()` this function stops benchmarking and displays the results on the screen using a hidden, collapsible div and a link called `timer`.

Templating

I never liked Smarty or most other templating systems - and in fact tended to use a home brewed system for years - until I discovered Savant. For more on Savant and why it is better than most of the other systems out there, check out <http://phpsavant.com/yawiki/>. I am using Savant 2.0 - as most of the servers I write for are not PHP 5 based.

Configuring

To use templating, the *template* option must be set to 1 or true, and *template_path* must point to the directory where your templates are stored.

Additionally, before you call `phpFramework::start()`, you should set the template to be used by the current path by calling `templateChoose($arg_template_name)`.

```
phpFramework::setOption ("template", true);  
phpFramework::setOption ("template_path", "/Users/Sites/mike/mywebsite/templates/");
```

Functions

`templateAssign($arg_tag, $arg_value)`

Assigns content to the template. For instance, if your template has a stub called "Main-Body", you'd assign values to that stub as follows:

```
phpFramework::setOption ("MainBody", "<p>This is the main body of my website");  
templateChoose($arg_template_name)
```

Choose the template to merge with the assigned data.

`templateDisplay($arg_template_name="")`

`templateDisplay` renders the template and sends it to the browser. If the template name has not been previously specified, or if the template needs to be override, it can be set by passing it in now.

If `phpFramework::stop()` is executed before `template::Display` is executed, and the rendered template is sent to the browser.

User Input

Configuring

None required.

Functions

```
userGet($arg_value, $arg_default='')
```

Returns the `$_GET` array for the value specified by key `$arg_value`. If it doesn't exist, the function returns the value specified in `$arg_default`. If it does exist, and magic quotes is turned on, strip slashes is executed before the value is returned.

```
userPost ($arg_value, $arg_default='')
```

Returns the `$_POST` array for the value specified by key `$arg_value`. If it doesn't exist, the function returns the value specified in `$arg_default`. If it does exist, and magic quotes is turned on, strip slashes is executed before the value is returned.

```
userGetPost ($arg_value, $arg_default='')
```

Checks the `$_GET` array for the value specified by key `$arg_value`. If it doesn't exist, we check the `$_POST` array. If that doesn't exist, we return `$default`.

If the value is present, we check to see if magic quotes are turned on. If so, we execute strip slashes before returning the value.

```
userPostGet ($arg_value, $arg_default='')
```

Checks the `$_POST` array for the value specified by key `$arg_value`. If it doesn't exist, we check the `$_GET` array. If that doesn't exist, we return `$default`.

If the value is present, we check to see if magic quotes are turned on. If so, we execute strip slashes before returning the value.

Miscellaneous Utilities

Functions

urlGetThirdLevel ()

This function returns the third level domain, if it has one. For instance, if the domain is update.microsoft.com, this function would return update. If the domain only has two levels, it returns a URL_SECOND_LEVEL_ONLY. If the third level is www, it returns URL_WWW_DOMAIN.

dateEdit (\$arg_ts, \$arg_name, \$arg_type, \$arg_year_start=0, \$arg_year_end=0, \$arg_styles=0)

Creates a drop down box loaded and ready to go for editing dates. *arg_ts* is the current timestamp you want the drop down to be set to, *arg_name* is the base name you want the select boxes to use, *arg_type* is the type of date box you want, either date, or time, or datetime. *arg_year_start* and *arg_year_end* are mean what they say - they specify your range of years you want to display.

arg_styles, which can either be an array or a string, specifies the classes you want to use. If an array, the two keys you are looking for are date or time.

This function should be used in conjunction with dateGetPost.

dateGetPost (\$arg_name, arg_type)

This function is complimentary with dateEdit. By passing in the name and type you used in creating the dateEdit boxes, it will return which date and/or time the user provided in a submitted form as a timestamp.

dateCheck (\$arg_text)

Validates a user supplied a valid date in mm/dd/yyyy format.

displayDataTable (\$arg_array, \$arg_db_connection, \$arg_table_string, \$arg_th_string, \$arg_toggle_string, \$arg_link_html, \$arg_td_formatting=null)

Takes the data returned from a sql query and displays it in a HTML table.

prettyNumberFormat (\$arg_input, \$arg_integer=false)

This function will format decimals or integers in a user friendly manner. Thousands are separated by commas, decimals are rounded to the nearest hundredth.

stringLeft (\$arg_string, \$arg_length)

Returns *arg_length* number of characters from the left of \$arg_string.

stringRight (\$arg_string, \$arg_length)

Returns *arg_length* number of characters from the right of \$arg_string.

urlForwarding (\$arg_id)

This function allows you to forward visitors to other pages, and track who you sent where. This option must be configured, options *db* must be set to 1 and *url_forwarding* must be initialized with the following array: Key *table* must be initialized with the name of the database table we are tracking with, *primary_key* should contain the primary key of that table, *url* is the column that contains the destination url, *counter* is the column that contains the number of times you have sent someone there.

arg_id is the primary key of the row of the table that contains the link the person is going to be sent to.