PHPLucidFrame Documentation

Release 1.18.0

Sithu K.

Aug 05, 2018

Table of Contents:

CHAPTER 1

About PHPLucidFrame

PHPLucidFrame (a.k.a LucidFrame) is a mini application development framework - a toolkit for PHP developers. It provides logical structure and several helper utilities for web application development. It uses a functional architecture to simplify complex application development.

1.1 Prerequisites

- Web Server (Apache with mod_rewrite enabled)
- PHP version 5.3.0 or newer (optional mcrypt extension enabled, but recommended)
- MySQL 5.0 or newer with MySQLi enabled.

1.2 License

PHPLucidFrame is licensed under the MIT license. This means that you are free to modify, distribute and republish the source code on the condition that the copyright notices are left intact. You are also free to incorporate PHPLucidFrame into any Commercial or closed source application.

CHAPTER 2

Installation

You can get a fresh copy of PHPLucidFrame on the official website. You can also fork and clone the GitHub repository using git. Alternatively, you can install it using Composer with the command:

```
composer create-project --prefer-dist phplucidframe/phplucidframe [your-project-name]
```

Regardless of how you downloaded it, place the code inside of your DocumentRoot. In **production**, your directory setup may look something like the following so that it can be accessible via `http://www.example.com`

```
/path_to_webserver_document_root
    /app
    /assets
    /business
    /db
    /files
    /i18n
    /inc
    /lib
    /tests
    /vendor
    .htaccess
    index.php
```

In this case, the configuration variable `baseURL` in `/inc/parameter/production.php` should look like this:

```
return array(
    # No trailing slash (only if it is located in a sub-directory of the document root)
    # Leave blank if it is located in the document root
    'baseURL' => '',
    // ....
);
```

In **development**, you could have a directory name, for example, `phplucidframe`. Then your directory setup may look something like the following so that it can be accessible via `http://localhost/phplucidframe`.

```
/path_to_webserver_document_root
    /phplucidframe
    /app
    /assets
    /business
    /db
    /files
    /i18n
    /inc
    /lib
    /tests
    /vendor
    .htaccess
    index.php
```

In this case, the configuration variable `baseURL` in `/inc/parameter/development.php` should look like this:

```
return array(
    # No trailing slash (only if it is located in a sub-directory of the document root)
    # Leave blank if it is located in the document root
    'baseURL' => 'phplucidframe',
    // ....
);
```

Note: `phplucidframe` would be your project name.

CHAPTER 3

Application Structure

Directory	Description
app	This directory structure contains the application files and folders of your site. The directory is auto-bootstrapped with PHPLucidFrame environment.
app/helpers	The helpers mapping to the system core helpers should be placed in this directory directory. They are auto-loaded. For example, the custom validation helper (`validation_helper.php`) should be placed in this directory and it is auto-loaded across the site. The following helper files are allowed: • `auth_helper.php` • `db_helper.php` • `pager_helper.php` • `session_helper.php` • `utility_helper.php` • `validation_helper.php`
app/cmd	The console command implementation should be placed in this directory. They are auto-loaded. For example, if you implement a custom command file `GreetCommand.php` it should be placed in this directory and it is auto-loaded across the site.
app/entity	This directory should be used to place the files which contains the business log functions or classes. They usually do the direct operations to the database layer.
app/inc	The directory can include the site template files and site configuration file. • `/site.config.php` (inherited by `/inc/site.config.php`) • `/tpl/head.php` (overridable by `/inc/tpl/head.php`) • `/tpl/401.php` (overridable by `/inc/tpl/401.php`)
	• `/tpl/403.php` (overridable by `/inc/tpl/403.php`) • `/tpl/404.php` (overridable by `/inc/tpl/404.php`) • `/tpl/header.php` (overridable by `/inc/`

3.1 Page Structure

PHPLucidFrame encourages a uniform and structural page organization. In brief, a page in LucidFrame is represented by a folder containing at least two files: `index.php` and `view.php`. As an example, you can see the directory `/app/home/` of the LucidFrame release you downloaded.

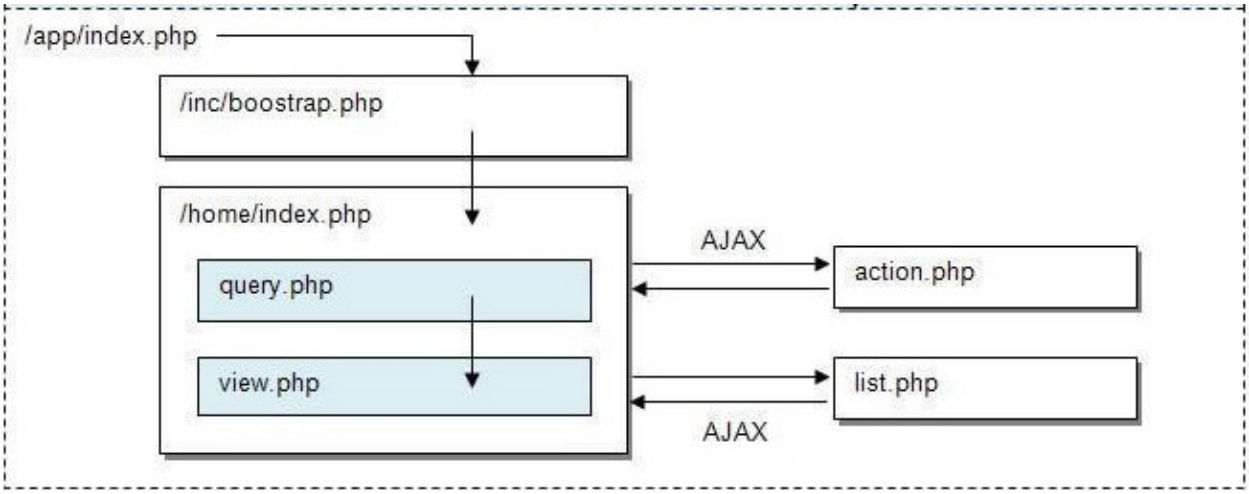
```
/path_to_webserver_document_root
    /phplucidframe
        /app
            /home
                |-- action.php
                |-- index.php
                |-- query.php
                |-- view.php
```

1. The **index.php** (required) serves as the front controller for the requested page, initializing the base resources needed to run the page.
2. The **action.php** (optional) handles form submission. It should perform form validation, create, update, delete of data manipulation to database. By default, a form is initiated for AJAX and `action.php` is automatically invoked if the action attribute is not given in the `<form>` tag.
3. The **query.php** (optional) should retrieve and process data from database and make it available to view.php.
4. The **view.php** (required) is a visual output representation to user using data provided by query.php. It generally should contain HTML between `<body>` and `</body>`.
5. The **list.php** (optional) is a server page requested by AJAX, which retrieves data and renders HTML to the client. It is normally implemented for listing with pagination.

PHPLucidFrame is not bound to any specific directory structure, these are simply a baseline for you to work from.

3.2 Page Workflow

This illustration demonstrates a request to `http://www.example.com` or `http://localhost/phplucidframe`.



3.3 Layout Mode

By default, PHPLucidFrame has two template files - `header.php` and `footer.php`. They will have to include in every `view.php`. Some developers may not want to have header and footer templates separately and not want to include the files in all views. They usually create a site layout file.

Since version 1.14, PHPLucidFrame provides a new feature to enable/disable layout mode globally or for a particular page.

3.3.1 Create a Layout File

Create your layout file in `/inc/tpl/` or `/app/inc/tpl/`. Default layout file name is `layout.php`. `<?php include _view(); ?>` has to be called in the layout file. Here is an example layout file content:

```
<!DOCTYPE html>
<html>
<head>
    <title><?php echo _title(); ?></title>
    <?php include _i('inc/tpl/head.php'); ?>
</head>
<body>
    <div id="wrapper">
        <div id="page-container">
            <div id="header">
                <div class="container clearfix">
                    <!-- header content -->
                </div>
            </div>
            <div id="page">
                <div class="container">
                    <?php include _view(); ?> <!-- page view -->
                </div> <!-- .container -->
            </div> <!-- #page -->
            <div id="footer">
                <div class="container">
                    <!-- footer content -->
                 </div>
            </div>
        </div> <!-- #page-container -->
    </div> <!-- #wrapper -->
</body>
</html>
```

3.3.2 Enable Layout Mode globally

To enable layout mode globally, set true to `$lc_layoutMode` in `/inc/config.php`.

```
# $lc_layoutMode: Enable layout mode or not
$lc_layoutMode = true;
# $lc_layoutMode: Default layout file name
$lc_layoutName = 'layout'; // layout.php
```

You can also configure `$lc_layoutName` using a custom file name other than `layout.php`. Now that you have enabled the layout mode globally, `query.php` and `view.php` are automatically included for every page.

```
/app
    /home
    |-- action.php
    |-- index.php
    |-- query.php (this file will be automatically included when layout mode is␣
↪enabled)
    |-- view.php (this file will be automatically included when layout mode is␣
↪enabled)
```

3.3.3 Enable Layout Mode for a Page

Assuming that you have `$lc_layoutMode = false` that makes layout mode disabled globally. If you want to enable it for a particular page. You can call `_cfg('layoutMode', true);` at the top of `index.php` of the page folder.

In addition, you can create a new layout for a particular page or a group of pages. You just need to call `_cfg('layoutName', 'another-layout-file-name');` for the pages. Check the example at `/app/example/layout/index.php`.

3.4 Savant Integration

Note: PHPLucidFrame does not tie to any template system.

Savant is a powerful but lightweight object-oriented template system for PHP. Unlike other template systems, Savant by default does not compile your templates into PHP; instead, it uses PHP itself as its template language so you don't need to learn a new markup system. You can easily integrate it into LucidFame.

Check the integration guide in the PHPLucidFrame wiki.

CHAPTER 4

Bootstrapping

The `/app` directory is auto-bootstrapped with PHPLucidFrame environment. If you have a new directory out of the app directory, all script files in that directory are not auto-bootstrapped. However you can manually bootstrap them. Let's say you have a directory `/scripts` at the same level of the `/app` directory and you have a script file `/scripts/example.php`, the file should have the following content.

```
<?php
chdir('../'); // change directory to the root directory from the current directory
require_once('bootstrap.php');

// Do something here ...
```

Then the script has been bootstrapped in PHPLucidFrame environment.

CHAPTER 5

Core Defined Constants & Variables

The following PHP constants are available across PHPLucidFrame.

Constant	Description
`_DS_`	Convenience use of `DIRECTORY_SEPARATOR`
`APP_ROOT`	File system path to the application's directory
`WEB_ROOT`	URL to the application root, e.g., `http://www.example.com` or `http://localhost/phplucidframe/`
`ROOT`	File system path to the directory `root`
`INC`	File system path to the directory `inc` in the root directory
`DB`	File system path to the directory `db` in the root directory
`LIB`	File system path to the directory `lib` in the root directory
`HELPER`	File system path to the directory `lib/helpers` in the root directory
`CLASSES`	File system path to the directory `lib/classes` in the root directory
`I18N`	File system path to the directory `i18n` in the root directory
`VENDOR`	File system path to the directory `vendors` in the root directory
`FILE`	File system path to the directory `files` in the root directory
`CACHE`	File system path to the directory `cache` in the root directory
`ASSETS`	File system path to the directory `assets` in the root directory
`CSS`	File system path to the directory `assets/css` in the root directory
`IMAGE`	File system path to the directory `assets/images` in the root directory
`JS`	File system path to the directory `assets/js` in the root directory
`TEST_DIR`	File system path to the directory `tests` in the root directory
`LC_NAMESPACE`	Namespace according to the site directories, for example, if you have `www.example.com/admin`, you may have a namespace `admin`.
`WEB_VENDOR`	Web-accessible path to the `vendors` directory., e.g., `http://www.example.com/vendors/`
`HOME`	The home page URL

PHPLucidFrame has a global object in Javascript – `LC`. The following Javascript global variables of `LC` are available to use.

Variable	Description
LC.root	URL to the application root. It could also be accessible as WEB_ROOT in PHP.
LC.self	The current URL
LC.lang	The current language code
LC.baseURL	The sub-directory name if your application is wrapped in. It would be blank if your application is located in the web server document root.
LC.route	The current route path
LC.cleanRoute	The current route path without query string and file name
LC.namespace	Namespace according to the site directories, for example, if you have www.example.com/admin, you may have a namespace admin.
LC.Page.root	The absolute path to the site root including the language code

You can also extend any global variable from LC by using a hook __script() in /app/helpers/utility_helper.php.

```
<?php
/**
 * This function is a hook to the core utility function _script()
 */
function __script(){
?>
    LC.cleanURL = <?php echo (int) _cfg('cleanURL'); ?>;
<?php
}
?>
```

CHAPTER 6

URL Routing

PHPLucidFrame typically requires `mod_rewrite` enabled on web server. As per the default configuration in `/inc/route.config.php`, the app home page is by default set to `/home` which is mapped to the directory `/app/home/` and it is accessible via `http://localhost/phplucidframe` or `http://www.example.com`. You can change it according to your need.

```
// inc/route.config.php
/**
 * The named route example `lc_home`
 * This will be overridden to `$lc_homeRouting` in /inc/config.php
 * If this is not defined here, `$lc_homeRouting` will be used
 * However, `$lc_homeRouting` is deprecated in 1.10 and it will be removed in 2.0
 * This is a recommended place to define routings if necessary
 */
route('lc_home')->map('/', '/home');
```

PHPLucidFrame is already designed for friendly URL without any custom route defined. Let's say for example, you have a page `/app/post/index.php` and URL `http://www.example.com/post/1/edit`. The URL will be mapped to the file `/app/post/index.php` by passing 2 arguments - `1` and `edit`.

```
// http://www.example.com/post/1/edit => /app/post/index.php
echo _arg(1); // 1
echo _arg(2); // edit
echo _url('post', array(1, 'edit')); // http://www.example.com/post/1/edit
```

6.1 Custom Routes

If you are not enough with PHPLucidFrame basic routing and if you need your own custom routes, you can easily define them in `/inc/route.config.php`. The following example shows the route key `lc_blog_show` of the route path `/blog/{id}/{slug}` mapping to `/app/blog/show/index.php` by passing two arguments `id` and `slug` with the requirements of `id` to be digits and `slug` to be alphabets/dashes/underscores.

```
// inc/route.config.php
/**
 * The named route example `lc_blog_show`
 * This is an example routed to the directory `/app/blog/show`
 */
route('lc_blog_show')->map('/blog/{id}/{slug}', '/blog/show', 'GET', array(
    'id'    => '\d+', # {id} must be digits
    'slug'  => '[a-zA-Z\-_]+' # {slug} must only contain alphabets, dashes and␣
↪underscores
));
```

Then you can get the argument values from `_arg('id')` and `_arg('slug')` in `/app/blog/show/index.php`.

```
// app/blog/show/index.php
$id   = _arg('id');
$slug = _arg('slug');
```

Here is the custom routing configuration syntax:

```
route($name)->map($path, $to, $method = 'GET', $patterns = null)
```

Argument Name	Type	Description
`$name`	string	Any unique route name to the mapped $path
`$path`	string	URL path with optional dynamic variables such as `/post/{id}/edit`
`$to`	string	`GET`, `POST`, `PUT` or `DELETE` or any combination with `|` such as `GET|POST`. Default to `GET`.
`$pattern`	array|null	Array of the regex patterns for variables in $path such as `array('id' => '\d+')`

6.2 Accessing URL

You can get the current routing path using a function `_r()` and you can get a component of the current path using `_arg()`. x

// url is www.example.com echo _r(); // home echo _arg(0); // home

// url is www.example.com/user/1 echo _r(); // user/1 echo _arg(0); // user echo _arg(1); // 1

PHPLucidFrame also provides to use URL component key preceding by a dash `-`. For example, `http://www.example.com/posts/-page/1` which reflects `http://www.example.com/posts?page=1`

```
// url is www.example.com/posts/-page/1/-sort/title/asc
echo _r();    // posts/-page/1/-sort/title
echo _arg(0); // posts
echo _arg(1); // -page
echo _arg(2); // 1
echo _arg(3); // -sort
echo _arg(4); // title
echo _arg(5); // asc

// The following is a formal way of getting the URI component "page"
echo _arg('page'); // 1
```

(continues on next page)

(continued from previous page)

```
// The following is a formal way of getting the URI component "sort"
_pr(_arg('sort')); // array( 'title', 'asc' )
// _pr() is an convenience method for print_r.
```

6.3 Creating and Getting URL

You can use the function `_url()` or `route_url()` to make an absolute URL.

```
echo _url('user', array(1));
// http://www.example.com/user/1

echo _url('posts', array('page' => 1, 'sort' => array('title','asc')));
// http://www.example.com/posts/-page/1/-sort/title/asc

echo _url(); // same as echo _self();
// it would return the current URL
```

6.4 Redirecting URL

You can use the function `_redirect()` to redirect to a URL.

```
// redirect to the home page according to $lc_homeRouting in /inc/config.php
// 'home' is a constant whatever you defined for $lc_homeRouting
_redirect('home');

// redirect to http://www.example.com/user/1
_redirect('user', array(1));

// redirect to http://www.example.com/posts/-page/1/-sort/title/asc
_redirect('posts', array('page' => 1, 'sort' => array('title','asc')));

// assuming that the current URL is http://www.example.com/posts/-page/1/-sort/title/
↪asc
// you can redirect to the current page itself by updating the query strings 'page'␣
↪and 'sort'
// in this case, you can use NULL or an empty string for the first parameter to _
↪redirect()
// redirect to http://www.example.com/posts/-page/2/-sort/title/desc
_redirect(NULL, array('page' => 2, 'sort' => array('title','desc')));

// redirect to the current page itself
_redirect(); // or _redirect('self');

// permanent redirect to the new page
_redirect301('path/to/a/new/replaced/page');

// redirect to 401 page
_page401(); // or _redirect('401')

// redirect to 403 page
_page403(); // or _redirect('403')
```

(continues on next page)

(continued from previous page)

```
// redirect to 404 page
_page404(); // or _redirect('404')
```

Check more details in `/lib/helpers/utility_helper.php` and `/lib/helpers/route_helper.php`.

6.5 Custom URL Rewrite

Note: This needs knowledge of Apache `.htaccess` rewrite rule syntax.

You may also write RewriteRule in `.htaccess` of the root directory, but by no means required.

```
# www.example.com/en/99/foo-bar to ~/app/post/?lang=en&id=99&slug=foo-bar
# www.example.com/zh-CN/99/foo-bar to ~/app/post/?lang=zh-CN&id=99&slug=foo-bar
RewriteRule ^(([a-z]{2}|[a-z]{2}-[A-Z]{2})/)?([0-9]+)/(.*)$ app/index.php?lang=$1&id=
↪$3&slug=$4&route=post [NC,L]
```

As the default routing name of LucidFrame is **route** and according to the RewriteRule above, `route=post` will map to the file `/app/post/index.php` or `/app/post.php` given the three URI components – `lang`, `id` and `slug`. For example, if the requested URL is `www.example.com/en/99/foo-bar`, this will be rewritten to `/app/post/index.php?lang=en&id=99&slug=foo-bar` or `/app/post.php?lang=en&id=99&slug=foo-bar`. In this case you can get the **id** and **slug** using `_arg()`:

```
$id = _arg('id');
$slug = _arg('slug');
```

CHAPTER 7

File Inclusion

PHPLucidFrame helps you to include files more easier. You can use `_i()` for PHP files, `_js()` for Javascript files and `_css()` for CSS files. The `_i()` is returning the system file path and it has to be used with the PHP built-in functions include and require. The `_js()` and `_css()` will look for the files in the directory `/assets/css/` and `/assets/js/` and include them automatically.

All of three functions will operate based on the configuration variable `$lc_sites` in `/inc/config.php`. They will look for the files from the most specific directory to the least. For example, if you use `include(_i('inc/tpl/head.php'))`, it will look for the files as follow and it will stop where the file is found.

1. `/app/inc/tpl/head.php`
2. `/inc/tpl/head.php`

Another example is that if you have a directory `/app/admin/` configured in `$lc_sites` as follow:

```
# $lc_sites: consider sub-directories as additional site roots and namespaces
/**
 * ### Syntax
 *  array(
 *   'virtual_folder_name (namespace)'  => 'physical_folder_name_directly_under_app_
↪directory'
 *  )
 *  For example, if you have the configuration `array('admin' => 'admin')` here, you␣
↪let LucidFrame know to include the files
 * from those directories below without specifying the directory name explicitly in␣
↪every include:
 *   /app/admin/assets/css
 *   /app/admin/assets/js
 *   /app/admin/inc
 *   /app/admin/helpers
 * you could also set 'lc-admin' => 'admin', then you can access http://localhost/
↪phplucidframe/lc-admin
 * Leave this an empty array if you don't want this feature
 * @see https://github.com/phplucidframe/phplucidframe/wiki/Configuration-for-The-
↪Sample-Administration
```

(continues on next page)

(continued from previous page)

```
-Module
 */
$lc_sites = array(
    'admin' => 'admin',
);
```

then, PHPLucidFrame will look for the file:

1. `/app/admin/inc/tpl/head.php`
2. `/app/inc/tpl/head.php`
3. `/inc/tpl/head.php`

For `js()` and `_css()`, you don't need to include the directory path as it looks for the files in the `/assets/js/` and `/assets/css/` folders and prints out `<script>` and `<link />` respectively if they found the files. There are two system provided directories - `/assets/js/` and `/assets/css/` under the root. Let's say you also have those two directories in other sub-directories as below:

```
/path_to_webserver_document_root
   /app
    |-- /admin
    |    |-- /assets
    |    |    |-- /css
    |    |    |-- /js
    |-- /assets
    |    |-- /css
    |    |-- /js
   /assets
    |-- /css
    |-- /js
```

When you use `_js('site.js')` and if you are at admin, it will look for the file as the following priority and it will stop where the file is found.

1. `/app/admin/assets/js/site.js`
2. `/app/assets/js/site.js`
3. `/assets/js/site.js`

It is same processing for the usage of `_css('base.css')`:

1. `/app/admin/assets/css/base.css`
2. `/app/assets/css/base.css`
3. `/assets/css/base.css`

CHAPTER 8

Database Configuration and Data Manipulation

You can configure your database settings in three files according to your deployment environments:

- `/inc/parameter/development.php` for development environment
- `/inc/parameter/production.php` for production environment
- `/inc/parameter/test.php` for test environment

```
return array(
    // ...
    # Database connection information
    'db' => array(
        'default' => array(
            'engine'    => 'mysql', // database engine
            'host'      => 'localhost', // database host
            'port'      => '', // database port
            'database'  => 'lucid_blog', // database name
            'username'  => 'root', // database username
            'password'  => '', // database password
            'prefix'    => '', // table name prefix
            'collation' => 'utf8_unicode_ci' // database collation
        )
    )
    // ...
);
```

If you enable `$lc_useDBAutoFields` in `/inc/config.php`, each of tables in your database should have the following four fields:

- `slug varchar(xxx)`
- `created datetime`
- `updated datetime`
- `deleted datetime`

Nonetheless, you don't need to worry about them if you write your own custom queries. They are also flaggable when you use more handy functions like `db_insert()` or `db_update()`.

Note: As of version 1.14, Schema Manager will manage those slug and timestamp fields for you. Just ignore `$lc_useDBAutoFields`.

8.1 Retrieving Your Data

PHPLucidFrame provides several db helper functions to retrieve data from your database. You can use the following functions to retrieve your data:

- db_query() which reflects to mysqli_query().
- db_fetchAssoc() which reflects to mysqli_fetch_assoc().
- db_fetchArray() which reflects to mysqli_fetch_array().
- db_fetchObject() which reflects to mysqli_fetch_object().
- db_fetch() which retrieves the only first field data
- db_fetchResult() which retrieves the one result in object.
- db_extract() which retrieves the result in array or array of objects.

Note: Instead of using native query to fetch data, QueryBuilder is recommended. See *Query Builder* usage.

The following is an example to retrieve multiple result rows:

```
$sql = 'SELECT * FROM '.db_table('post').' ORDER BY postName';
if ($result = db_query($sql)){
    while($row = db_fetchAssoc($result)){
        // do somethings here...
    }
}

// Extract all data into an array of objects
// db_extract() invokes db_fetchObject() by default internally
$sql = 'SELECT * FROM '.db_table('post').' ORDER BY postName';
$posts = db_extract($sql); // The second or third argument can be given one of these:␣
↪LC_FETCH_OBJECT (default), LC_FETCH_ASSOC, LC_FETCH_ARRAY
_pr($posts);

// Extract all data into key/value pair of array
$sql = 'SELECT postId key, postTitle value FROM '.db_table('post).' ORDER BY postName
↪';
$posts = db_extract($sql);
_pr($posts);
/*
array(
  $postId => $postTitle
)
*/
```

The following is an example to retrieve a single result:

```
// Retrieving a single-row result
$sql = 'SELECT * FROM '.db_table('post').' WHERE postId = :id';
if ($post = db_fetchResult($sql, array(':id'=>$id))) {
    _pr($post);
    // $post->postId;
    // $post->postTitle;
}

// Retrieving the result count
$sql = 'SELECT COUNT(*) FROM '.db_table('post');
$count = db_count($sql);

// Retrieving a field
$sql = 'SELECT MAX(postId) FROM '.db_table('post');
$max = db_fetch($sql);
```

8.2 Inserting Your Data

`db_insert()` will save you when you are trying to insert your data into the database without writing `INSERT` statement. The syntax is `db_insert('table_name', $data=array(), $useSlug=true)`. For example,

```
$success = db_insert('post', array(
    'postTitle' => 'New Title', // this will be used for the slug field while third␣
↪argument is true
    'postBody' => 'Post complete description here',
));

if ($success) {
    // do something with db_insertId() or db_insertSlug()
}
```

You can also provide a custom slug in the `$data` array.

```
$slug = 'your-custom-slug-string';
$success = db_insert('post', array(
    'slug' => $slug,
    'postTitle' => 'Updated Title',
    'postBody' => 'Updated post complete description here'
));
```

- db_insertId() which reflects to mysqli_insert_id().
- db_insertSlug() returns the generated slug used in the last query.

Note:

- The first field in data array will be used to insert into the slug field.
- Table prefix to the table name of the first parameter is optional.

8.3 Updating Your Data

`db_update()` is a convenience method for your SQL `UPDATE` operation. The syntax is `db_update('table_name', $data=array(), $useSlug=true, $condition=NULL)`. For example,

```
$success = db_update('post', array(
    'postId'    => 1, // this first data value must be the record ID to be updated
    'postTitle' => 'Updated Title', // this will be used for the slug field while
↪third parameter is true
    'postBody'  => 'Updated post complete description here'
));
// UPDATE post SET
//   slug = "updated-title",
//   postTitle = "Updated Title",
//   postBody = "Updated post complete description here
//   updated = "....."
// WHERE postId = 1

if($success){
    // do something
}
```

You can also provide a custom slug in the *$data'* array.

```
$slug = 'your-custom-slug-string';
$success = db_update('post', array(
    'postId'    => $updateID, // this first data value must be the record id to be
↪updated
    'slug'      => $slug, // providing custom slug string
    'postTitle' => 'Updated Title',
    'postBody'  => 'Updated post complete description here'
));
```

You can provide the third or fourth parameter `$condition`. See *Query Conditions*.

```
$condition = array(
    'fieldName1'    => 'value1',
    'fieldName2 !=' => 'value2',
    'fieldName2 >'  => 'value3',
);
```

8.4 Deleting Your Data

`db_delete()` is a handy method for your SQL `DELETE` operation. This is only applicable for single record deletion. The syntax is `db_delete('table_name', $condition=null)`. LucidFrame encourages MYSQL Foreign Key Constraints to use. If `ON DELETE RESTRICT` is found, it performs soft delete (logical delete) by updating the current date/time into the field `deleted`, otherwise it performs hard delete (physical delete).

```
if (db_delete('post', array('postId' => $idToDelete))) {
    $success = true;
}
```

`db_delete_multi()` is useful for batch record deletion for the given condition, but it does not check foreign key constraints.

```
db_delete_multi('table_name', $condition=array(
    'fieldName1'    => $value1,
    'fieldName2 >=' => $value2,
    'fieldName3'    => null,
))
```

See next section for *query conditions* with `db_delete()` and `db_delete_multi()`.

8.5 Query Conditions

You can provide a condition array to third or fourth parameter to `db_update()` and second parameter to `db_delete()` or `db_delete_multi()`. You can also use `db_and()` and `db_or()`. The following are some examples.

Updating with simple condition:

```
db_update('post', array(
    'postTitle' => 'Updated Title',
), array(
    'postId' => 1
));
// UPDATE post SET
//   slug = "updated-title",
//   postTitle = "Updated Title",
//   updated = "....."
// WHERE postId = 1
```

Updating using AND condition:

```
db_update('post', array(
        'catId' => 1,
    ),
    false, // slug field is not updated
    db_and(array(
        'id' => 1,
        'delete !=' => NULL
    ))
);
// UPDATE post SET
//   catId = 1,
//   updated = "....."
// WHERE id = 1 AND deleted IS NOT NULL
```

Updating using IN condition:

```
db_update('post', array(
        'catId' => 1,
    ),
    false, // slug field is not updated
    array(
        'postId' => array(1, 2, 3)
    ))
);
// UPDATE post SET
//   catId = 1,
```

(continues on next page)

(continued from previous page)

```
//   updated = "....."
// WHERE postId IN (1, 2, 3)
```

Updating using OR condition:

```
db_update('post', array(
    'catId' => 1,
    ),
    false, // slug field is not updated,
    db_or(
        array('postId' => 1),
        array('postId' => 2)
    )
);
// UPDATE post SET
//   catId = 1,
//   updated = "....."
// WHERE postId = 1 OR postId = 2
```

Updating using IN and OR condition:

```
db_update('post', array(
        'catId' => 1,
    ),
    false, // slug field is not updated
    db_or(array(
        'id' => array(1, 2, 3),
        'id >' => 10,
    ))
);
// UPDATE post SET
//   catId = 1,
//   updated = "....."
// WHERE id IN (1, 2, 3) OR id > 10
```

Updating with complex AND/OR condition:

```
db_update('post', array(
        'catId' => 1,
    ),
    false, // slug field is not updated
    db_and(array(
        'postTitle' => 'a project',
        'catId' => 2,
        db_or(array(
            'id' => array(1, 2, 3),
            'id >=' => 10,
        ))
    ))
);
// UPDATE post SET
//   catId = 1,
//   updated = "....."
// WHERE postTitle = "a project"
// AND catId= 2
// AND ( id IN (1, 2, 3) OR id >= 10 )
```

8.6 Condition Operators

Operator	Usage Example	Equivalent SQL Condition
=	`array('postId' => 1)` `array('postId' => array(1, 2, 3))`	`WHERE postId = 1` `WHERE postId IN (1, 2, 3)`
!=	`array('postId !=' => 1)` `array('postId !=' => array(1, 2, 3))`	`WHERE postId != 1` `WHERE postId NOT IN (1, 2, 3)`
>	`array('postId >' => 1)`	`WHERE postId > 1`
>=	`array('postId >=' => 1)`	`WHERE postId >= 1`
<	`array('postId <' => 1)`	`WHERE postId < 1`
<=	`array('postId <=' => 1)`	`WHERE postId <= 1`
between	`array('postId between' => array(1, 10))`	`WHERE postId BETWEEN 1 and 10`
nbetween	`array('postId nbetween' => array(1, 10))`	`WHERE postId NOT BETWEEN 1 and 10`
like like%%	`array('postTitle like' => 'a project')` `array('postTitle like%%' => 'a project')`	`WHERE postTitle LIKE "%a project%"`
like%~	`array('postTitle like%~' => 'a project')`	`WHERE postTitle LIKE "%a project"`
like~%	`array('postTitle like~%' => 'a project')`	`WHERE postTitle LIKE "a project%"`
nlike nlike%%	`array('postTitle nlike' => 'a project')` `array('postTitle nlike%%' => 'a project')`	`WHERE postTitle NOT LIKE "%a project%"`
nlike%~	`array('postTitle nlike%~' => 'a project')`	`WHERE postTitle NOT LIKE "%a project"`
nlike~%	`array('postTitle nlike~%' => 'a project')`	`WHERE postTitle NOT LIKE "a project%"`

8.7 Connecting to Multiple Databases

Sometimes, we need to connect multiple databases in our app. . In `/inc/config.php` (copy of `/inc/config.default.php`), `$lc_databases` is an array composed of multiple database connection strings. Here's the default syntax, specifying a single connection:

```
$lc_databases = array(
    'default' => array( // default database; you could also have other database␣
↪settings here
          'engine'    => _p('db.default.engine'),
          'host'      => _p('db.default.host'),
          'port'      => _p('db.default.port'),
          'database'  => _p('db.default.database'),
          'username'  => _p('db.default.username'),
          'password'  => _p('db.default.password'),
          'prefix'    => _p('db.default.prefix'),
          'collation' => _p('db.default.collation')
    )
);
```

As an example, you might have two databases, the default database and a legacy database and the syntax would be as below:

```
$lc_databases = array(
    'default' => array( // default database; you could also have other database␣
↪settings here
          'engine'    => _p('db.default.engine'),
          'host'      => _p('db.default.host'),
          'port'      => _p('db.default.port'),
          'database'  => _p('db.default.database'),
          'username'  => _p('db.default.username'),
          'password'  => _p('db.default.password'),
          'prefix'    => _p('db.default.prefix'),
          'collation' => _p('db.default.collation')
    )
    'legacy' => array(
          'engine'    => _p('db.legacy.engine'),
          'host'      => _p('db.legacy.host'),
          'port'      => _p('db.legacy.port'),
          'database'  => _p('db.legacy.database'),
          'username'  => _p('db.legacy.username'),
          'password'  => _p('db.legacy.password'),
          'prefix'    => _p('db.legacy.prefix'),
          'collation' => _p('db.legacy.collation')
    )
);
```

The next step is to define the parameters in `/inc/parameter/development.php` or `/inc/parameter/production.php` for your two databases in the configuration db. Here is any example.

```
return array(
    // ...
    # Database connection information
    'db' => array(
        'default' => array(
            'engine'    => 'mysql', // database engine
            'host'      => 'localhost', // database host
            'port'      => '', // database port
            'database'  => 'lucid_blog', // database name
            'username'  => 'yourusername', // database username
            'password'  => 'yourpassword', // database password
            'prefix'    => '', // table name prefix
            'collation' => 'utf8_general_ci' // database collation
        ),
        'legacy' => array(
            'engine'    => 'mysql',
            'host'      => 'localhost',
            'port'      => '',
            'database'  => 'legacy_db',
            'username'  => 'legacyusername',
            'password'  => 'legacypassword',
            'prefix'    => '', // table name prefix
            'collation' => 'utf8_general_ci'
        )
    ),
    // ...
);
```

When you need to connect to one of the other databases, you activate it by its key name and switch back to the default connection when finished:

```
# Get some information from the legacy database.
db_switch('legacy');
# Fetching data from the `user` table of the legacy database
$result = db_select('user')
    ->where('uid', $uid)
    ->getSingleResult()

# Switch back to the default connection when finished.
db_switch(); // or db_switch('default');
```

8.8 Database Session

Since version 1.5, PHPLucidFrame supports database session management. It is useful when your site is set up with load balancer that distributes workloads across multiple resources. Here's the minimum table schema requirement for database session.

```
CREATE TABLE `lc_sessions` (
  `sid` varchar(64) NOT NULL DEFAULT '',
  `host` varchar(128) NOT NULL DEFAULT '',
  `timestamp` int(11) unsigned DEFAULT NULL,
  `session` longblob NOT NULL DEFAULT '',
  `useragent` varchar(255) NOT NULL DEFAULT '',
  PRIMARY KEY (`sid`)
);
```

Once you have the table created, you just need to configure `$lc_session['type'] = 'database'` in `/inc/config.php` (copy of `/inc/config.default.php`) such as

```
$lc_session = array(
    'type' => 'database',
    'options' => array(
        /* you can configure more options here, see the comments in /inc/config.
↪default.php */
    )
);
```

8.9 Query Builder

As of version 1.9, PHPLucidFrame added a new feature called query builder using `db_select()`. Here are some examples:

```
$result = db_select('post')->getResult();
// SELECT * FROM `post`
_pr($result); // array of results

$result = db_select('post')->getSingleResult();
// SELECT * FROM `post`
_pr($result); // the result object

$postTitle = db_select('post', 'p')
    ->field('postTitle')
    ->fetch();
```

(continues on next page)

(continued from previous page)

```
// SELECT `p`.`postTitle` FROM `post`
echo $postTitle;

$result = db_select('post', 'p')
    ->fields('p', array('postId', 'postTitle', 'created'))
    ->orderBy('p.created', 'desc')
    ->getResult();
// SELECT `p`.`postId`, `p`.`postTitle`, `p`.`created` FROM `post` `p` ORDER BY `p`.
↪`created` DESC
_pr($result); // array of results

$result = db_select('post', 'p')
    ->join('category', 'c', 'p.catId = c.catId')
    ->fields('p', array('postId', 'postTitle', 'created'))
    ->fields('c', array('catName'))
    ->where()
    ->condition('p.postId', 1)
    ->getSingleResult();
// SELECT `p`.`postId`, `p`.`postTitle`, `p`.`created`, `c`.`catName`
// FROM `post` `p`
// INNER JOIN `category` `c` ON `p`.`catId` = `c`.`catId`
// WHERE `p`.`postId` = 1
_pr($result); // the result object

$rowCount = db_count('post')
    ->where()->condition('deleted', null)
    ->fetch();
// SELECT COUNT(*) count FROM `post` WHERE deleted IS NULL
echo $rowCount;

$qb = db_select('post', 'p')
    ->join('category', 'c', 'p.catId = c.catId')
    ->join('user', 'u', 'p.uid = u.uid')
    ->fields('p', array(
        'postId', 'created', 'postTitle', 'postBody',
        array('postTitle_en', 'postTitle_i18n'),
        array('postBody_en', 'postBody_i18n')
    ))
    ->fields('c', array(
        'catName',
        array('catName_en', 'catName_i18n')
    ))
    ->fields('u', array('fullName'))
    ->where()->condition('deleted', null)
    ->orderBy('p.created', 'DESC')
    ->orderBy('u.fullName')
    ->limit(0, 20);

// SELECT `p`.`postId`, `p`.`created`, `p`.`postTitle`, `p`.`postBody`, `p`.
↪`postTitle_en` `postTitle_i18n`, `p`.`postBody_en` `postBody_i18n`, `c`.`catName`,␣
↪`c`.`catName_en` `catName_i18n`, `u`.`fullName` FROM `post` `p`
// INNER JOIN `category` `c` ON `p`.`catId` = `c`.`catId`
// INNER JOIN `user` `u` ON `p`.`uid` = `u`.`uid`
// WHERE `p`.`deleted` IS NULL
// ORDER BY `p`.`created` DESC, `u`.`fullName` ASC
// LIMIT 0, 20
```

(continues on next page)

(continued from previous page)

```
if ($qb->getNumRows()) {
    while ($row = $qb->fetchRow()) {
        // do something with result
    }
} else {
    // no result
}
```

Note:

- More complex query examples can be found in https://github.com/phplucidframe/phplucidframe/blob/master/tests/lib/query_builder.test.php.
- You may also check *how to retrieve data using native SQL*.

CHAPTER 9

Schema Manager

As of version 1.14, PHPLucidFrame added a new feature called **Schema Manager** to manage your site database. A schema file for your database can be defined in the directory `/db`. The file name format is `schema.[namespace].php` where namespace is your database namespace defined in `$lc_databases` of `/inc/config.php`. If `[namespace]` is omitted, "default" is used. The schema file syntax is an array of options and table names that must be returned to the caller. A sample schema file is available at `/db/schema.sample.php` in the release.

9.1 Default Options for Tables

The schema syntax starts with `_options` which is a set of defaults for all tables, but it can be overidden by each table definition.

```
'_options' => array(
    // defaults for all tables; this can be overidden by each table
    'timestamps'    => true, // all tables will have 3 datetime fields - `created`,␣
↪`updated`, `deleted`
    'constraints'   => true, // all FK constraints to all tables
    'engine'        => 'InnoDB', // db engine for all tables
    'charset'       => 'utf8', // charset for all tables
    'collate'       => _p('db.default.collation'), // collation for all tables;␣
↪inherited from /inc/parameter/
),
```

9.2 Table Definition

After then, each table can be defined using table name as key and value as an array of field definition. The following is a snapshot of example schema definition for two tables (category and post) that have one-to-many relation.

```
// array keys are table names without prefix
'category' => array(
```

(continues on next page)

(continued from previous page)

```
    'slug'          => array('type' => 'string', 'length' => 255, 'null' => false,
↪'unique' => true),
    'catName'       => array('type' => 'string', 'length' => 200, 'null' => false),
    'catName_en'    => array('type' => 'string', 'length' => 200, 'null' => true),
    'catName_my'    => array('type' => 'string', 'length' => 200, 'null' => true),
    'options' => array(
        'pk'  => array('catId'),     // type: integer, autoinc: true, null: false,␣
↪unsigned: true
        'timestamps' => true,        // created, updated, deleted; override to _
↪options.timestamps
        'charset'    => 'utf8',      // override to _options.charset
        'collate'    => 'utf8_unicode_ci',  // override to _options.collate
        'engine'     => 'InnoDB',           // override to _options.engine
    ),
    '1:m' => array(
        // one-to-many relation between `category` and `post`
        // there must also be 'm:1' definition at the side of `post`
        'post' => array(
            'name' => 'catId', // FK field name in the other table (defaults to
↪"table_name + _id")
            //'unique'  => false,   // Unique index for FK; defaults to false
            //'default' => null,    // default value for FK; defaults to null
            'cascade'   => true,    // true for ON DELETE CASCADE; false for ON␣
↪DELETE RESTRICT
        ),
    ),
),
'post' => array(
    'slug'          => array('type' => 'string', 'length' => 255, 'null' => false,
↪'unique' => true),
    'postTitle'     => array('type' => 'string', 'null' => false),
    'postTitle_en'  => array('type' => 'string', 'null' => true),
    'postTitle_my'  => array('type' => 'string', 'null' => true),
    'postBody'      => array('type' => 'text', 'null' => false),
    'postBody_en'   => array('type' => 'text', 'null' => true),
    'postBody_my'   => array('type' => 'text', 'null' => true),
    'options' => array(
        'Pk' => array('postId'), // if this is not provided, default field name to␣
↪`id`
    ),
    '1:m' => array(
        // one-to-many relation between `post` and `post_image`
        // there must also be 'm:1' definition at the side of `post_image`
        'post_image' => array(
            'name'      => 'postId',
            'cascade'   => true,
        ),
    ),
    'm:1' => array(
        'category', // reversed 1:m relation between `category` and `post`
        'user',     // reversed 1:m relation between `user` and `post`
    ),
    'm:m' => array(
        // many-to-many relation between `post` and `tag`
        // there must also be 'm:m' definition at the side of `tag`
        'tag' => array(
            'name'      => 'postId',
```

(continues on next page)

(continued from previous page)

```
            'cascade'   => true,
        ),
    ),
),
```

The following describes the rule explanation of table schema array.

Name	Default	Explanation
`{field_name}`		The field name (a valid field name for the underlying database table). Use the field name "slug" for the sluggable field. Generally, you don't have to define primary key field. It will be added by default using the field name "id" with the following rule: • type: integer • autoinc: true • null: false • unsigned: true However, if you want to use other field type (e.g, string type) and rule for your primary key, you must define the field here using your own rule, for example, `'id' => array('type' => 'string', 'length' => 64, 'null' => false)`
`{field_name}.type`		The data type (See *Data Type Mapping Matrix* for the underlying database)
`{field_name}.length`	`255 for string 11 for int/integer 1 for boolean array(0, 0) for decimal array(0, 0) for float`	The length of the field
`{field_name}.null`	true	Allow `NULL` or `NOT NULL`
`{field_name}.default`		The default value for the field
`{field_name}.unsigned`	false	Unsigned or signed
`{field_name}.autoinc`	false	Auto-increment field
`{field_name}.unique`	false	Unique index for the field
`options`		The array of the table options

Continued on next page

Table 1 – continued from previous page

Name	Default	Explanation
`options.pk`	`array('id')`	One or more primary key field names. The default primary key field name is "id". If you want to use a different name rather than "id" (e.g., user_id, post_id), you can define it here. The default primary key field definition is • type: integer • autoinc: true • null: false • unsigned: true
`options.timestamps`	true	Include 3 datetime fields - `created`, `updated`, `deleted`; override to `_optons.timestamps`
`options.charset`	utf8	The charset for the table; override to `_options.charset`
`options.collate`	utf8_unicode_ci	The charset for the table; override to `_options.collate`
`options.engine`	InnoDB	The charset for the table; override to `_options.engine`
`1:m`		One-to-Many relationship; if you define this, there must be `m:1` definition at the many-side table
`1:m.{table_name}`		The name of the many-side table as array key with the following options.
`1:m.{table_name}.name`	table_name + "_id"	The foreign key field name in the many-side table
`1:m.{table_name}.unique`	false	Unique index for the foreign key field
`1:m.{table_name}.default`	null	Default value for the foreign key field
`1:m.{table_name}.cascade`	false	• `true` for `ON DELETE CASCADE` • `false` for `ON DELETE RESTRICT` • `null` for `ON DELETE SET NULL`
`m:1`		Array of table names that are reverse of one-to-many relations to `1:m`
`m:1.{table_name}`		The name of the one-side table
`m:m`		Many-to-many relationship; if you define this, there must be `m:m` definition at the other many-side table
`m:m.{table_name}`		The name of the reference table

Continued on next page

Table 1 – continued from previous page

Name	Default	Explanation
`m:m.{table_name}.table`		Optional pivot table name; if it is not defined, the two table names will be used concatenating with `_to_` such as `table1_to_table2`
`m:m.{table_name}.name`	table_name + "_id"	The reference field name in the pivot table
`m:m.{table_name}.cascade`	false	• `true` for `ON DELETE CASCADE` • `false` for `ON DELETE RESTRICT` • `null` for `ON DELETE SET NULL`
`1:1`		One-to-One relationship
`1:1.{table_name}`		The name of the reference table
`1:1.{table_name}.name`	table_name + "_id"	Foreign key field name that will be included in the table; it maps to the primary key of the reference table
`1:1.{table_name}.cascade`	false	• `true` for `ON DELETE CASCADE` • `false` for `ON DELETE RESTRICT` • `null` for `ON DELETE SET NULL`

9.3 Data Type Mapping Matrix

The following table shows the matrix that contains the mapping information for how a specific type is mapped to the database.

Type Name	MySQL Data Type	Explanation
smallint	SMALLINT	Maps and converts 2-byte integer values. • Unsigned integer with a range of **0** to **65535** • Signed integer with a range of **32768** to **32767**
int/integer	INT	Maps and converts 4-byte integer values. • Unsigned integer with a range of **0** to **4294967295** • Signed integer with a range of **2147483648** to **2147483647**
bigint	BIGINT	Maps and converts 8-byte integer values. • Unsigned integer with a range of **0** to **18446744073709551615** • Signed integer with a range of **9223372036854775808** to **9223372036854775807**
decimal	NUMERIC(p,s)	Maps and converts numeric data with fixed (exact) point precision. The precision (p) represents the number of significant digits that are stored for values. The scale (s) represents the number of digits that can be stored following the decimal point.
float	DOUBLE(p,s)	Maps and converts numeric data with floating (approximate) point precision. The precision (p) represents the number of significant digits that are stored for values. The scale (s) represents the number of digits that can be stored following the decimal point.
string	VARCHAR	Maps and converts string data with a maximum length.
char	CHAR	Maps and converts string data with a fixed length.
binary	VARBINARY	Maps and converts binary string data with a maximum length.
text	TEXT	Maps and converts string data without a maximum length.
blob	BLOB	Maps and converts binary string data withou a maximum length.
array	TEXT	Maps and converts array data based on PHP serialization. It uses serialization to represent an exact copy of your array as string the database and values retrieved from the database are always converted to PHP's array type using deserialization.
json	TEXT	Maps and converts array data based on PHP's JSON encoding functions.

9.4 Loading Your Schema

Assuming that you have created your application database and you have defined your schema in `/db/schema.php` for the database, you can load or import the database using LucidFrame console tool by running the command:

```
$ php lucidframe schema:load
```

It will import the database defined under the namespace "default". If you want to load another database defined under a different namespace, for example "sample", you just need to provide the namespace in the command such as

```
$ php lucidframe schema:load sample
```

9.5 Exporting Your Schema

You can export or dump your database loaded by your schema definition. The LuicdFrame console command `schema:export` will help you.

```
$ php lucidframe schema:export
```

It will export the database of the namespace "default" in the directory `/db/generated/` as `.sql` file. You can also provide the namespace in the command such as

```
$ php lucidframe schema:export sample
```

9.6 Managing Schema Changes

As of version 1.17.0, PHPLucidFrame provides a way to manage schema changes. It helps you to programmatically deploy new versions of your database schema easily in a standardized way.

Let's say an example, we use the sample database as our default and we are adding a new field `wechatUrl` in the table `social_profile`. Let's edit the file `/db/schema.sample.php`

```
'social_profile' => array(
    'facebookUrl'  => array('type' => 'string', 'length' => 100, 'null' => true),
    'twitterUrl'   => array('type' => 'string', 'length' => 100, 'null' => true),
    'gplusUrl'     => array('type' => 'string', 'length' => 100, 'null' => true),
    'linkedinUrl'  => array('type' => 'string', 'length' => 100, 'null' => true),
    'wechatUrl'    => array('type' => 'string', 'length' => 100, 'null' => true), //
↪<- add this
    '1:1' => array(
        // one-to-one relation between `social_profile` and `user`
        // no need to define 1:1 at the side of `user`
        'user' => array(
            'name'      => 'uid',
            'cascade'   => true,
        ),
    ),
),
```

Then, run `schema:diff sample` and it will generate a file with extension **sqlc** in `/db/version/sample`

```
$ php lucidframe schema:diff sample
PHPLucidFrame 1.17.0 by Sithu K.

./db/version/sample/20170406223436.sqlc is exported.
Check the file and run `php lucidframe schema:update sample`
Done.
```

You can open that **sqlc** file and check its content. Finally, you can run `schema:update sample` to apply this changes in your underlying database.

```
$ php lucidframe schema:update sample
PHPLucidFrame 1.17.0 by Sithu K.

IMPORTANT! Backup your database before executing this command.
Some of your data may be lost. Type "y" or "yes" to continue: y

Executing 20170406223436

Your schema has been updated.
Done.
```

The following example will show you in another scenario where renaming the fields. Let's say we are remove `Url` from all field names of the table `social_profile` such as

```
'social_profile' => array(
    'facebook'  => array('type' => 'string', 'length' => 100, 'null' => true),
    'twitter'   => array('type' => 'string', 'length' => 100, 'null' => true),
    'gplus'     => array('type' => 'string', 'length' => 100, 'null' => true),
    'linkedin'  => array('type' => 'string', 'length' => 100, 'null' => true),
    'wechat'    => array('type' => 'string', 'length' => 100, 'null' => true),
    '1:1' => array(
        // one-to-one relation between `social_profile` and `user`
        // no need to define 1:1 at the side of `user`
        'user' => array(
            'name'      => 'uid',
            'cascade'   => true,
        ),
    ),
),
```

Again, run `schema:diff sample` and you will be confirmed for renaming fields.

```
$ php lucidframe schema:diff sample
PHPLucidFrame 1.17.0 by Sithu K.


Type "y" to rename or type "n" to drop/create for the following fields:

Field renaming from `facebookUrl` to `social_profile.facebook`: y
Field renaming from `twitterUrl` to `social_profile.twitter`: y
Field renaming from `gplusUrl` to `social_profile.gplus`: y
Field renaming from `linkedinUrl` to `social_profile.linkedin`: y
Field renaming from `wechatUrl` to `social_profile.wechat`: y

./db/version/sample/20170406224852.sqlc is exported.
Check the file and run `php lucidframe schema:update sample`
Done.
```

Now you can see there are two **sqlc** files in the directory `/db/version/sample`. Then, as suggested above, you just need to run `schema:update sample` to update your database schema.

```
$ php lucidframe schema:update sample
PHPLucidFrame 1.17.0 by Sithu K.

IMPORTANT! Backup your database before executing this command.
Some of your data may be lost. Type "y" or "yes" to continue: y

Executing 20170406224852

Your schema has been updated.
Done.
```

That's it! You now have two version files of your schema changes stored in `/db/version/sample`.

If you are of team of developers and your team uses version control system, those **sqlc** files should be tracked in your VCS to make it available to other developers in the team. When they get the files, they simply needs to run the command `schema:update` to synchronize their databases as yours.

CHAPTER 10

Database Seeding

Database seeding is a very useful feature to initialize your database with default data or sample data. The seeding feature is available since PHPLucidFrame 1.14.

By default, LucidFrame has two directories - `/db/seed/default` and `/db/seed/sample`. If you have only one database for your application, `/db/seed/default` is the right folder where you should create your seeding files. `/db/seed/sample` has the sample seeding files for the sample database which would be the namespace `sample`.

10.1 Seeding Syntax

The following is seeding file syntax.

```
<?php
use LucidFrame\Core\Seeder; // required ony if you have any reference field to insert␣
↪and to use Seeder::setReference()

// Must return the array
return array(
    'order' => 1, // Execution order: lower number will be executed in greater␣
↪priority, especially for reference fields
    'record-1' => array( // "record-1" can be used for the reference field in the␣
↪other file
        // a record is an array of field and value
        // field  => value
        'field1'  => 'value2',
        'field2'  => 'value2',
        'field3'  => Seeder::setReference('record-key-from-previously-executed-seed'),
    ),
    'record-2' => array(
        'field1'  => 'value2',
        'field2'  => 'value2',
        'field3'  => Seeder::setReference('record-key-from-previously-executed-seed'),
    ),
);
```

The record key `record-1` should be unique for each record and is to be used for reference field in other seeding file. Typically it could be the format `{table-name}-{number}`, e.g., `category-1`, `category-2`, `post-1`, `post-2`, etc. However, it is just a naming convention and it does not tie to any rule.

10.2 Seeding Example

Let's say we have 4 tables to be populated with a set of data - `category`, `post`, `tag` and `post_to_tag`. The relationship between category and post is one-to-many relationship; `post` and ‘‘tag‘ have many-to-many relationship. So, there must be 4 seeding PHP files with the names of respective table names.

```
/db
    /default
        |-- category.php
        |-- post.php
        |-- post_to_tag.php
        |-- tag.php
```

The followings are example contents for each seeding file.

category.php

```
<?php
// Data for the table "category"
return array(
    'order' => 1, // Execution order: this file will be executed firstly
    'category-1' => array( // a record is an array of field and value
        // field => value
        'slug'     => 'technology',
        'catName'  => 'Technology',
    ),
    'category-2' => array(
        'slug'     => 'framework',
        'catName'  => 'Framework',
    ),
);
```

post.php

```
<?php
// Data for the table "post"
use LucidFrame\Core\Seeder; // required if you have any reference field to insert

return array(
    'order' => 2, // this file will be executed after seeding is executed for the␣
↪table "category"
    'post-1' => array(
        'catId'     => Seeder::setReference('category-1'), // reference field that␣
↪will be inserted with category id that will be created by the previous category␣
↪seeding execution
        'slug'      => 'welcome-to-the-lucidframe-blog',
        'postTitle' => 'Welcome to the LucidFrame Blog',
        'postBody'  => 'LucidFrame is a mini application development framework - a␣
↪toolkit for PHP developers. It provides logical structure and several helper␣
↪utilities for web application development. It uses a module architecture to make␣
↪the development of complex applications simplified.',
    ),
```

(continues on next page)

(continued from previous page)

```
);
```

tag.php

```
<?php
// Data for the table "tag"
return array(
    'order' => 3, // this file will be executed in third
    'tag-1' => array(
        'slug'    => 'php',
        'tagName' => 'PHP',
    ),
    'tag-2' => array(
        'slug'    => 'mysql',
        'tagName' => 'MySQL',
    ),
);
```

post_to_tag.php

```
<?php
// Data for the many-to-many table "post_to_tag"
use LucidFrame\Core\Seeder;

return array(
    'order' => 4, // this file will be executed lastly in all of four files
    'post-to-tag-1' => array(
        'postId'    => Seeder::setReference('post-1'), // reference field to the␣
↪table "post"
        'tagId'     => Seeder::setReference('tag-1'),  // reference field to the␣
↪table "tag"
    ),
    'post-to-tag-2' => array(
        'postId'    => Seeder::setReference('post-1'),
        'tagId'     => Seeder::setReference('tag-2'),
    ),
);
```

Note:

- You can check the example seeding files at /db/seed/sample.

10.3 Executing Seeds

When you have defined your seeding files, you can load your seeding data into your database using LucidFrame console tool by running the following command:

```
$ php lucidframe db:seed
```

If your database has a different namespace other than “default”, you can also provide the namespace in the command such as

```
$ php lucidframe db:seed sample
```

CHAPTER 11

Forms

You can implement a form in two ways – using AJAX and without using AJAX. PHPLucidFrame provides AJAX form submission by default.

11.1 Creating AJAX Form

Create a form tag as usual. If you do not set the attribute `action`, LucidFrame will look for `action.php` in the same directory and will submit to it. Until this time of writing, `id="formId"` must be used. Your form should also have a message container in which any error message can be shown.

```
<form name="yourFormName" id="yourFormId" method="post">
    <div class="message error"></div>
    <!-- HTML input elements here as usual -->
    <?php form_token(); ?>
</form>
```

You can also provide the `action` attribute for your desired form handling file.

```
<form name="yourFormName" id="yourFormId" action="<?php echo _url('path/to/action.php
↪'); ?>" method="post">
    <div class="message error"></div>
    <!-- HTML input elements here as usual -->
    <?php form_token(); ?>
</form>
```

One of the following two button implementation should be made to your AJAX form.

1. `<input type="submit" />` or `<button type="submit">..</button>`
2. `<input type="button" class="submit" />` or `<button type="button" class="submit">...</button>`

If your form has no submit type button and if you want the form submission when pressing "Enter" in any text box, set `class="default-submit"` to the form tag.

11.2 Creating A Generic Form Without AJAX

You can make a generic form submission without AJAX using `class="no-ajax"` in the `<form>` tag.

```
<form name="yourFormName" id="yourFormId" method="post" class="no-ajax">
    <div class="message error"></div>
    <!-- HTML input elements here as usual -->
    <?php form_token(); ?>
</form>
```

11.3 Form Action Handling & Validation

The following is a scaffold of AJAX form handling and validation. You can check the sample codes in the release.

```
<?php
/**
 * The action.php (optional) handles form submission.
 * It should perform form validation, create, update, delete of data manipulation to␣
↪database.
 * By default, a form is initiated for AJAX and action.php is automatically invoked␣
↪if the action attribute is not given in the <form> tag.
 */
$success = false;

if (sizeof($_POST)) {
    $post = _post($_POST); // Sanitize your inputs

    /** Form validation prerequisites here, for example */
    $validations = array(
        'txtTitle' => array(
            'caption'   => _t('Title'),
            'value'     => $post['txtTitle'],
            'rules'     => array('mandatory'),
        ),
        'txtBody' => array(
            'caption'   => _t('Body'),
            'value'     => $post['txtBody'],
            'rules'     => array('mandatory'),
        ),
    );

    if (form_validate($validations) == true) {
        /**
        Database operations here
        */

        if ($success) {
            form_set('success', true);
            form_set('message', _t('Your successful message is here'));

            // If you want to redirect to another page, use the option 'redirect'
            // form_set('redirect', _url('path/to/your/page'));
        }
    } else {
        form_set('error', validation_get('errors'));
```

(continues on next page)

(continued from previous page)

```
    }
}

// Respond to the client
form_respond('yourFormId'); // HTML form ID must be used here
```

If your form is a generic form without using AJAX, the last line in above code is not required in `action.php`. Instead, you have to use `form_respond('yourFormId', validation_get('errors'))` at the end of the form in `view.php` in order to show error messages correctly.

```
<form name="yourFormName" id="yourFormId" method="post" class="no-ajax">
    <div class="message error"></div>
    <!-- HTML input elements here as usual -->
    <?php form_token(); ?>
</form>
<?php form_respond('yourFormId', validation_get('errors')); ?>
```

11.4 Setting Data Validation

PHPLucidFrame provides a number of functions that aid in form validation. There are several validation rules provided and using them can be quite easy. First of all, a validation array has to be defined and the syntax of the validation array is:

```
$validations = array(
    'htmlIdOrName' => array( // The HTML id or name of the input element
        'caption'    => _t('Your Element Caption'); // The caption to show in the␣
↪error message
        'value'      => $value, // The value to be validated
        'rules'      => array(), // Array of validation rules defined, e.g., array(
↪'mandatory', 'email')
        'min'        => '', // The required property for the rule 'min', 'minLength',
↪'between'
        'max'        => '', // The required property for the rule 'max', 'maxLength',
↪'between'
        'protocol'   => '', // The required property for the rule 'ip'
        'maxSize'    => '', // The required property for the rule 'fileMaxSize'
        'maxWidth'   => '', // The required property for the rule 'fileMaxWidth',
↪'fileMaxDimension'
        'maxHeight'  => '', // The required property for the rule 'fileMaxHeight'
↪'fileMaxDimension'
        'width'      => '', // The required property for the rule 'fileExactDimension'
        'height'     => '', // The required property for the rule 'fileExactDimension'
        'extensions' => '', // The required property for the rule 'fileExtension'
        'dateFormat' => '', // The required property for the rule 'date', 'datetime'
        'pattern'    => '', // The required property for the rule 'custom'
        'parameters' => array(
            // The arguments (starting from the second) passing to the custom␣
↪validation functions
            // this may be needed when you set your custom rule in the property 'rules
↪'
            'validate_customRule' => array('param2', 'param3')
        ),
        'messages'   => array(
            // to overwrite the default validation messages OR
```

(continues on next page)

(continued from previous page)

```
            // to define the custom message for the custom validation rules
            'coreRule' => _t('The overwritten message here'), // 'coreRule' means the␣
↪core validation rule provided by LucidFrame, e.g., mandatory, email, username, etc.
            'validate_customRule' => _t('Your custom message here')
        )
    ),
    'anotherInputHtmlIdOrName' => array(
        // similiar options described above ...
    ),
);
```

The validation array should be passed to ‘‘form_validate()‘ to be processed.

```
if (form_validate($validations) === true) { // or validation_check($validations)
    // ...
}
```

Note:

- `validation_check()` doesn't check the form token generated by `form_token()`.

11.5 Core Validation Rules

The core validation rules are defined in `/lib/helpers/validation_helper.php` and you could also define your own custom validation functions in `/app/helpers/validation_helper.php` which will be autoloaded.

11.5.1 alphaNumeric

The field must only contain letters and numbers (integers). Spaces are not allowed to include.

11.5.2 alphaNumericDash

The field must only contain letters, numbers (integers) and dashes.

11.5.3 alphaNumericSpace

The field must only contain letters, numbers (integers) and spaces.

11.5.4 between

This rule checks the data for the field is within a range. The required options - min, max.

```
$validations = array(
    'txtVote' => array( // txtVote is HTML input element name or id
        'caption' => _t('Vote');
        'value'   => $valueToCheck,
```

(continues on next page)

(continued from previous page)

```
        'rules'   => array('mandatory', 'between'),
        'max'     => 0,
        'max'     => 5,
    ),
); // The error message will be shown as "'Vote' should be between 0 and 5".
```

11.5.5 custom

It is used when a custom regular expression is needed. The required option - `pattern`.

```
$validations = array(
    'txtPhone' => array(
        'caption'  => _t('Phone');
        'value'    => $valueToCheck,
        'rules'    => array('custom'),
        'pattern'  => '/^\(?([0-9])*\)?([ 0-9\-])*([0-9])+$/',
        'messages' => array(
            'custom' => _t('Phone number should have a valid format, e.g., (123) 456␣
↪7890'),
            // if this is not specified, the default message "'Phone' should be a␣
↪valid format." will be shown.
        ),
    ),
);
```

11.5.6 date

This checks the field is a valid date. The option is `dateFormat` - `y-m-d`, `d-m-y` or `m-d-y` where separators can be a period, dash, forward slash, but not allowed space. Default is `y-m-d`.

```
$validations = array(
    'txtDate' => array(
        'caption'   => _t('Date');
        'value'     => $valueToCheck,
        'rules'     => array('date'),
        'dateFormat'=> 'd-m-y', // if not given, the default is y-m-d
    ),
);
```

11.5.7 datetime

This checks the field is a valid date and time. The option is `dateFormat` - `y-m-d`, `d-m-y` or `m-d-y` where separators can be a period, dash, forward slash, but not allowed space. Default is `y-m-d`. The option `timeFormat` can also given - `12` or `24`. See *time*.

```
$validations = array(
    'txtDate' => array(
        'caption'   => _t('Date');
        'value'     => $valueToCheck,
        'rules'     => array('datetime'),
        'dateFormat'=> 'd-m-y', // if not given, the default is y-m-d
```

(continues on next page)

(continued from previous page)

```
        'timeFormat'=> '24', // 12 or 24; if not given, default is both which␣
↪validates against both format
    ),
);
```

11.5.8 domain

This checks the field is a valid domain (alpha-numeric and dash only). It must start with letters and end with letters or numbers.

```
$validations = array(
    'txtSubDomain' => array(
        'caption'   => _t('Sub-domain');
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'domain'),
    ),
); // The error message will be shown as "'Sub-domain' should be a valid domain name␣
↪with letters, numbers and dash only.".
```

11.5.9 email

This checks the field is a valid email address.

```
$validations = array(
    'txtEmail' => array(
        'caption'   => _t('Email');
        'value'     => $valueToChecck,
        'rules'     => array('mandatory', 'email'),
    ),
); // The error message will be shown as "'Email' should be a valid format, e.g.,␣
↪username@example.com".
```

11.5.10 fileExtension

This rule allows you to check the uploaded file extension. The required option is `extension` - array of extensions. See example at *fileMaxDimension*.

11.5.11 fileMaxSize

This rule checks the uploaded file size meets the maximum allowed size. The require option is `maxSize` in MB. See example at *fileMaxDimension*.

11.5.12 fileMaxDimension

This rule checks the width and height of the uploaded image file to not exceed the maximum image dimension allowed. The required options are `maxWidth` and `maxHeight` in pixels.

```
$validations = array(
    'filLogo' => array(
        'caption'    => _t('Logo');
        'value'      => $valueToCheck, // $_FILES['logo']
        'rules'      => array('fileExtension', 'fileMaxSize', 'fileMaxDimension'),
        'extensions' => array('jpg', 'jpeg', 'png', 'gif'), // for the rule
↪'fileExtension'
        'maxSize'    => 20 // 20MB for the rule 'fileMaxSize'
        'maxWidth'   => 1280, // for the rule 'fileMaxDimension'
        'maxHeight'  => 986 // for the rule 'fileMaxDimension',
    ),
);
```

11.5.13 fileExactDimension

This rule checks the width and height of the uploaded image file to meet the image dimension specified. The required options are `width` and `height` in pixels.

11.5.14 fileMaxWidth

This rule checks the width of the uploaded image file to not exceed the maximum image width allowed. The required option is `maxWidth` in pixels.

11.5.15 fileMaxHeight

This rule checks the height of the uploaded image file to not exceed the maximum image width allowed. The required option is `maxHeight` in pixels.

11.5.16 integer

The rule checks the field is a positive or negative integer. No decimal is allowed.

11.5.17 ip

This rule checks the field is a valid IPv4 or IPv6 address. The required property is `protocol` - `v4`, `ipv4`, `v6`, `ipv6` or `both` (default).

```
$validations = array(
    'txtIP' => array(
        'caption'  => _t('IP');
        'value'    => $valueToCheck,
        'rules'    => array('ip'),
        'protocol' => 'ipv4',
    ),
);
```

11.5.18 mandatory

This checks the field is required. `0` is allowed. If you don't want to allow `0`, use the rule *notAllowZero* in combination.

```
$validations = array(
    'txtName' => array(
        'caption'   => _t('Name');
        'value'     => $nameValueToCheck,
        'rules'     => array('mandatory'),
    ),
    'cboCountry' => array(
        'caption'   => _t('Country');
        'value'     => $countryValueToCheck,
        'rules'     => array('mandatory'),
        'messages'  => array(
            'mandatory' => _t('Country must be selected.') // this overwrites the␣
↪default message
        ),
    )
);
```

11.5.19 mandatoryOne

This checks at least one field of the field group is required.

```
<!-- HTML -->
<div id="phones">
    <input type="text" name="txtPhones[]" />
    <input type="text" name="txtPhones[]" />
<div>

### PHP ###
$post = _post($_POST);

$validations = array(
    'phones[]' => array( // HTML id of the group element
        'caption'   => _t('Phone(s)');
        'value'     => $post['txtPhones'],
        'rules'     => array('mandatoryOne'),
    ),
);
```

11.5.20 mandatoryAll

This checks all fields of the field group is required.

```
<!-- HTML -->
<div id="phones">
    <input type="text" name="txtPhones[]" />
    <input type="text" name="txtPhones[]" />
<div>

### PHP ###
$post = _post($_POST);

$validations = array(
    'phones[]' => array( // HTML id of the group element
        'caption'   => _t('Phone(s)');
```

(continues on next page)

(continued from previous page)

```
        'value'     => $post['txtPhones'],
        'rules'     => array('mandatoryAll'),
```

11.5.21 max

This rule checks the data for the field is equal or less than a specific maximum number. The required option - `max`.

```
$validations = array(
    'txtMaxVote' => array(
        'caption' => _t('Max. Vote');
        'value'   => $valueToCheck,
        'rules'   => array('mandatory', 'max'),
        'max'     => 5,
    ),
);
```

11.5.22 maxLength

This rule checks the field string length is less than a specific length. The required option - `max`.

```
$validations = array(
    'txtPassword' => array(
        'caption' => _t('Password');
        'value'   => $valueToCheck,
        'rules'   => array('mandatory', 'minLength', 'maxLength'),
        'min'     => 8,
        'max'     => 20,
    ),
);
```

11.5.23 min

This rule checks the data for the field is equal or greater than a specific minimum number. The required option - `min`.

```
$validations = array(
    'txtNoOfPage' => array(
        'caption' => _t('No. of Pages');
        'value'   => $valueToCheck,
        'rules'   => array('min'),
        'min'     => 100,
    ),
); // The error message will be shown as "'No. of Pages' should be greater than or␣
↪equal to 100.".
```

11.5.24 minLength

This rule checks the field string length is greater than a specific length. The required option - `min`.

```
$validations = array(
    'txtPassword' => array(
        'caption' => _t('Password');
        'value'   => $valueToCheck,
        'rules'   => array('mandatory', 'minLength'),
        'min'     => 8,
    ),
);
```

11.5.25 naturalNumber

The rule checks the field is a positive integer starting from 1. No decimal is allowed.

11.5.26 notAllowZero

This ensures that the field is not zero.

11.5.27 numeric

It checks the field is numeric.

11.5.28 numericDash

The field must only contain numbers (integers) and dashes.

11.5.29 numericSpace

The field must only contain numbers (integers) and spaces.

11.5.30 positiveRationalNumber

It checks the field is a positive numbers. It allows decimals.

11.5.31 rationalNumber

It checks the field is a positive or negative numbers. It allows decimals.

11.5.32 time

This checks the field is a valid 24-hr or 12-hr format. The optional option is `timeFormat` - `12`, `24` or `both` where `both` is default.

```
$validations = array(
    'txtTime' => array(
        'caption'   => _t('Time');
        'value'     => $valueToCheck,
        'rules'     => array('time'),
        'timeFormat'=> '24',
    ),
);
```

11.5.33 url

This rule checks for valid URL formats. It supports **http**, **http(s)** and **ftp(s)**. "**www**" must be included.

```
$validations = array(
    'txtWebsite'  => array(
        'caption' => _t('Company Website');
        'value'   => $valueToCheck,
        'rules'   => array('url'),
    ),
);
```

11.5.34 username

The rule is used to make sure that the field must not contain any special character, start with letters, end with letters and numbers. It can contain underscores (_), dashes (-) and periods (.) in the middle.

```
$validations = array(
    'txtUsername' => array(
        'caption'   => _t('Username');
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'username'),
    ),
);
```

11.5.35 wholeNumber

The rule checks the field is a positive integer starting from `0`. No decimal is allowed.

```
$validations = array(
    'txtPrice' => array(
        'caption'   => _t('Price');
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'wholeNumber'),
    ),
); // The error message will be shown as "'Price' should be a positive integer.".
```

11.6 Custom Validation Rules

In addition to the core validation rules, you could also define your own custom validation functions in `/app/helpers/validation_helper.php`. They will be auto-loaded. The custom validation rule must start with `validate_`.

For example,

```
$validations = array(
    'txtUsername' => array(
        'caption'   => _t('Username');
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'username', 'validate_duplicateUsername'),
        'parameters' => array(
            'validate_duplicateUsername' => array($theEditId), // $theEditId will be␣
↪the second argument to validate_duplicateUsername()
        ),
        'messages' => array(
            'validate_duplicateUsername' => _t('Username already exists. Please try␣
↪another one.'),
        ),
    ),
);
```

Then, you must define a function `validate_duplicateUsername()` in `/app/helpers/validation_helper.php`, for example,

```
/**
 * Custom validation function to check username is duplicate
 * @param string $value Username to be checked
 * @param integer $id The edit id if any
 * @return boolean TRUE for no duplicate; FALSE for duplicate
 */
function validate_duplicateUsername($value, $id = 0) {
    $value = strtolower($value);
    if (empty($value)) {
        return true;
    }

    $qb = db_count('user')
        ->where()
        ->condition('LOWER(username)', strtolower($value));
    if ($id) {
        $qb->condition('id <>', $id);
    }

    return $qb->fetch() ? false, true;
}
```

Alternatively, if you don't want to define a function, you could add it right in your form action handling as the code snippet below. In this case, you have to call `Validation::addError('htmlIdOrName', 'Error message to be shown')`, but it is not recommended.

```
if (form_validate($validations) == true) {
    $qb = db_count('user')
        ->where()
        ->condition('LOWER(username)', strtolower($value));

    if ($id) {
        $qb->condition('id <>', $id);
    }

    if ($qb->fetch()) {
        validation_addError('txtUsername', _t('Username already exists. Please try␣
↪another one.'));
```

(continues on next page)

(continued from previous page)

```
    } else {
        // No duplicate && success
    }
}
```

CHAPTER 12

File Helper

PHPLucidFrame provides a basic file upload handling using generic form and AsynFileUploader that can be used in AJAX form.

12.1 File Upload Form and File Handling

PHPLucidFrame provides a basic file upload handling using generic form. First of all, you could define maximum file upload size, upload directory and required dimension (if image upload) in `/inc/site.config.php`.

```
define('MAX_FILE_UPLOAD_SIZE', 20); // in MB

define('PHOTO_DIR', FILE . 'photos/'); // assuming that you have this directory
define('WEB_PHOTO_DIR', WEB_ROOT . 'files/photos/');

// $lc_photoDimensions - desired width and height for the uploaded photos
// 3 photos will be uploaded according to the defined dimensions
$lc_photoDimensions = array('400x300', '200x150');
```

Since 1.6, a new configuration variable `$lc_imageFilterSet` is added.

```
# $lc_imageFilterSet - Default image filter setting that applies to image upload
$lc_imageFilterSet = array(
    'maxDimension' => '800x600', // or null for client original image size to keep,␣
↪but not recommended
    'resizeMode'   => FILE_RESIZE_BOTH,
    'jpgQuality'   => 75
);
```

The uploaded images will be stored under the following directories according to the above configuration:

```
/path_to_webserver_document_root
    /files
        /photos
```

(continues on next page)

(continued from previous page)

```
            /400x300  <- 400x300 thumbnails
            /200x150  <- 200x150 thumbnails
            |-- xxxxxx.jpg <- primary/original images according to $lc_
↪imageFilterSet[maxDimension]
            |-- .........
            |-- xxxxxx.jpg
```

Note:

- The upload directory must be writable.

According to the framework-recommended page structure, you could have the following structure for your file uploading page:

```
/path_to_webserver_document_root
    /app
        /photo
        |-- action.php
        |-- index.php
        |-- view.php
```

Your `/app/photo/index.php` will look like this:

```
<?php
/** app/post/index.php */

$pageTitle = _t('Photo Uploader');

include('action.php'); // include action.php explicitly
?>
<!DOCTYPE html>
<html lang="<?php echo _lang(); ?>">
<head>
    <title><?php echo _title($pageTitle); ?></title>
    <?php include( _i('inc/head.php') ); ?>
</head>
<body>
    <?php include('view.php'); ?>
</body>
</html>
```

Here is how your `/app/photo/view.php` will go:

```
<!-- app/photo/view.php -->
<form method="post" name="frmPhoto" id="frmPhoto" enctype="multipart/form-data" class=
↪"no-ajax">
    <?php if ($msg = flash_get()) { ?>
        <?php echo $msg; ?>
    <?php } else { ?>
        <div class="message error"></div>
    <?php } ?>
    <table cellpadding="0" cellspacing="0" class="form">
        <tr>
            <td class="label"><?php echo _t('Photo'); ?></td>
            <td class="labelSeparator">:</td>
```

(continues on next page)

(continued from previous page)

```
            <td class="entry">
                <input type="file" name="filPhoto" id="filPhoto" />
            </td>
        </tr>
        <tr>
            <td colspan="2"></td>
            <td class="entry">
                <button type="submit" id="btnUpload" name="btnUpload"><?php echo _t(
↪'Upload'); ?></button>
            </td>
        </tr>
    </table>
    <?php form_respond('frmPhoto', validation_get('errors')); ?>
</form>
```

Note:

- You will have to add `class="no-ajax"` and `enctype="multipart/form-data"` to the form tag because this file upload process needs normal HTTP request.
- `action.php` has to be included explicitly.

Finally, you need the form upload process handling in `/app/photo/action.php` like below:

```
<?php
/** app/photo/action.php */
$success = false;

if (sizeof($_POST)) {
    $post = _post($_POST);
    extract($post);

    # UPLOAD
    $photo = $_FILES['filPhoto'];

    $validations = array(
        'filPhoto' => array(
            'caption'       => _t('Photo'),
            'value'         => $photo,
            'rules'         => array('mandatory', 'fileExtension', 'fileMaxSize'),
            'extensions'    => array('jpg', 'jpeg', 'png', 'gif'),
            'maxSize'       => MAX_FILE_UPLOAD_SIZE,
            'messages'      => array(
                'mandatory' => _t('Please select a photo.')
            ),
        ),
    ),

    if (Validation::check($validations) == true) {
        $file = _fileHelper();
        // set image dimension to resize
        $file->set('dimensions', _cfg('photoDimension'));
        // set file upload directory; default to `/files/tmp/`
        // this should be defined in site.config.php
        $file->set('uploadDir', PHOTO_DIR); // optional
```

(continues on next page)

(continued from previous page)

```
        // image resize mode:
        // FILE_RESIZE_BOTH (by default) - resize to the fitted dimension to the␣
↪given dimension
        // FILE_RESIZE_WIDTH - resize to the given width, but height is aspect ratio␣
↪of the width
        // FILE_RESIZE_HEIGHT - resize to the given height, but width is aspect ratio␣
↪of the height
        $file->set('resize', FILE_RESIZE_BOTH);

        $uploads = $file->upload($photo);
        /**
        $upload will return in the format:
        array(
            'name'              => 'Name of the input element',
            'fileName'          => 'The uploaded file name',
            'originalFileName'  => 'The original file name',
            'extension'         => 'The selected and uploaded file extension',
            'dir'               => 'The uploaded directory',
        )
        */
        if ($uploads) {
            $data = array(
                'filename' => $uploads['fileName'];
            );

            if (db_insert('your_table', $data, false)) {
                $success = true;
                form_set('success', true);
                flash_set(_t('The photo has been uploaded.'));
                _redirect(); // or _redirect('self')
                // redirect to the current page itself
                // and will show the flash message set above.
            }
        } else {
            $error = $file->getError();
            Validation::addError('filPhoto', $error['message']);
            form_set('error', validation_get('errors'));
        }
    } else {
        form_set('error', validation_get('errors'));
    }
}
```

12.2 AsynFileUploader (Asynchronous File Uploader)

The file helper in the previous section is not compatible with AJAX form. Since version 1.3, PHPLucidFrame added a new feature "**AsynFileUploader**" that helps you to upload a file in smarter way with instant preview and that is compatible with AJAX form.

Firstly, you can have a few image-related configurations in `/inc/site.config.php` or `/app/inc/site.config.php` as described in the previous section *File Upload Form and File Handling*.

Create an instance of the class **AsynFileUploader** and call its method `html()` at where you want to display the file uploader. Normally it is in your view layer, for example, /app/example/asyn-file-uploader/view.php.

```
<form id="frmAsynFileUpload" method="post">
    <div class="message error"></div>
    <div class="table">
        <div class="row">
            <?php
            # The constructor argument
            # string/array The input file name or The array of property/value pairs
            $file = _asynFileUploader('photo');

            # Button caption; default to "Choose File"
            $file->setCaption('Choose Image');

            # Max file upload size; default to 10MB
            $file->setMaxSize(MAX_FILE_UPLOAD_SIZE);

            # Image dimension to resize
            # $lc_photoDimensions could be defined in /inc/site.config.php (see in␣
↪the previous section).
            # This is not required for the non-image file uploads
            $file->setDimensions($lc_photoDimensions);

            # Allowed file extensions; default to any file
            $file->setExtensions(array('jpg', 'jpeg', 'png', 'gif'));

            # The file uploaded directory; default to /files/tmp
            # PHOTO_DIR could be defined in /inc/site.config.php (see in the previous␣
↪section).
            $file->setUploadDir(PHOTO_DIR);

            # The button #btnSubmit will be disabled while the upload is in progress
            $file->setButtons('btnSubmit');

            # The uploaded file name is displayed or not below the file upload button;
↪ default is true
            $file->isFileNameDisplayed(false);

            # The uploaded file name is allowed to delete or not; default is true;
            # The delete icon will be displayed when it is true
            $file->isDeletable(false);

            # The OnUpload hook which could be defined in /app/helpers/file_helper.php
            # This hook runs when the file is uploaded
            $file->setOnUpload('example_photoUpload');

            # The OnDelete hook which could be defined in /app/helpers/file_helper.php
            # This hook runs when the file is delete
            $file->setOnDelete('example_photoDelete');

            # If there is any previously uploaded files, set them using setValue()
            # $images could be retrieved in query.php
            # @see /app/example/asyn-file-uploader/query.php
            # @see /app/example/asyn-file-uploader/view.php
            if (isset($images) && $images) {
                # $image is retrieved in query.php
                $file->setValue($image->pimgFileName, $image->pimgId);
            }
```

(continues on next page)

(continued from previous page)

```
            $file->html();
            ?>
        </div>
        <div class="row">
            <input type="submit" id="btnSubmit" name="btnSubmit" value="<?php echo _t(
↪'Submit'); ?>" class="button green" />
        </div>
    </div>
    <?php form_token(); ?>
</form>
```

As the form In the above coding is attached to AJAX and the form action attribute is not explicitly defined, it will submit to action.php in the same level directory when the button `#btnSubmit` is clicked, for example, /app/example/asyn-file-uploader/action.php.

The following is an example code for the possible `action.php` where you will have to use the name which you provided to the **AsynFileUploader** constructor in the previous code example, i.e., photo. LucidFrame automatically adds some additional file upload information upon form submission that you can get them from the `POST` array. See the code below.

```
<?php
$success = false;

if (sizeof($_POST)) {
    $post = _post($_POST);

    $validations = array(
        'photo' => array(
            'caption' => _t('Image') ,
            'value' => $post['photo'],
            'rules' => array(
                'mandatory'

            ) ,
        )
    );

    if (form_validate($validations) === true) {

        // # You can get the uploaded file information as below
        // $post['photo']            = The uploaded file name saved in disk
        // $post['photo-id']         = The ID in database related to the previously␣
↪uploaded file
        // $post['photo-dimensions'] = (Optional) Array of dimensions used to resize␣
↪the images uploaded
        // $post['photo-dir']        = The directory where the file(s) are saved,␣
↪encoded by base64_encode()
        // $post['photo-fileName']   = The same value of $post['photo']
        // $post['photo-anyKey']     = if you set it using AsynFileUploader->
↪setHidden('anyKey', 'anyValue')
        // ## Do database operation here ###

        $success = true;
        if ($success) {
            form_set('success', true);
            form_set('message', _t('The photo has been saved.'));
```

(continues on next page)

(continued from previous page)

```
        }
    } else {
        form_set('error', validation_get('errors'));
    }
}

form_respond('frmAsynFileUpload');
```

12.3 PHP Hooks for AsynFileUploader

There are some available hooks to be run during file handling process of AsynFileUploader.

12.3.1 The onUpload hook

The hook is to do database operation regarding to the uploaded files and it runs just after file is uploaded. It can be defined in `/app/helpers/file_helper.php` and the hook function name has to be given in the method call `setOnUpload()`. The two arguments will be passed to your function.

Argument	Type	Description
Argument 1	array	The array of the following keys of the uploaded file information: • `name` Name of the input element • `fileName` The uploaded file name • `originalFileName` The original file name • `extension` The selected and uploaded file extension • `dir` The uploaded directory
Argument 2	array	The POSTed information: • `{name}` Array of the file names uploaded and saved in drive • `{name}-id` Optional array of the database value IDs (if a file have previously been uploaded) • `{name}-dimensions` Optional array of the file dimensions in WxH (it will not be available if it is not an image file) • `{name}-{fieldname}` Optional hidden values If you set the name "**photo**" to the **AsynFileUploader** constructor, the keys will be `photo`, `photo-id` and `photo-dimensions`. If you set `AsynFileUploader->setHidden('key', 'value')`, you can get it here using `photo-key`.

The hook must return an array of IDs.

12.3.2 The onDelete hook

This hook is to do database operation regarding to the deleted files. It runs when delete button is clicked and just after file is deleted. It can be defined in `/app/helpers/file_helper.php` and the hook function name has to be given in the method call `setOnDelete()`. An argument will be passed to your function:

Argument	Type	Description
Argument 1	mixed	The ID related to the file deleted to delete from the database table.

Note:

- See the example code at /app/helpers/file-helper.php.

12.4 Javascript Hooks for AsynFileUploader

Besides the server-side hooks, there are some available javascript hooks to be run during file handling process of AsynFileUploader. They are `afterUpload`, `afterDelete` and `onError`. Each can be defined using `LC.AsynFileUploader.addHook(name, hook, function)` where the parameters are:

Argument	Type	Description
name	string	The name you given for `AsynFileUploader`.
hook	string	The hook name `afterUpload`, `afterDelete` and `onError`.
function	function	The callback function to be called

12.4.1 The afterUpload hook

The hook runs just after file is uploaded. It can be defined using `LC.AsynFileUploader.addHook(yourInputName, 'afterUpload', callback)`. The following two arguments `name` and `file` will be passed to your callback function.

Argument	Type	Description
name	string	The input element name you given for `AsynFileUploader`
file	object	The uploaded file information.
file.name	string	The file input name
file.id	string	The HTML id for the file browsing button
file.value	string	The uploaded file name
file.savedId	mixed	The ID in the database related to the uploaded file (if any)
file.fileName	string	The original file name to be displayed
file.extension	string	The uploaded file extension
file.url	string	The actual file URL
file.caption	string	The caption if the uploaded file is image

12.4.2 The afterDelete hook

The hook runs just after file is deleted. It can be defined using `LC.AsynFileUploader.addHook(yourInputName, 'afterDelete', callback)`. The following two arguments `name` and `data` will be passed to your callback function.

Argument	Type	Description
name	string	The input element name you given for `AsynFileUploader`
data	object	The uploaded file information.
data.name	string	The file input name
data.success	boolean	`true` if file deletion succeeded; otherwise `false`
data.error	string	The error message if file deletion failed
data.id	mixed	The ID deleted from database
data.value	string	The file name deleted from hard drive

12.4.3 The onError hook

The hook runs when the file upload fails with error. It can be defined using `LC.AsynFileUploader.addHook(yourInputName, 'onError', callback)`. The two arguments `name` and `error` will be passed to your callback function.

Argument	Type	Description
name	string	The input element name you given for `AsynFileUploader`
error	object	The error object
error.id	string	The HTML ID which is generally given the validation key option in PHP
error.plain	string	The error message in plain format
error.html	mixed	The error message in HTML format

Note:

- If you defined this, the error message will not be shown until you code to show the error message in the callback function.
- See the example code in /app/example/asyn-file-uploader/index.php

CHAPTER 13

Pagination

Listings with pagination are required in most of applications. PHPLucidFrame provides two ways of creating listings - with AJAX and without AJAX. There are two configuration variables in `/inc/site.config.php` which will be used here.

```
# $lc_pageNumLimit: number of page numbers to be shown in pager
$lc_pageNumLimit = 10;
# $lc_itemsPerPage: number of items per page in pager
$lc_itemsPerPage = 15;
```

For every grow-in-size and data-centric web application, displaying a reasonable number of records per page has always been a crucial part. PHPLucidFrame provides a quick, easy and handy way to paginate data to cure headaches of developers. LucidFrame offers a class Pager for pagination to make building paginated queries easier.

We start by getting the number of total records which is expected by the class Pager.

```
# Count query for the pager
$totalCount = db_count('post')
    ->where()->condition('deleted', null)
    ->fetch();
```

Once we retrieved the number of total records, we can create an instance from the Pager class. By default, the helper uses the query string for the current page number page. We can customize it by passing the name to the constructor like new `Pager('pg')`.

The instance looks for the number of records per page `$lc_itemsPerPage` and the number of page limit `$lc_pageNumLimit`. The total number of records has to be set to the instance as well. If we use images for navigation arrows, we can set the image directory path to the `imagePath` property of the instance, but we can also make them from CSS.

To incorporate AJAX functionality into pagination, you can make it easily by setting the ajax property to true. The `calculate()` method has to be invoked to calculate offset.

```
# Prerequisite for the Pager
$pager = _pager();
```

(continues on next page)

(continued from previous page)

```
$pager->set('itemsPerPage', _cfg('itemsPerPage')); // $lc_itemsPerPage
$pager->set('pageNumLimit', _cfg('pageNumLimit')); // $lc_pageNumLimit
$pager->set('total', $totalCount);                 // the total record count
$pager->set('imagePath', WEB_ROOT.'images/pager/');// optional; if you use images for␣
↪pagination
$pager->set('ajax', true);                         // flag this is AJAX-enabled list
$pager->calculate();                               // required to calculate offset
```

Then, we can use `$pager->get('offset')` and `$pager->get('itemsPerPage')` in our query `LIMIT` clause.

```
$qb = db_select('post', 'p')
    ->where()->condition('p.deleted', null)
    ->orderBy('p.created', 'DESC')
    ->limit($pager->get('offset'), $pager->get('itemsPerPage'));
```

Finally, we call `$pager->display()` where we want to appear the pager. By default, the pager will be displayed using `<table class="pager">` tag, however, we can easily change it to `<ul>` or `<div>` by setting `$pager->set('htmlTag', '<ul>')` or `$pager->set('htmlTag', '<div>')`. The default output HTML will look like:

```
<table cellspacing="0" cellpadding="0" border="0" class="pager">
    <tbody>
        <tr>
            <td class="first-disabled">
                <label>First</label>
            </td>
            <td class="prev-disabled">
                <label>« Prev</label>
            </td>
            <td class="pages">
                <span class="currentPage">1</span>
                <span><a href="">2</a></span>
            </td>
            <td class="next-enabled">
                <a href="">
                    <label>Next »</label>
                </a>
            </td>
            <td class="last-enabled">
                <a href="">
                    <label>Last</label>
                </a>
            </td>
        </tr>
    </tbody>
</table>
```

If we use `imagePath`, the output HTML will be generated with `<img />` tag. The following images have to be available in our `imagePath`:

- end.png
- end_disabled.png
- next.png
- next_disabled.png

- previous.png
- previous_disabled.png
- start.png
- start_disabled.png

```
<table cellspacing="0" cellpadding="0" border="0" class="pager">
    <tbody>
        <tr>
            <td class="first-disabled"><img ... /></td>
            <td class="prev-disabled"><img ... /></td>
            <td class="pages">
                <span class="currentPage">1</span>
                <span><a href="">2</a></span>
            </td>
            <td class="next-enabled">
                <a href=""><img ... /></a>
            </td>
            <td class="last-enabled">
                <a href=""><img ... /></a>
            </td>
        </tr>
    </tbody>
</table>
```

If we use `$pager->set('htmlTag', '<ul>')`, the output will look like:

```
<ul class="pager">
    <li class="first-disabled">
        <label>First</label>
    </li>
    <li class="prev-disabled">
        <label>« Prev</label>
    </li>
    <li class="pages">
        <span class="currentPage">1</span>
        <span><a href="">2</a></span>
    </li>
    <li class="next-enabled">
        <a href="">
            <label>Next »</label>
        </a>
    </li>
    <li class="last-enabled">
        <a href="">
            <label>Last</label>
        </a>
    </li>
</ul>
```

If we use `$pager->set('htmlTag', '<div>')`, the output will look like:

```
<div class="pager">
    <div class="first-disabled">
        <label>First</label>
    </div>
    <div class="prev-disabled">
```

(continues on next page)

(continued from previous page)

```
        <label>« Prev</label>
    </div>
    <div class="pages">
        <span class="currentPage">1</span>
        <span><a href="">2</a></span>
    </div>
    <div class="next-enabled">
        <a href="">
            <label>Next »</label>
        </a>
    </div>
    <div class="last-enabled">
        <a href="">
            <label>Last</label>
        </a>
    </div>
</div>
```

We can adjust and extend the default pager CSS in `/css/base.css` according to our needs or we can write it in our own.

13.1 Create an AJAX Listing Page

According to the framework-recommended page structure, you could have the following structure for your listing page.

```
/path_to_webserver_document_root
    /app
        /post
            |-- index.php
            |-- list.php
            |-- view.php
```

In you `/app/post/view.php` you need to create an empty HTML container which AJAX will respond HTML to.

```
<!-- app/post/view.php -->
<?php include( _i('inc/header.php') ); ?>

<h3><?php echo $pageTitle; ?></h3>

<div id="list"></div> <!-- #list will be a first parameter to Page.request(). See␣
↪later -->

<?php include( _i('inc/footer.php') ); ?>
```

Create a small javascript snippet in your `/app/js/site.js`.

```
/** app/js/site.js */
LC.Page.Post = {
    url : LC.Page.url('post'), /* mapping directory */
    /* Initialization of the Post page */
    init : function() {
        LC.Page.Post.list();
    },
```

(continues on next page)

(continued from previous page)

```
    list : function() {
        /* LC.Page.request('HTML container ID', 'Requested URL', 'Optional Parameter␣
↪in JSON {}'); */
        LC.Page.request('list', Page.Post.url + 'list.php');
    }
}
```

Call the script at the end of `/app/post/view.php`

```
<?php include( _i('inc/header.php') ); ?>

<h3><?php echo $pageTitle; ?></h3>

<div id="list"></div> <!-- #list will be a first parameter to Page.request(). See␣
↪later -->

<?php include( _i('inc/footer.php') ); ?>

<script type="text/javascript">
    $(function() {
        LC.Page.Post.init();
    });
</script>
```

Finally you have to write `/app/post/list.php` to request and respond by AJAX. In the script, query, paginate and display your data.

```
<?php
/** app/post/list.php */

$get = _get($_GET);

# Count query for the pager
$totalCount = db_count('post')
    ->where()->condition('deleted', null)
    ->fetch();

# Prerequisite for the Pager
$pager = _pager();
$pager->set('itemsPerPage', _cfg('itemsPerPage')); // $lc_itemsPerPage
$pager->set('pageNumLimit', _cfg('pageNumLimit')); // $lc_pageNumLimit
$pager->set('total', $totalCount);                 // the total record count
$pager->set('imagePath', WEB_ROOT.'images/pager/');// optional; if you use images for␣
↪pagination
$pager->set('ajax', true);                         // flag this is AJAX-enabled list
$pager->calculate();                               // required to calculate offset

$qb = db_select('post', 'p')
    ->where()->condition('p.deleted', null)
    ->orderBy('p.created', 'DESC')
    ->limit($pager->get('offset'), $pager->get('itemsPerPage'));
?>

<?php if ($qb->getNumRows()) { ?>
    <?php while ($row = $qb->fetchRow()) { ?>
        <p class="post">
```

(continues on next page)

(continued from previous page)

```
            <h5>
                <a href="<?php echo _url('post', array($row->postId, $row->slug)); ?>
↪"><?php echo $row->title; ?></a>
            </h5>
            <p><?php echo $b->body; ?></p>
            <p>
                <a href="<?php echo _url('post', array($row->postId, $row->slug)); ?>
↪" class="button mini green"><?php echo _t('Read More'); ?></a>
            </p>
        </p>
    <?php } // while end ?>
    <!-- display the pager where you want to appear -->
    <div class="pager clearfix">
        <?php echo $pager->display(); ?>
        <div class="pagerRecords"><?php echo _t('Total %d records', $totalCount); ?></
↪div>
    </div>
<?php } else { ?>
    <div class="noRecord"><?php echo _t('There is no record.'); ?></div>
<?php } ?>
```

13.2 Create a Generic Listing Page without AJAX

Sometimes, you may not want to use AJAX list. You can easily disable LucidFrame pagination helper for AJAX. In this case, you don't need to have `/app/post/list.php` like in the above example. Instead, you should have `/app/post/query.php`. Retrieve your data in `query.php` and then render your HTML in `/app/post/view.php`.

```
/path_to_webserver_document_root
    /app
        /post
            |-- index.php
            |-- query.php
            |-- view.php
```

Include `query.php` in your `/app/post/index.php` like below. You don't need to write Javascript in this case.

```
<?php
/** app/post/index.php */

$pageTitle = _t('Latest Posts');

include('query.php');
?>
<!DOCTYPE html>
<html lang="<?php echo _lang(); ?>">
<head>
    <title><?php echo _title($pageTitle); ?></title>
    <?php include( _i('inc/head.php') ); ?>
</head>
<body>
    <?php include('view.php'); ?>
</body>
</html>
```

In `query.php`, retrieve and paginate your data.

```
<?php
/** app/post/query.php */

# Count query for the pager
$totalCount = db_count('post')
    ->where()->condition('deleted', null)
    ->fetch();

# Prerequisite for the Pager
$pager = _pager();
$pager->set('itemsPerPage', _cfg('itemsPerPage')); // $lc_itemsPerPage
$pager->set('pageNumLimit', _cfg('pageNumLimit')); // $lc_pageNumLimit
$pager->set('total', $totalCount);                 // the total record count
$pager->set('imagePath', WEB_ROOT.'images/pager/');// optional; if you use images for␣
↪pagination
$pager->set('ajax', false);                        // trun off AJAX
$pager->calculate();                               // required to calculate offset

$qb = db_select('post', 'p')
    ->where()->condition('p.deleted', null)
    ->orderBy('p.created', 'DESC')
    ->limit($pager->get('offset'), $pager->get('itemsPerPage'));
```

Finally, your `view.php` will look like this:

```
<!-- app/post/view.php -->
<?php include( _i('inc/header.php') ); ?>

<h3><?php echo $pageTitle; ?></h3>
<div id="list">
<?php if ($totalCount) { ?>
    <?php while ($row = $qb->fetchRow()) { ?>
        <p class="post">
            <h5>
                <a href="<?php echo _url('post', array($row->postId, $row->slug)); ?>
↪"><?php echo $row->title; ?></a>
            </h5>
            <p><?php echo $b->body; ?></p>
            <p>
                <a href="<?php echo _url('post', array($row->postId, $row->slug)); ?>
↪" class="button mini green"><?php echo _t('Read More'); ?></a>
            </p>
        </p>
    <?php } // while end ?>
    <!-- display the pager where you want to appear -->
    <div class="pager clearfix">
        <?php echo $pager->display(); ?>
        <div class="pagerRecords"><?php echo _t('Total %d records', $totalCount); ?></
↪div>
    </div>
<?php } else { ?>
    <div class="noRecord"><?php echo _t('There is no record.'); ?></div>
<?php } ?>
</div>

<?php include( _i('inc/footer.php') ); ?>
```

CHAPTER 14

User Authentication

User authentication is one of the critical parts of almost every web application. PHPLucidFrame provides a flexible way of identifying and checking user authentication. There is a configuration variable available in `/inc/config.php` - `$lc_auth`. You can set up your authentication components corresponding to your user data table to load it.

```
/*
 * Auth Module Configuration
 */
# $lc_auth: configuration for the user authentication
# This can be overidden by defining $lc_auth in /inc/site.config.php
$lc_auth = array(
    'table' => '',  // table name, for example, user
    'fields' => array(
        'id' => '',     // PK field name, for example, user_id
        'role' => '',    // User role field name, for example, user_role
    ),
    'perms' => array(
        // allowed permissions
    ),
    'block' => array(
        // blocked permissions
    ),
    /* // for example
        'perms' => array( // editor is not allowed for post deletion
            'editor' => array('post-list', post-add', 'post-edit'),
            'admin' => array('post-list', post-add', 'post-edit', 'post-delete')
        ),
        'block' => array( // editor is denied for post deletion
            'editor' => array('post-delete'),
            'admin' => array() // no action is blocked for admin
        )
        // you don't have to use both - "perms" and "block",
        // according to the above configuration, both are same concept
        // just pick up the one that suits to your need
```

(continues on next page)

(continued from previous page)

```
    */
);
```

You can update it in `config.php` (copy of `config.default.php`) or you can copy it to `/inc/site.config.php` (copy of `site.config.default.php`) and override it.

14.1 Hashing Passwords

Hashing passwords are always required in every secured web application. When user data are inserted in user registration process, it is advisable to hash user input password using the core function `_encrypt()`.

```
$theEncryptedPassword = _encrypt($theUserInputPassword);
```

When validating user input password in log-in process, you should also use `_encrypt()` to check it against the encrypted password stored in the database.

14.2 Logging In and Logging Out

Upon user login form handling of the user inputs, you could query and get the authenticate user id and then you could pass it to `auth_create()`. You are suggested to check the example codes (`/app/admin/login`) in the release.

```
auth_create($theUserID);
```

It will load all of data from the table configured in `$lc_auth` for the given authenticated user and make it available in the global object `$_auth`. You can also get it from `auth_get()`.

If you already have all of your user data, you can pass it to `auth_create()` as second argument. It is useful for multiple table joined result of your user data. The configuration in `$lc_auth` works only for one table.

```
auth_create($theUserID, $theUserDataObject);
```

Besides the user data result available as properties in the returning `$_auth` object, it also contains session id, timestamp and permissions:

- `$_auth->sessId` - The session id for the current session returning from `session_id()`.
- `$_auth->timestamp` - The created date/time of this authentication object
- `$_auth->permissions` - The permissions allowed according to the defined role and perms in `$lc_auth`.
- `$_auth->blocks` - The permissions blocked according to the defined role and block in `$lc_auth`.

Sometimes, you may need to update the `$_auth` object for the user data changes, for example, user's name changed. For that case, you can use `auth_set()` by passing the updated user data object.

```
$_auth->fullname = $theUpdatedFullName;
auth_set($_auth);
```

The function `auth_clear()` is available for use to destroy the current session and to allow user signed out. You can check the sample code `/app/admin/logout` in the release.

14.3 Checking Anonymous User or Logged-in User

PHPLucidFrame provides an easy way to check if the user is anonymous or logged-in.

```
if (auth_isAnonymous()) {
    // do something
}

// or

if (auth_isLoggedIn()) {
    // do something
}
```

14.4 Access Control with Permissions and User Roles

You might assign specific permissions to each user role in the configuration `$lc_auth['fields']['role']` and `$lc_auth['perms']` to fine tune the security, use and administration of the site.

PHPLucidFrame allows you to check the authenticate user is belong to a particular user role by using `auth_role()` and allows you to check the user is accessible to a particular page or section by using `auth_access()`, for example,

```
if (auth_role('editor')) {
    // if user is editor, do something
} else {
    // redirect to the access-denied page
}

if (auth_access('content-delete')) {
    // if user has permission to delete content, do content delete
}

if (auth_block('content-delete')) {
    // if user is denied to delete content
}
```

You could define custom wrapper functions in `/app/helpers/auth_helper.php` for checking the user roles, for example,

```
/**
 * Check if the current logged-in user is admin or not
 */
function auth_isAdmin() {
    return auth_role('admin');
}

/**
 * Check if the current logged-in user is editor or not
 */
function auth_isEditor() {
    return auth_role('editor');
}
```

You can also check the URL routing path to not allow user access to a page. You are suggested to check the code sample /app/admin/inc/authenticate.php in the release.

```
// Let's say the current URL is http://www.example.com/admin/post/edit/99
if (auth_role('editor') && _arg(1) == 'post' && _arg(2) == 'edit') {
    // redirect to the access-denied page
    // the editor is not allowed to access this page.
}
```

CHAPTER 15

Creating A Multi-lingual Site

Making your site supported multiple languages can reach a larger global audience. The internationalization and localization features in PHPLucidFrame makes it much easier. In a single-language application, your code will look like this:

```
<h1>Blog</h1>
```

For a multi-lingual application, you need to internationalize your code by using `_t()` function in your code:

```
<h1><?php echo _t('Blog') ?></h1>
```

However, even though your application is single-language, you are always recommended to use `_t()` function so that you can easily switch from single-language to multi-language application at any time. You can pass multiple arguments to `_t()` like `sprintf()` for the dynamic value replacement.

PHPLucidFrame operates the translation process based on the configuration variables `$lc_translationEnabled`, `$lc_languages` and `$lc_defaultLang`. So, you don't need to worry about overhead using `_t()`.

15.1 Configuration of Internationalization

If you are code is ready for internationalization, you should then configure `/inc/config.php` for your language settings. There are three global configuration variables for this purpose - `$lc_translationEnabled`, `$lc_languages` and `$lc_defaultLang`.

```
# $lc_translationEnabled - Enable/Disable language translation
$lc_translationEnabled = true;
# $lc_languages: Site languages (leave this blank for single-language site)
$lc_languages = array(
    /* 'lang_code' => 'lang_name' */
    'en' => 'English',
    'my' => 'Myanmar',
);
```

(continues on next page)

(continued from previous page)

```
# $lc_defaultLang: Default site language (leave blank for single-language site)
# One of the key of $lc_languages
$lc_defaultLang = 'en';
```

You can also use optional sub-language-code, in capital letters, that specifies the national variety (e.g., en-GB or en-US or zh-CN) for `$lc_languages`. The sub-codes are typically linked with a hyphen.

15.2 Creating PO files

The next step is to create your po files, which contain all translatable strings in your application. PHPLucidFrame will look for your po files in the following location.

```
/i18n/<language-code>.po
```

To create or edit your po files it's recommended that you do not use your favorite editor. There are free tools such as PoEdit which make editing and updating your po files an easy task; especially for updating an existing po file.

Basically, you don't need to have po file for the default language. As per the code sample in the release, there are two languages - English and Myanmar. As English is default language, there is no `en.po` file in the directory. You can copy and rename the file `default.en.po` to `[your-language-code].po`. For example, if you want to create Japanese translation file, you would rename it to `ja.po` and add your translation strings in the file.

There are two types of translations:

- language-string-based translation
- custom key-based translation

Typically, you could use **language-string-based** translation in case a string is same meaning across the pages. For example, you have the "Cancel" buttons in several pages and if they have same meaning, you could have the following one statement in your language po file:

```
msgid "Cancel"
msgstr "Your language meaning for Cancel"
```

You just need to use the only msgid string for all of your "Cancel" buttons.

```
<button type="button" name="btnCancel"><?php echo _t('Cancel'); ?></button>
```

However, you may want to set different meaning for different "Cancel" buttons depending on pages or actions. Then, you should use **custom key-based** translation.

```
msgid "cancel-at-this-page"
msgstr "Your language meaning for Cancel"

msgid "cancel-at-another-page"
msgstr "Your language different meaning for Cancel"
```

In this case, you must use the appropriate key for your "Cancel" buttons in your coding and you must also have a po file for your default language.

```
<button type="button" name="btnCancel">
    <?php echo _t('cancel-at-this-page'); ?>
</button>
```

(continues on next page)

(continued from previous page)

```
<!-- at another page -->
<button type="button" name="btnCancel">
    <?php echo _t('cancel-at-another-page'); ?>
</button>
```

Note:

- If you updated your po file, you have to clear browser cookie or session to take effect.

15.3 Translation of Long Paragraphs

The po files are useful for short messages, if you find you want to translate long paragraphs, or even whole pages - PHPLucidFrame has a way to handle it. Let's say you have a page "About Us" for the whole page translation, you can create a template file in the directory `/i18n/ctn/<language-code>/`.

```
/i18n/ctn/<language-code>/about-us.<language-code>
```

Then, you just need to use `_tc()` function where you want to appear the paragraphs. It will render the current default language file content.

```
echo _tc('about-us');
```

This function is available to use for dynamic value replacement, for example, you could have the placeholders `:contact-url` and `:home-url` in the file content, you can pass the values to the function to replace them.

```
echo _tc('about-us', array(':contact-url' => _url('contact'), ':home-url' => HOME));
```

15.4 Switching the Site Language

Letting your site visitors switch their preferred language is a must for multi-language sites; by clicking the flag or by selecting the option from a language drop-down. There is a Javascript API available - `LC.Page.languageSwitcher()` by passing the language code being switched, for example,

```
$(function() {
    $('#language-combo').change(function () {
        LC.Page.languageSwitcher($(this).val());
    });
});
```

You could also generate hyperlinks to click on the language flags. For this case, you can use the framework function `_self(NULL, $theLanguageCodeToBeSwitched)` to get the current URL replaced by the language code. You could check the sample code in at `/app/inc/tpl/header.php`.

Note:

- For dynamic content generation from database, you are suggested to check the sample codes in the release at `/app/admin`.
- PHPLucidFrame stores all translation strings read from po file in session. If you updated the po file, you need to clear your session to take it affect.

CHAPTER 16

The LucidFrame Console

As of version 1.11.0, PHPLucidFrame introduces command line based utilities that don't need to be accessible from a web browser. The LucidFrame console provides built-in commands available to use and it allows you to create custom command-line commands as well.

Note:

- A command-line (CLI) build of PHP must be available on the system if you plan to use the Console.
- `php` must be available in your system environment variables.

Before get started, make sure you can run the LucidFrame console. Assuming that you are currently at the root of your LucidFrame application, simply run the Console with no argument:

```
$ cd /path/to/yourapp
$ php lucidframe
```

This produces this help message:

```
PHPLucidFrame 1.17.0 by Sithu K.

1.17.0
PHP Version: 5.6.23
The MIT License
Simple, lightweight & yet powerful PHP Application Framework
Copyright (c) 2014-2017, phplucidframe.com
```

16.1 Running a Built-in Command

Since the version 1.11.0, PHPLucidFrame added a basic Console command to generate a secret hash key. You should run it once you installed LucidFrame to re-generate your own secret key for your application.

```
$ php lucidframe secret:generate
```

You can check the help for the command using an option `-h` or `--help` which is available for every command implemented.

```
$ php lucidframe secret:generate -h
```

That produces the help message for the command `secret:generate` as below:

```
PHPLucidFrame 1.17.0 by Sithu K.

Usage:
  secret:generate [options]

Options:
  -h, --help     Display the help message
  -m, --method   The hashing algorithm method (e.g. "md5", "sha256", etc..) [default:
↪"md5"]
  -d, --data     Secret text to be hashed.

Help:
  Generate a secret hash key
```

Any secret text can be given to the command using the option `-d` or `--data`, and the hash method using the option `-m` or `--method`, for example,

```
$ php lucidframe secret:generate --data=any_scret_string --method=sha256
```

You can check all available command list by running

```
$ php lucidframe list
```

16.2 Creating a Basic Command

You can create your own console command. Let's create a simple console greeting command that greets you from the command line. Create `/app/cmd/GreetCommand.php` and add the following to it:

```
_consoleCommand('demo:hello')
    ->setDescription('Greet someone')
    ->addArgument('name', 'Who do you want to greet?')
    ->addOption('yell', 'y', 'If set, the task will yell in uppercase letters', null,␣
↪LC_CONSOLE_OPTION_NOVALUE)
    ->setDefinition(function(\LucidFrame\Console\Command $cmd) {
        $name = $cmd->getArgument('name');
        $msg = $name ? 'Hello ' . $name : 'Hello';
        $msg .= '!';

        if ($cmd->getOption('yell')) {
            $msg = strtoupper($msg);
        }

        _writeln($msg);
    })
    ->register();
```

`demo:hello` is the command name. It accepts one optional argument `name` and one optional option `yell`. For argument, you don't have to specify `name` in your command call, but you will have to provide `--yell` for the option. You could check help for the command before trying out.

```
$ php lucidframe demo:hello -h
```

Now let's try to test the new console command by running

```
$ php lucidframe demo:hello Sithu
```

Since "Sithu" will be passed to the `name` argument, this will print the following output to the command line.

```
Hello Sithu!
```

You can also use the `--yell` option to make everything uppercase.

```
$ php lucidframe demo:hello Sithu --yell
```

This prints:

```
HELLO SITHU!
```

CHAPTER 17

Ajax and Javascript API

PHPLucidFrame makes use of jQuery to take advantage of the increased interest in developing Web 2.0 rich media applications. There are a wide range of built-in Javascript API functions provided.

The framework has a Javascript object declared LC and it has some global variables available to use. You can check them in the section Core Defined Constants & Variables. `LC` has two core objects - `LC.Page` and `LC.Form`. Both provides utilities to make your pages and forms easier.

PHPLucidFrame recommends the code organization in a Javascript file so-called `site.js`, but it is by no means required and you are free to use your prefer file name. You can extend the global Javascript object `LC.Page` and you can create namespace for each page in your application. You are suggested to check the sample example code `/app/js/site.js` in the release.

```
// /app/js/site.js

LC.Page.init = function() {
    // do some fancy stuff.
}

LC.Page.Blog = {
    url: LC.Page.url('blog'), /* mapping directory or route */

    /* Initialization of the blog page */
    init: function() {
        LC.Page.Blog.list();
    },

    /* Load list by Ajax */
    list: function() {
        LC.Page.request('list', LC.Page.Blog.url + 'list.php');
    }
}

$(function() {
    LC.Page.init();
});
```

The file can be included in `/inc/tpl/head.php` or `/app/inc/tpl/head.php` to use it globally.

```
<?php _js('site.js'); ?>
```

17.1 The Page

PHPLucidFrame has an object declared `LC.Page` and it provides API functions available to use.

17.1.1 LC.Page.languageSwitcher(lng)

The helper callback for *language switch* required in multi-language sites.

Parameters:

Name	Type	Description
`lng`	string	The language code to be switched

17.1.2 LC.Page.request(id, url, params, callback)

The Ajax helper. It is a wrapper function to `jQuery.get` or `jQuery.post`.

Parameters:

Name	Type	Description
`id`	string	An HTML element id to attach Ajax and respond HTML to. If `POST/GET` is given, no HTML is responded and any script can be executed upon the request complete.
`url`	string	URL to be requested
`params`	object	The object of query string passing to the page requested, e.g, `{ key: value, key2, value2 }`
`callback`	function	The callback function to be executed upon page request complete.

17.1.3 LC.Page.throbber.register(id, callback)

Register a custom throbber for ajax requests by overriding the default throbber provided by LucidFrame which is triggered whenever Ajax request is thrown.

Parameters:

Name	Type	Description
`id`	string	An HTML element id to attach Ajax pager.
`callback`	function	The functional object like `{ start: function() { .. }, stop: function() { .. } }`

17.1.4 LC.Page.pager(id)

Attach ajax to the pager element. Internal call after `LC.Page.request()`. You can also use this for your own requirement.

Parameters:

Name	Type	Description
`id`	string	An HTML element id to attach Ajax pager.

17.1.5 LC.Page.url(path)

Get the absolute URL corresponding to the given route path.

Parameters:

Name	Type	Description
`path`	string	The route path.

Returns:

`<string>` The complete absolute URL, for example, `http://www.example.com/<language>/blog` if `Page.url('blog')`.

17.1.6 LC.Page.detectCSSFeature(featureName)

Check to see if CSS support is available in the browser.

Parameters:

Name	Type	Description
`featureName`	string	The CSS feature/property name in camel case

Returns:

`<boolean>` true if the CSS feature is supported; otherwise false.

17.2 The Form

PHPLucidFrame has a global object `LC.Form` which provides several useful functions.

17.2.1 Ajaxing your form

Forms are by default initialized for ajax submission if they have a `type="submit"` button or a `type="button"` button which has `class="submit"`. By adding a class `no-ajax` to the form tag, you can detach Ajax from the form.

17.2.2 jQuery Datepicker

If HTML element has a class `datepicker`, it will bind to jQuery datepicker.

17.2.3 jQuery Button

If HTML element has a class `jqbutton`, it will bind to jQuery UI button theme.

17.2.4 LC.Form.clear(formId)

Clear the form element values and form messages.

Parameters:

Name	Type	Description
`formId`	string	HTML element id set to `<form>` tag

17.2.5 LC.Form.data(id)

Get the embedded JSON form data.

Parameters:

Name	Type	Description
`id`	integer	The data row unique id

This function is useful when passing data from PHP to Javascript in the form of JSON and get them in JS for further usage such as loading data into the form elements of the jQuery dialog. The HTML hierarchy must be `#row-{unique-id} .row-data`. Here is an example snippet:

```
<?php while($row = db_fetchObject($result)) { ?>
    <?php
    $data = array(
        'catId' => $row->catId,
        'catName' => $row->catName,
    );
    ?>
    <tr id="row-<?php echo $row->catId; ?>">
        <td class="tableLeft colAction">
            <span class="row-data" style="display:none"><?php echo json_encode($data);
↪ ?></span>
            <a href="javascript:" class="edit" title="Edit" onclick="LC.Page.Category.
↪edit(<?php echo $row->catId; ?>)">
                <span><?php echo _t('Edit'); ?></span>
            </a>
        </td>
        <td class="colAction">
            <a href="#" class="delete" title="Delete" onclick="LC.Page.Category.
↪remove(<?php echo $row->catId; ?>)">
                <span><?php echo _t('Delete'); ?></span>
            </a>
        </td>
        <td class="colName">
            <?php echo $row->catName; ?>
        </td>
    </tr>
<?php } ?>
```

CHAPTER 18

Hooks And Overrides

PHPLucidFrame allows you to define hooks and overrides so that you can interact with core and enhance the functionality of core.

18.1 Hooks

Hooks allow you to interact with the LucidFrame core. A hook is a PHP function which has a defined set of parameters and a specified result type. It is executed upono certain condition. It is very similar to event observers. The available hooks to implement are explained here.

18.1.1 __script()

It can be defined in `/app/helpers/utility_helper.php` and executed after the core function `_script()` in `/lib/helpers/utility_helpers.php` runs. Use this to make global PHP variables available to the global Javascript variable `LC`.

18.1.2 __getLang()

Get the language to process. Read `lang` from query string. Basically, it is useful for admin content management by language.

It can be defined in `/app/helpers/utility_helper.php` and executed after the core function `_getLang()` in `/lib/helpers/utility_helpers.php` runs.

18.1.3 db_insert_<table_name>($table, $data=array(), $useSlug=true)

Customize database insert operation for a specific table, for example, if you define `db_insert_post()` for the `post` table, it will be automatically executed when you call `db_insert('post', ...)`.

It can be defined in `/app/helpers/db_helper.php` and executed when `db_insert()` in `/lib/helpers/db_helper.mysqli.php` is called.

18.1.4 db_update_<table_name>($table, $data=array(), $useSlug=true)

Customize database update operation for a specific table, for example, if you define `db_update_post()` for the `post` table, it will be automaticaclly executed when you call `db_update('post', ...)`.

It can be defined in `/app/helpers/db_helper.php` and executed when `db_update()` in `/lib/helpers/db_helper.mysqli.php` is called.

18.1.5 db_delete_<table_name>($table, $data=array(), $useSlug=true)

Customize database delete operation of **a single record** for a specific table, for example, if you define `db_delete_post()` for the `post` table, it will be automatically executed when you call `db_delete('post', ...)`.

It can be defined in `/app/helpers/db_helper.php` and executed when `db_delete()` in `/lib/helpers/db_helper.mysqli.php` is called.

18.1.6 db_delete_multi_<table_name>($table, $data=array(), $useSlug=true)

Customize database delete operation of **multiple records** for a specific table, for example, if you define `db_delete_multi_post()` for the `post` table, it will be automatically executed when you call `db_delete('post', ...)`.

It can be defined in `/app/helpers/db_helper.php` and executed when `db_delete_multi()` in `/lib/helpers/db_helper.mysqli.php` is called

18.2 Overrides

Overrides allow you to rewrite some functionalities of the LucidFrame core. An override is a PHP function which has a defined set of parameters and a specified result type. The available override to implement are explained here.

18.2.1 __flush($buffer, $mode)

Overrides the core `ob_start` callback function `_flush()` in `/lib/helpers/utility_helpers.php`. You can use this function to manipulate the output buffer before sending it to browser. It can be defined in `/app/helpers/utility_helper.php`.

18.2.2 __metaSeoTags($tags)

Overrides the core function `_metaSeoTags()` in `/lib/helpers/utility_helpers.php`. You can use this function to manipulate the output buffer before sending it to browser. It can be defined in `/app/helpers/utility_helper.php`.

18.2.3 auth_create($id, $data = null)

Overrides the function `auth_create()` in `/lib/helpers/auth_helper.php`. Create user authentication object. It can be defined `/app/helpers/auth_helper.php`.

18.2.4 auth_role($role)

Overrides the function `auth_role()` in `/lib/helpers/auth_helper.php`. Check if the authenticate user has the specific user role. It can be defined in `/app/helpers/auth_helper.php`.

18.2.5 auth_permissions($id, $data = null)

Overrides the function `auth_permissions()` in `/lib/helpers/auth_helper.php`. Get the permissions of a particular role. It can be defined in `/app/helpers/auth_helper.php`.

18.2.6 auth_access($perm)

Overrides the function `auth_access()` in `/lib/helpers/auth_helper.php`. Check if the authenticate uses has a particular permission. It can be defined in `/app/helpers/auth_helper.php`.

18.2.7 flash_set($msg, $name = '', $class = 'success')

Overrides the function `flash_set()` in `/lib/helpers/session_helper.php`. Set the flash message in session. It can be defined in `/app/helpers/session_helper.php`.

18.2.8 flash_get($name = '', $class = 'success')

Overrides the function `flash_get()` in `/lib/helpers/session_helper.php`. Get the flash message from session and then delete it. It can be defined in `/app/helpers/session_helper.php`.

18.2.9 _pr($input, $pre=true)

Overrides the function `_pr()` in `/lib/helpers/utility_helper.php`. Convenience method for `print_r` to display information about a variable in a way that's readable by humans. It can be defined in `/app/helpers/utility_helper.php`

18.2.10 _fstr($value, $glue = ', ', $lastGlue = 'and')

Overrides the function `_fstr()` in `/lib/helpers/utility_helper.php`. Format a string. It can be defined in `/app/helpers/utility_helper.php`.

18.2.11 _fnum($value, $decimals = 2, $unit = '')

Overrides the function `_fnum()` in `/lib/helpers/utility_helper.php`. Format a number. It can be defined in `/app/helpers/utility_helper.php`.

18.2.12 _fnumSmart($value, $decimals = 2, $unit = '')

Overrides the function `_fnumSmart()` in `/lib/helpers/utility_helper.php`. Format a number in a smarter way, i.e., decimal places are omitted where necessary. It can be defined in `/app/helpers/utility_helper.php`.

18.2.13 _fdate($date, $format = '')

Overrides the function `_fdate()` in `/lib/helpers/utility_helper.php`. Format a date. It can be defined in `/app/helpers/utility_helper.php`.

18.2.14 _fdatetime($dateTime, $format = '')

Overrides the function `_fdatetime()` in `/lib/helpers/utility_helper.php`. Format a date/time. It can be defined in `/app/helpers/utility_helper.php`.

18.2.15 _ftimeAgo($time, $format = 'M j Y')

Overrides the function `_ftimeAgo()` in `/lib/helpers/utility_helper.php`. Display elapsed time in wording. It can be defined in `/app/helpers/utility_helper.php`.

18.2.16 _msg($msg, $class = 'error', $return = null, $display = 'display:block')

Overrides the function `_msg()` in `/lib/helpers/utility_helper.php`. Print or return the message formatted with HTML. It can be defined in `/app/helpers/utility_helper.php`.

18.2.17 _randomCode($length=5, $letters = array())

Overrides the function `_randomCode()` in `/lib/helpers/utility_helper.php`. Generate a random string from the given array of letters. It can be defined in `/app/helpers/utility_helper.php`.

18.2.18 _slug($string, $table = '', $condition = null)

Overrides the function `_slug()` in `/lib/helpers/utility_helper.php`. Generate a slug of human-readable keywords. It can be defined in `/app/helpers/utility_helper.php`.