PHPLucidFrame Documentation

Release 3.3.0

Sithu K.

Sep 06, 2023

Table of Contents:

1 About PHPLucidFrame **1**
- 1.1 Prerequisites . . . 1
- 1.2 License . . . 1

2 Installation **3**
- 2.1 Development Environment . . . 3
- 2.2 Production Environment . . . 4
- 2.3 Secret key . . . 5

3 Application Structure **7**
- 3.1 Page Structure . . . 9
- 3.2 Directory and File Precedence . . . 9
- 3.3 Page Workflow . . . 10
- 3.4 Layout Mode . . . 10

4 Configuration & Parameters **13**

5 The View **15**
- 5.1 Creating View . . . 15
- 5.2 Passing Data To view . . . 15
- 5.3 Nested Views . . . 16
- 5.4 Layout File . . . 16
- 5.5 Stylesheets & Scripts In Head . . . 18

6 Bootstrapping **19**

7 Core Defined Constants & Variables **21**

8 URL Routing **23**
- 8.1 Custom Routes . . . 23
- 8.2 Route Groups . . . 24
- 8.3 Accessing URL . . . 25
- 8.4 Creating and Getting URL . . . 25
- 8.5 Redirecting URL . . . 26
- 8.6 Custom URL Rewrite . . . 26

9 File Inclusion **29**

10 Auto-loading Libraries **31**
10.1 Composer . . . 31
10.2 Custom Autoloader . . . 31

11 Database Configuration **33**
11.1 Make Your Credentials Secret . . . 33
11.2 Connecting to Multiple Databases . . . 34
11.3 Database Session . . . 35

12 Working With Data **37**
12.1 Inserting Your Data . . . 37
12.2 Updating Your Data . . . 38
12.3 Deleting Your Data . . . 40
12.4 Query Conditions . . . 40
12.5 Condition Operators . . . 43
12.6 Finding Data . . . 43

13 Query Builder **47**
13.1 Selecting Data for Multiple Results . . . 47
13.2 Selecting Data for Single Result . . . 47
13.3 Selecting Data for Muliple Fields . . . 48
13.4 Selecting Data for Single Field . . . 48
13.5 Joining Tables . . . 48
13.6 Fetching Specific Data (WHERE condition) . . . 49
13.7 Grouping Results . . . 52
13.8 HAVING Condition on Group Result . . . 52
13.9 Ordering Results . . . 53
13.10 Counting Results . . . 53
13.11 Limiting Results . . . 53
13.12 Aggregates . . . 54

14 Native Queries **57**

15 Schema Manager **59**
15.1 Default Options for Tables . . . 59
15.2 Table Definition . . . 59
15.3 Data Type Mapping Matrix . . . 63
15.4 Loading Your Schema . . . 65
15.5 Exporting Your Schema . . . 65
15.6 Managing Schema Changes . . . 65

16 Database Seeding **69**
16.1 Seeding Syntax . . . 69
16.2 Seeding Example . . . 70
16.3 Executing Seeds . . . 71

17 Middleware **73**
17.1 Before Middleware . . . 73
17.2 After Middleware . . . 73
17.3 Order/Precedence for Middlewares . . . 74
17.4 Assigning Middleware to Routes . . . 74

18 Form & Validation **77**
18.1 Creating AJAX Form . . . 77
18.2 Creating A Generic Form Without AJAX . . . 78

18.3 Form Action Handling & Validation . . . 78
18.4 Setting Data Validation . . . 79
18.5 Sanitizing Form Inputs . . . 80
18.6 Core Validation Rules . . . 81
18.7 Custom Validation Rules . . . 89

19 File Helper **91**
19.1 File Upload Form and File Handling . . . 91
19.2 Generic File Upload . . . 92
19.3 AsynFileUploader (Asynchronous File Uploader) . . . 94
19.4 PHP Hooks for AsynFileUploader . . . 97
19.5 Javascript Hooks for AsynFileUploader . . . 100

20 List with Pagination **103**
20.1 Create an AJAX Listing Page . . . 106
20.2 Create an AJAX Listing Page with jQuery Dialog Form . . . 108
20.3 Create a Generic Listing Page without AJAX . . . 112
20.4 Customize Pagination Display . . . 113

21 Authentication & Authorization **117**
21.1 Encrypting Passwords . . . 118
21.2 Logging In and Logging Out . . . 118
21.3 Checking Anonymous User or Logged-in User . . . 118
21.4 Access Control with Permissions and User Roles . . . 119
21.5 Working with Permissions in Your Database . . . 120

22 Creating A Multi-lingual Site **123**
22.1 Configuration of Internationalization . . . 123
22.2 Creating PO files . . . 124
22.3 Translation of Long Paragraphs . . . 125
22.4 Switching the Site Language . . . 125

23 The LucidFrame Console **127**
23.1 Running a Built-in Command . . . 127
23.2 Creating a Basic Command . . . 128

24 Useful Helpers **131**
24.1 _app($name, $value = null) . . . 131
24.2 _cfg($key, $value = ‘’) . . . 132
24.3 _cfgOption($name, $key) . . . 132
24.4 _env($name, $default = ‘’) . . . 132
24.5 _p($name = ‘env’) . . . 133
24.6 _baseUrlWithProtocol() . . . 133
24.7 _arg($index = null, $path = null) . . . 133
24.8 _entity($table, $dbNamespace = null) . . . 133
24.9 _addJsVar($name, $value = ‘’) . . . 134
24.10 _addHeadStyle($file) . . . 134
24.11 _addHeadScript($file) . . . 134
24.12 _json(array $data, $status = 200) . . . 134
24.13 _requestHeader($name) . . . 135
24.14 _r() . . . 135
24.15 _rr() . . . 135
24.16 _url($path = null, $queryStr = array(), $lang = ‘’) . . . 136
24.17 _redirect($path = null, $queryStr = array(), $lang = ‘’, $status = null) . . . 136
24.18 _page404() . . . 136

24.19 _shorten($str, $length = 50, $trail = '...') . . . 136
24.20 _fdate($date = '', $format = '') . . . 137
24.21 _fdatetime($dateTime = '', $format = '') . . . 137
24.22 _randomCode($length = 5, $letters = array(), $prefix = '') . . . 137
24.23 _slug($string, $table = '', array $condition = array()) . . . 138
24.24 _isHttpPost() . . . 138

25 Ajax and Javascript API **139**
25.1 The Page . . . 140
25.2 The Form . . . 141
25.3 The List . . . 143
25.4 LC.DependentUpdater . . . 145
25.5 LC.eval . . . 146
25.6 LC.getKeyByValue . . . 146

26 Hooks And Overrides **147**
26.1 Hooks . . . 147
26.2 Overrides . . . 148

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 5.6 or newer is recommended, but we strongly advise you to use one of the currently supported versions.
- MySQL 5.0 or newer

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 or from the github repository.

1. Extract the downloaded archive in your local webserver document root, and you will get a folder named **phplucidframe-x.y.z** where **x.y.z** would be your downloaded version.
2. Rename it as **acme** (the name is up to you).
3. Change `baseURL` to `acme` in `/inc/parameter/development.php`.
4. Check http://localhost/acme in your browser.

Alternatively, you can install PHPLucidFrame using Composer. Open your terminal and CD to your webserver document root, and then run

```
composer create-project --prefer-dist phplucidframe/phplucidframe acme
```

2.1 Development Environment

In **development**, your directory setup may look something like the following structure so that it can be accessible via `http://localhost/acme`.

```
/path_to_webserver_document_root
    /acme
        /app
        /assets
        /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' => 'acme',
    // ....
);
```

Note: `acme` would be your project or app name.

2.2 Production Environment

In **production**, your directory setup may look something like the following structure so that it can be accessible via `http://www.example.com`

```
/path_to_webserver_document_root
    /app
    /assets
    /db
    /files
    /i18n
    /inc
    /lib
    /tests
    /vendor
    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' => '',
    // ....
);
```

By default, development environment is active. You can make production environment active by running the following command in your terminal.

```
php lucidframe env production
```

If you don't want to use or can't use command line, you can save the file `.lcenv` (in the project root) with the content `production`.

Note:

1. `.lcenv` should have writtable permission.
2. You can check the currently active environment by running the command `php lucidframe env --show`.

2.3 Secret key

To generate a secret key for your application, open your terminal or command line and CD to your project root, and then run `php lucidframe secret:generate`. For more about the PHPLucidFrame console, read the documentation section "The LucidFrame Console".

Alternatively, you can also get a secret key from http://phplucidframe.com/secret-generator and save it in `/inc/.secret` without using command line.

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` • `file_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/services	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/middleware	This directory should contain the files used for middleware.
app/inc	The directory can include the site template files and site configuration file. • `/tpl/layout.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/head.php` (when layoutMode is dis-

3.1 Page Structure

PHPLucidFrame encourages a uniform and structural page organization. In brief, a web page in LucidFrame is represented by a folder containing at least one file: `view.php` or two files: `index.php` and `view.php`.

```
/path_to_webserver_document_root
    /acme
        /app
            /home
                |-- view.php (required)
                |-- index.php (optional)
                |-- action.php (optional)
                |-- list.php (optional)
```

1. 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>`.
2. The **index.php** (optional) serves as the front controller for the requested page, initializing some basic resources and business logic needed to run the page. This is optional. `view.php` will be served as the front controller if `index.php` doesn't exist.
3. 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.
4. 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.

As an example, you can see the directory `/app/home/` and the directories under `/app/example/` of the PHPLucidFrame release you downloaded.

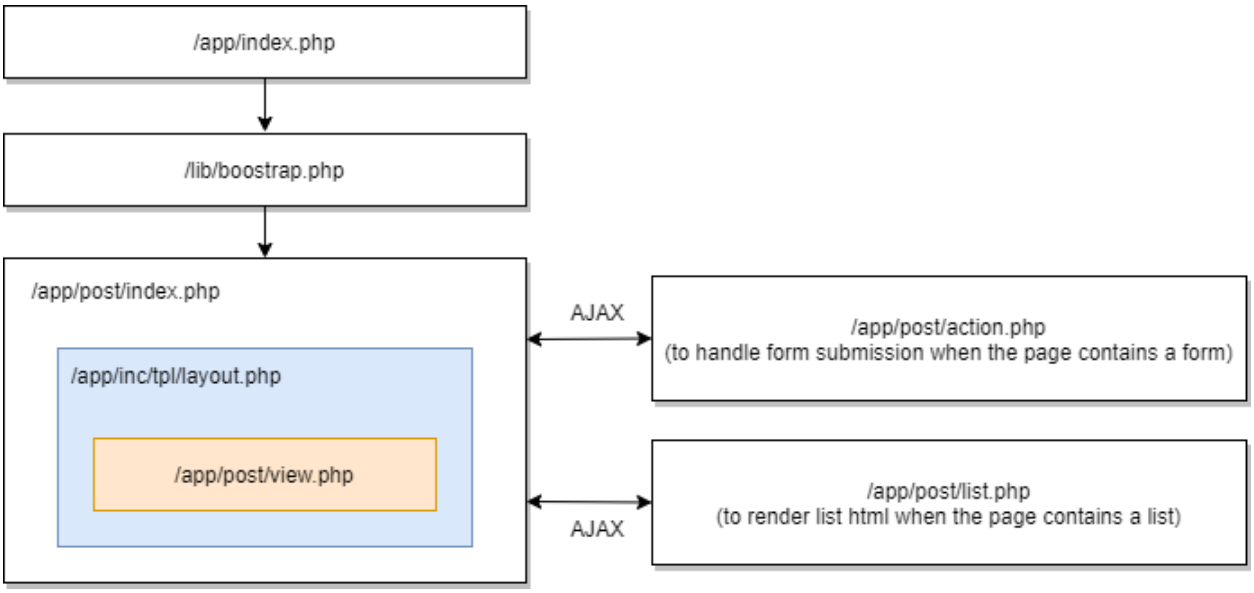
3.2 Directory and File Precedence

PHPLucidFrame has directory and file precedence to look for when a page request is made. For example, a request to `http://www.example.com/post` or `http://localhost/acme/post` will look for the directory and file as the following order:

Order	File	Description
1.	/app/post/view.php	when `index.php` doesn't exist in the `post` directory
2.	/app/post/index.php	when `index.php` and `view.php` eixst in the post directory
3.	/app/post.php	when there is no `post` directory with `view.php`; It is good for implementation without view presentation such as API response with json. `post.php` may end up with `_json(array(...));`

3.3 Page Workflow

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



3.4 Layout Mode

Since version 3.0, layout mode is enabled by default with the following two configurations in `/inc/config.php`.

```
# $lc_layoutMode: Enable layout mode or not
$lc_layoutMode = true;
# $lc_layoutMode: Default layout file name
$lc_layoutName = 'layout'; // default layout file name pointed to app/inc/tpl/layout.
↪php
```

You can see the default layout file `app/inc/tpl/layout.php` which contains the whole page HTML layout and its load the particular page view (`view.php`) by calling `_app('view')->load()`.

You may have a separate layout file for a particular page, let's say for example, you have a login page which have a different layout other than the rest pages of the site. You can create a new layout file `app/inc/tpl/layout_login.php`.

```
/path_to_webserver_document_root
    /acme
        /app
            /inc
                /tpl
                    |-- layout.php
                    |-- layout_login.php
            /login
                |-- action.php
                |-- index.php
                |-- view.php
```

You can set the new layout name for login page in `app/login/index.php` such as

```
_app('view')->layout = 'layout_login';
```

Then, the login page will use `layout_login.php` whereas the other pages use `layout.php`.

3.4.1 Disabling Layout Mode

By disabling layout mode, you can have two template files - `header.php` and `footer.php` in `app/inc/tpl`, and they will have to be included in every `view.php` explicitly. You can disable layout mode by adding the setting in `app/inc/site.config.php`.

```
# $lc_layoutMode: Enable layout mode or not
$lc_layoutMode = false;
```

Then, you can include header and footer files by using `_app('view')->block('fileName')` in each `view.php`.

```
<?php _app('view')->block('header') ?>

<!--- page stuffs here -->

<?php _app('view')->block('footer') ?>
```

If you want to disable layout mode for a particular page only. You can add `_cfg('layoutMode', false);` at the top of `index.php` of the page folder.

Note:

- The disabled layout mode is a legacy way and not recommended since version 3.0. You can check the version 2 documentation about application structure at https://phplucidframe.readthedocs.io/en/v2.2.0/application-structure.html

---

CHAPTER 4

Configuration & Parameters

There are a couple of configuration files in PHPLucidFrame. These configuration files allow you to configure things like your database connection information, your mail server information, as well as various other core configuration values.

	File	Description
1	/inc/config.php	The core configuration settings (copy of `config.default.php`)
2	/inc/route.config.php	Configuration for routes
3	/inc/constants.php	The core global constants
4	/inc/parameter/development.php	The configuration settings for development environment
5	/inc/parameter/production.php	The configuration settings for production environment
6	/inc/parameter/staging.php	The configuration settings for staging environment
7	/inc/parameter/test.php	The configuration settings for testing environment
8	/inc/parameter/parameter.env.inc	The configuration settings what would be excluded from your version Control (copy of `/inc/parameter/parameter.env.example.inc`)
9	/app/inc/site.config.php	The custom app-specific configuration settings and the configration settings from `/inc/config.php` can also be overidden here
10	/app/inc/route.config.php	The custom app-specific route configuration settings
11	/app/inc/constants.php	The app-specific constants

There may be additional configuration files if you have sub-sites or modules in your application. Let's say, if you have admin module in your application, there could be the following configuration files in the admin directory.

1. `/app/admin/inc/site.config.php`
2. `/app/admin/inc/route.config.php`
3. `/app/admin/inc/constants.php`

	File	Description
1	/app/inc/site.config.php	The configuration settings for admin and the settings from `/inc/config.php` can also be overidden here
2	/app/admin/inc/route.config.php	You can define admin routes here
3	/app/admin/inc/constants.php	You can define constants for admin here

Note that it works only when you define "admin" in `$lc_sites` in `/inc/config.php`.

```
$lc_sites = array(
    /* 'virtual_folder_name (namespace)'  => 'path/to/physical_folder_name_directly_
↪under_app_directory' */
    'admin' => 'admin'
);
```

Let's say, you have http://example.com and http://example.com/admin

When you access http://example.com, these config files are included in the following priority:

1. /inc/config.php
2. /app/inc/site.config.php

When you access http://example.com/admin, these config files are included in the following priority:

1. /inc/config.php
2. /app/inc/site.config.php
3. /app/admin/inc/site.config.php

When you access http://example.com, these route config files are included in the following priority:

1. /inc/route.config.php
2. /app/inc/route.config.php

When you access http://example.com/admin, these config files are included in the following priority:

1. /inc/route.config.php
2. /app/inc/route.config.php
3. /app/admin/inc/route.config.php

When you access http://example.com, these constant files are included in the following priority:

1. /inc/constants.php
2. /app/inc/constants.php

When you access http://example.com/admin, these constant files are included in the following priority:

1. /inc/constants.php
2. /app/inc/constants.php
3. /app/admin/inc/constants.php

CHAPTER 5

The View

As of version 3.0.0, PHPLucidFrame added a default global View object for managing your layout file, rendering variables to your views, including head scripts/styles, loading your view templates. The View object is available by calling `_app('view')`.

5.1 Creating View

A view is a visual output representation to user and is simply a web page, or a page fragment. You can create a view by placing a file `view.php` in a particular page directory, for example, a single post page `http://www.example.com/post/{id}` or `http://localhost/acme/post/{id}` may look like the below directory structure:

```
/app
    /post
        |-- index.php
        |-- view.php <--
```

The `view.php` generally should contain HTML between `<body>` and `</body>`. This may include header and footer fragments. But the header and footer may be also put into the layout file. See *Layout File* section.

5.2 Passing Data To view

You can pass data to your view by using the `addData()` method of the View object. You can get the View object using `$view = _app('view')` and set data using `$view->addData('name', $value)` in a particular `index.php`. For example, `/app/post/index.php` may look like this

```
$id = _get('id');
$view = _app('view');

$post = db_findOrFail('post', $id);
```

(continues on next page)

(continued from previous page)

```
_app('title', $blog->title);

$view->addData('pageTitle', $post->title); // This will be available as $pageTitle in␣
↪view
$view->addData('post', $post); // This will be available as $post in view
```

Alternatively, you can pass an array of data to view by directly assigning to the `data` property of the View object.

```
$view->data = array(
    'pageTitle' => $post->title,
    'post'      => $post,
);
```

5.3 Nested Views

Views may also be nested. You can include another view in a view. Let's say for example, you have a view (fragement) to show recent posts in single post page.

```
/app
    /post
        |-- index.php
        |-- recent-posts.php <--
        |-- view.php
```

You can include `recent-posts.php` in `view.php` like this

```
<?php _app('view')->block('recent-posts') ?>
```

If `recent-posts.php` is needed to include in more than one page, you can move the file into `/app/inc/tpl/` and `_app('view')->block('recent-posts')` will automatically look for the file in that directory when it is not found in the current directory.

A new option to return html from the `block()` method is added since version 3.1. You can provide thrid parameter to the method.

```
<?php
$html = _app('view')->block('recent-posts', $data, true)
echo $html;
?>
```

5.4 Layout File

Since verion 3.0, layout mode is enabled by default (`$lc_layoutMode = true` in `/inc/config.php`). The default layout file is configured as `$lc_layoutName = 'layout'` which points to `app/inc/tpl/layout.php`.

Basically a layout file would have the below structure.

```
<!DOCTYPE html>
<html>
```

(continues on next page)

(continued from previous page)

```
<head>
    <title><?php echo _title() ?></title>
    <link rel="canonical" href="<?php echo _canonical() ?>" />
    <?php _hreflang() ?>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <?php _metaSeoTags() ?>
    <link rel="shortcut icon" href="<?php echo _img('favicon.ico'); ?>" type="image/x-
↪icon" />
    <?php _css('base.css') ?>
    <?php _css('base.' . _lang() . '.css') ?>  <!-- you may not need this if your␣
↪site doesn't have multi-languages ->
    <?php _css('responsive.css') ?>
    <?php _css('jquery.ui') ?>
    <?php _app('view')->headStyle() ?> <!-- styles added by a particular page ->
    <?php _js('jquery') ?>
    <?php _js('jquery.ui') ?>
    <?php _script() ?> <!-- this is required for global JS variables -->
    <?php _js('LC.js') ?>
    <?php _app('view')->headScript() ?>  <!-- scripts added by a particular page ->
    <?php _js('app.js') ?>
</head>
<body>
    <div>
        <header>
            <!-- your header stuffs -->
        </header>
        <section>
            <?php _app('view')->load() ?> <!-- This injects a particular view␣
↪template here -->
        </section>
        </footer>
            <!-- your footer stuffs -->
        </footer>
    </div>
</body>
</html>
```

Note:

- You can check `/app/inc/tpl/layout.php`.

You may have a separate layout file for a particular page, let's say for example, you have a login page which have a different layout other than the rest pages of the site. You can create a new layout file `/app/inc/tpl/layout_login.php`.

You can set the new layout name for login page in `/app/login/index.php` such as

```
_app('view')->layout = 'layout_login';
```

Then, the login page will use `layout_login.php` whereas the other pages use `layout.php`.

5.5 Stylesheets & Scripts In Head

You may include stylesheets and scripts for a particular page rather than globally including in the layout file. Then you can use `addHeadStyle()` and `addHeadScript()` of the View object in `index.php`

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

# If locally stored files
$view->addHeadStyle('select2.min.css');  // app/assets/css/select2.min.css or assets/
↪css/select2.min.css
$view->addHeadScript('select2.min.css'); // app/assets/js/select2.min.css or assets/
↪js/select2.min.css

# If CDN
$view->addHeadStyle('https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/css/select2.
↪min.css');
$view->addHeadScript('https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/js/select2.
↪min.js');
```

Alternatively, you can use the helper functions - `_addHeadStyle()` and `_addHeadScript()`.

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

# If locally stored files
_addHeadStyle('select2.min.css');  // app/assets/css/select2.min.css or assets/css/
↪select2.min.css
_addHeadScript('select2.min.css'); // app/assets/js/select2.min.css or assets/js/
↪select2.min.css

# If CDN
_addHeadStyle('https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/css/select2.min.
↪css');
_addHeadScript('https://cdn.jsdelivr.net/npm/select2@4.1.0-rc.0/dist/js/select2.min.js
↪');
```

CHAPTER 6

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 7

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
THIRD_PARTY	File system path to the directory third-party 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

Note:

- You can also define your own constants in `/inc/constants.php` or `/app/inc/constants.php`.

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 8

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`
 */
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
```

8.1 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`
```

(continues on next page)

(continued from previous page)

```
 */
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 `_get('id')` and `_get('slug')` in `/app/blog/show/index.php`.

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

8.2 Route Groups

PHPLucidFrame 2.0 supports route groups using prefix which allows you to prepend a URI prefix to a large number of routes. In the following example, the URI prefix `/api/posts` is added to all routes defined inside `route_group()`:

```
route_group('/api/posts', function () {
    route('lc_post')->map('/', '/example/api/post', 'GET');
    route('lc_post_create')->map('/', '/example/api/post/create', 'POST');
    route('lc_post_update')->map('/{id}', '/example/api/post/update', 'PUT', array('id
↪' => '\d+'));
    route('lc_post_delete')->map('/{id}', '/example/api/post/delete', 'DELETE', array(
↪'id' => '\d+'));
});
```

The above route groups definition is equal to these individual route definitions:

```
route('lc_post')->map('/api/posts', '/example/api/post', 'GET');
route('lc_post_create')->map('/api/posts', '/example/api/post/create', 'POST');
route('lc_post_update')->map('/api/posts/{id}', '/example/api/post/update', 'PUT',␣
↪array('id' => '\d+'));
route('lc_post_delete')->map('/api/posts/{id}', '/example/api/post/delete', 'DELETE',␣
↪array('id' => '\d+'));
```

Note:

- You can define your custom routes in `/inc/route.config.php` or `/app/inc/route.config.php`

8.3 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()`.

```
// 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
// 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.
```

8.4 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
```

When you have a custom route defined in `/inc/route.config.php` as described above at *Custom Routes*, you can use the route name as below:

```
_url('lc_blog_show', array('id' => 1, 'slug' => 'hello-world'))
// http://www.example.com/blog/1/hello-world
```

8.5 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')

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

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

8.6 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 `_get()` or `_arg()`:

```
$id = _get('id'); // or _arg('id')
$slug = _get('slug'); // or _arg('slug')
```

CHAPTER 9

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('app.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/app.js
2. /app/assets/js/app.js
3. /assets/js/app.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 10

Auto-loading Libraries

There are two ways of auto-loading third-party libraries:

1. Using Composer
2. Using Custom Autoloader

10.1 Composer

Composer support is automatically initialized by default. It looks for Composer's autoload file at `/vendor/autoload.php`. You just need to add libraries to `composer.json`.

10.2 Custom Autoloader

If you are not using composer, you can update `/inc/autoload.php` to load a library. The following is a few steps to do:

1. Download any third-party library
2. Put it in the folder `/third-party`
3. Load the file using the helper function `_loader()` in `/inc/autoload.php`

Let's say for example, you are trying to integrate the library PHP-JWT.

1. Download the library from downloading the library from https://github.com/firebase/php-jwt/tags.
2. Unzip and put the folder in `/third-party` as `/third-party/php-jwt-x.y.z` (`x.y.z` is the library version number you downloaded). However, you can also rename it as `/third-party/php-jwt`.
3. Add `_loader('JWT', THIRD_PARTY . 'php-jwt-x.y.z/src/');` in `/inc/autoload.php`. This will load `/third-party/php-jwt-x.y.z/src/JWT.php`.
4. Then, you can try the following code.

```
use Firebase\\JWT\\JWT;

$key = "example_key";
$payload = array(
    "iss" => "http://example.org",
    "aud" => "http://example.com",
    "iat" => 1356999524,
    "nbf" => 1357000000
);

/**
* IMPORTANT:
* You must specify supported algorithms for your application. See
* https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-40
* for a list of spec-compliant algorithms.
*/
$jwt = JWT::encode($payload, $key);
_pr($jwt);

$decoded = JWT::decode($jwt, $key, array('HS256'));
_pr($decoded);

/*
NOTE: This will now be an object instead of an associative array. To get
an associative array, you will need to cast it as such:
*/
$decoded_array = (array) $decoded;
_dpr($decoded_array);
```

CHAPTER 11

Database Configuration

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

1. `/inc/parameter/development.php` for development environment
2. `/inc/parameter/staging.php` for staging environment
3. `/inc/parameter/production.php` for production environment
4. `/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
        )
    )
    // ...
);
```

11.1 Make Your Credentials Secret

As of version 2.0, PHPLucidFrame includes a file `/inc/parameter/parameter.env.example.inc`. You can copy and rename it to `parameter.env.inc` which is already ignored from version control. So, in the file, you can define your important information that needs to be secret and to not share with others. For example, you can define your production database credentials in `/inc/parameter/parameter.env.inc` like below:

```
return array(
    'prod' => array( # either prod or production as you like
        'db' => array(
            'default' => array(
                'database'  => 'your_prod_db',
                'username'  => 'your_prod_username',
                'password'  => 'your_prod_pwd',
                'prefix'    => '',
            )
        )
    )
);
```

then, you can call those parameters from `/inc/parameter/production.php` using `_env('prod.db.default.xxxx')`

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

11.2 Connecting to Multiple Databases

Sometimes, we need to connect multiple databases in our app. As an example, you might have two databases, the default database and a legacy database. The configuration in `/inc/parameter/development.php` or `/inc/parameter/production.php` for your two databases would be as below:

```
return array(
    // ...
    # Database connection information
    'db' => array(
        'default' => array(
            'driver'    => 'mysql',
            'host'      => 'localhost',
            'port'      => '',
            'database'  => 'your_db_name',
            'username'  => 'your_db_username',
            'password'  => 'your_db_pwd',
            'prefix'    => '',
            'charset'   => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
```

(continues on next page)

(continued from previous page)

```
            'engine'    => 'InnoDB',
        ),
        'legacy' => array(
            'driver'    => 'mysql',
            'host'      => 'localhost',
            'port'      => '',
            'database'  => 'legacy_db_name',
            'username'  => 'legacy_db_username',
            'password'  => 'legacy_db_pwd',
            'prefix'    => '',
            'charset'   => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
            'engine'    => 'InnoDB',
        )
    ),
    // ...
);
```

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');
```

11.3 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 */
```

(continues on next page)

(continued from previous page)

```
    )
);
```

CHAPTER 12

Working With Data

To insert, update, delete data, PHPLucidFrame provides the helper functions - `db_insert()`, `db_update()`, `db_delete()` and `db_delete_multi()`.

12.1 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(
    'title' => 'New Title', // this will be used for the slug field while third␣
↪argument is true
    'body' => '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,
    'title' => 'Updated Title',
    'body' => 'Updated post complete description here'
));
```

- db_insertId() which returns the auto generated id used in the last query.

- 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.

12.2 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, array $condition = array())
```

For example,

```
$success = db_update('post', array(
    'id'    => 1, // The first field/value pair will be used as condition when you do␣
↪not provide the fourth argument
    'title' => 'Updated Title', // this will be used for the slug field while third␣
↪parameter is true
    'body'  => 'Updated post complete description here'
));

// # Generated query
// UPDATE post SET
//   slug = "updated-title",
//   title = "Updated Title",
//   body = "Updated post complete description here
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1

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

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

```
$success = db_update('post', array(
    'id'    => 1, // The first field/value pair will be used as condition when you do␣
↪not provide the fourth argument
    'slug'  => 'custom-updated-title', // providing custom slug string
    'title' => 'Updated Title',
    'body'  => 'Updated post complete description here'
));

// # Generated query
// UPDATE post SET
//   slug = "custom-updated-title",
//   title = "Updated Title",
//   body = "Updated post complete description here
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1
```

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

```
db_update('post',
    array(
        'title' => 'Updated Title', // this will be used for slug as well
        'body'  => 'Updated post complete description here'
    ),
    array('id' => 1) // condition for update as third parameter
);

// # Generated query
// UPDATE post SET
//   slug = "updated-title",
//   title = "Updated Title",
//   body = "Updated post complete description here
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1
```

As fouth parameter,

```
db_update('post',
    array(
        'title' => 'Updated Title',
        'body'  => 'Updated post complete description here'
    ),
    false, // To not update the slug field
    array('id' => 1) // condition for update as fouth parameter
);

// # Generated query
// UPDATE post SET
//   title = "Updated Title",
//   body = "Updated post complete description here
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1
```

If you want to update the records by OR condition, use `$or` operator as key in the condition array.

```
db_update('post',
    array(
        'status' => 'active'
    ),
    array(
        '$or' => array(
            'created <=' => date('Y-m-d H:i:s'),
            'created >=' => date('Y-m-d H:i:s'),
        )
    )
);

// # Generated query
// UPDATE post SET
//   status = 'active'
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE created <= 'xxxx-xx-xx xx:xx:xx' OR created >= 'xxxx-xx-xx xx:xx:xx'
```

12.3 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', array $condition = array(), $softDelete = false)
```

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('id' => $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(
    'field_name1'    => $value1,
    'field_name2 >=' => $value2,
    'field_name3'    => null,
))
```

If you want to delete the records by OR condition, use `$or` operator as key in the condition array.

```
db_delete_multi('table_name', $condition = array(
    '$or' => array(
        'field_name1'    => $value1,
        'field_name2 >=' => $value2,
        'field_name3'    => null,
    )
))
```

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

12.4 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 use `$and` and `$or` operators as key in the condition array. The following are some examples.

Updating with simple condition:

```
db_update('post', array(
    'title' => 'Updated Title',
), array(
    'id' => 1
));

// # Generated query
// UPDATE post SET
//   slug = "updated-title",
//   title = "Updated Title",
```

(continues on next page)

(continued from previous page)

```
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1
```

Updating using AND condition:

```
db_update('post', array(
        'cat_id' => 1 // The field to be updated
    ),
    false, // To not update the slug field
    array(
        'id' => 1,
        'delete !=' => NULL
    )
);

// # Generated query
// UPDATE post SET
//   cat_id = 1,
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1 AND deleted IS NOT NULL
```

Updating using IN condition:

```
db_update('post', array(
        'cat_id' => 1 // The field to be updated
    ),
    false, // To not update the slug field
    array(
        'id' => array(1, 2, 3)
    ))
);

// # Generated query
// UPDATE post SET
//   cat_id = 1,
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id IN (1, 2, 3)
```

Updating using OR condition:

```
db_update('post', array(
        'cat_id' => 1 // The field to be updated
    ),
    false, // To not update the slug field
    array(
        '$or' => array(
            array('id' => 1),
            array('id' => 2)
        )
    )
);

// # Generated query
// UPDATE post SET
//   cat_id = 1,
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id = 1 OR id = 2
```

Updating using IN and OR condition:

```
db_update('post', array(
        'cat_id' => 1 // The field to be updated
    ),
    false, // To not update the slug field
    array(
        '$or' => array(
            'id' => array(1, 2, 3),
            'id >' => 10,
        )
    )
);

// # Generated query
// UPDATE post SET
//   cat_id = 1,
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE id IN (1, 2, 3) OR id > 10
```

Updating with complex AND/OR condition:

```
db_update('post', array(
        'cat_id' => 1 // The field to be updated
    ),
    false, // To not update the slug field
    array(
        'title' => 'a project',
        'cat_id' => 2,
        '$or' => array(
            'id' => array(1, 2, 3),
            'id >=' => 10,
        )
    )
);

// # Generated query
// UPDATE post SET
//   cat_id = 1,
//   updated = "xxxx-xx-xx xx:xx:xx"
// WHERE title = "a project"
// AND cat_id= 2
// AND ( id IN (1, 2, 3) OR id >= 10 )
```

12.5 Condition Operators

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

12.6 Finding Data

A couple of quick helpful functions are provided for fetching data from tables.

12.6.1 db_find()

Returns a single entity result where the primary key matches the value passed in as the second parameter for the table name in the first parameter.

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`id`	int	The value of the primary key to match
‘‘fields‘	array	The array of fields to select

Example:

```
$result = db_find('post', 1);
// or
$result = db_find('post', 1, ['id', 'title']);
```

Note: See more at http://www.phplucidframe.com/api-doc/latest/function-db_find.html.

12.6.2 db_findOrFail()

Returns a single entity result where the primary key matches the value passed in as the second parameter for the table name in the first parameter OR throws 404 if any result is not found.

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`id`	int	The value of the primary key to match
‘‘fields‘	array	The array of fields to select

Example:

```
$result = db_findOrFail('post', 1);
// or
$result = db_findOrFail('post', 1, ['id', 'title']);
```

Note: See more at http://www.phplucidframe.com/api-doc/latest/function-db_findOrFail.html.

12.6.3 db_findAll()

Returns all rows from the given table.

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`fields`	array	The list of the field names to select (optional)
`orderBy`	array	The order by clause for query (optional)

Example:

```
$result = db_findAll('post', array('id', 'cat_id', 'title'));
```

Note:

- This function is added in the version 3.1.0.
- See more at http://www.phplucidframe.com/api-doc/latest/function-db_findAll.html.

12.6.4 db_findBy()

Returns array of data row objects of a table by condition.

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`condition`	array	The condition array for query (optional)
`orderBy`	array	The order by clause for query (optional)
`limit`	int	The number of records to return; No limit by default

Example:

```
$result = db_findBy('post', array('cat_id' => 1), array('created' => 'desc'), 3);
```

Note:

- This function is added in the version 3.1.0.
- See more at http://www.phplucidframe.com/api-doc/latest/function-db_findBy.html.

12.6.5 db_findOneBy()

Returns a single entity result of a table by condition

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`condition`	array	The condition array for query (optional)
`orderBy`	array	The order by clause for query (optional)

Example:

```
$result = db_findOneBy('post', array('cat_id' => 1), array('created' => 'desc'));
```

Note:

- This function is added in the version 3.1.0.
- See more at http://www.phplucidframe.com/api-doc/latest/function-db_findOneBy.html.

12.6.6 db_findOneByOrFail()

Returns a single entiry result of a table by condition or throw 404 if not found

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`condition`	array	The condition array for query (optional)
`orderBy`	array	The order by clause for query (optional)

Example:

```
$result = db_findOneBy('post', array('cat_id' => 1), array('created' => 'desc'));
```

Note:

- This function is added in the version 3.1.0.
- See more at http://www.phplucidframe.com/api-doc/latest/function-db_findOneByOrFail.html.

12.6.7 db_findWithPager()

Returns array of data row objects with pagination result.

Parameters:

Name	Type	Description
`table`	string	The table name to fetch data from
`condition`	array	The condition array for query (optional)
`orderBy`	array	The order by clause for query (optional)
`pagerOption`	array	Array of key/value pairs to Pager options (optional)

Return:

Array of three items:

1. QueryBuilder - An instance of `LucidFrame\Core\QueryBuilder`
2. Pager - An instance of `LucidFrame\Core\Pager`
3. int - The total number of records

Example:

```
list($qb, $pager, $total) = db_findWithPager('post', array('cat_id' => 1), array(
↪'created' => 'desc'));
```

Note: See more at http://www.phplucidframe.com/api-doc/latest/function-db_findWithPager.html.

CHAPTER 13

Query Builder

As of version 1.9, PHPLucidFrame added a new feature called query builder that allows data to be retrieved in your database without writing raw SQL statements.

13.1 Selecting Data for Multiple Results

If you want to fetch an array of result set, you can use `getResult()`.

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

This generates the following query:

```
SELECT * FROM `post`
```

13.2 Selecting Data for Single Result

To get a single result set, you can use `getSingleResult()`.

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

This generates the following query:

```
SELECT * FROM `post` LIMIT 1
```

13.3 Selecting Data for Muliple Fields

To fetch multiple fields, you can use `fields('table`, array('field1', 'field2', ...))`. The first parameter is table name or alias. The second paramter is a list of field names to fetch from the table.

```
$result = db_select('post', 'p')
    ->fields('p', array('id', 'title', 'created'))
    ->getResult();

_pr($result); // array of results
```

This generates the following query:

```
SELECT `p`.`id`, `p`.`title`, `p`.`created` FROM `post` `p`
```

If you want field alias, you can use nested array in `fields()`, for example,

```
$result = db_select('post', 'p')
    ->fields('p', array('id', array('title', 'title'), 'created'))
    ->getResult();

_pr($result); // array of results
```

In this case, `post_title` is alias for `title`. This generates the following query:

```
SELECT `p`.`id`, `p`.`title` `title`, `p`.`created` FROM `post` `p`
```

13.4 Selecting Data for Single Field

To fetch a single field, you can use `field('field_name')` and then `fetch()`.

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

echo $title;
```

This generates the following query:

```
SELECT `p`.`title` FROM `post`
```

13.5 Joining Tables

If you want to join multiple tables, you can use `join($table, $alias, $condition, $type = 'INNER')`. Here is explanation of the list of arguments:

- `$table` is the table name to join.
- `$alias` is the alias for the table name and you can also set `null` for this.
- `$condition` is the joining condition e.g., `table1.pk_id = table2.fk_id`.
- `$type` is the join type. Available options are `INNER`, `LEFT`, `RIGHT`, `OUTER`. Default is `INNER`.

```
$result = db_select('post', 'p')
    ->fields('p', array('id', 'title'))
    ->fields('u', array(array('name', 'author')))
    ->fields('c', array(array('name', 'categoryName')))
    ->join('user', 'u', 'p.uid = u.uid')
    ->leftJoin('category', 'c', 'p.cat_id = c.cat_id')
    ->getResult();
_pr($result); // array of results
```

It generates the following query:

```
SELECT `p`.`id`, `p`.`title`, `u`.`name` `author`, `c`.`name` `categoryName`
FROM `post` `p`
INNER JOIN `user` `u` ON `p`.`uid` = `u`.`uid`
LEFT JOIN `category` `c` ON `p`.`id` = `c`.`id`
```

Note:

- Instead of fourth parameter to `join()`, you could also use the individual methods - `leftJoin()`, `rightJoin()` and `outerJoin()`.

13.6 Fetching Specific Data (WHERE condition)

There are some methods available to create query conditions - `where()`, `andWhere()`, `orWhere()` and `condition()`. `where()` is an alias of `andWhere()`. You can use `where()`, `andWhere()` and `orWhere()` with array parameter or without parameter.

13.6.1 Simple condition

For array parameter, it accepts all conditional operators described in the previous section, for example,

```
$result = db_select('post', 'p')
    ->fields('p', array('id', 'title'))
    ->fields('u', array(array('name', 'author')))
    ->fields('c', array(array('name', 'categoryName')))
    ->join('user', 'u', 'p.user_id = u.id')
    ->leftJoin('category', 'c', 'p.cat_id = c.id')
    ->where(array(
        'c.id' => 1,
        'u.id' => 2
    ))
    ->getResult();
```

Without parameter, it initializes to create conditions by using `condition()`:

```
$result = db_select('post', 'p')
    ->fields('p', array('id', 'title'))
    ->fields('u', array(array('name', 'author')))
    ->fields('c', array(array('name', 'categoryName')))
    ->join('user', 'u', 'p.user_id = u.id')
    ->leftJoin('category', 'c', 'p.cat_id = c.id')
    ->where()
```

(continues on next page)

(continued from previous page)

```
        ->condition('c.id', 1)
        ->condition('u.id', 2)
    ->getResult();
```

The above two queries would generate the following same query:

```
SELECT `p`.`id`, `p`.`title`, `u`.`name` `author`, `c`.`name` `categoryName`
FROM `post` `p`
INNER JOIN `user` `u` ON `p`.`user_id` = `u`.`id`
LEFT JOIN `category` `c` ON `p`.`cat_id` = `c`.`id`
WHERE `c`.`id` = 1
AND `u`.`id` = 2
```

13.6.2 Complex condition using AND/OR

You can use the operator keys, `$and` and `$or`, for complex conditions. Here is an exmaple:

```
$result = db_select('post', 'p')
    ->fields('p')
    ->fields('u', array('username', array('name', 'author')))
    ->join('user', 'u', 'p.user_id = u.id')
    ->leftJoin('category', 'c', 'p.cat_id = c.id')
    ->where(array(
        'title like' => 'Sample project',
        '$or' => array(
            'p.id' => array(1, 2, 3),
            'u.id' => 1
        )
    ))
    ->orderBy('p.created', 'desc')
    ->limit(0, 20)
    ->getResult();
```

It generates the following query:

```
SELECT `p`.*, `u`.`username`, `u`.`name` `author`
FROM `post` `p`
INNER JOIN `user` `u` ON `p`.`user_id` = `u`.`id`
LEFT JOIN `category` `c` ON `p`.`cat_id` = `c`.`id`
WHERE `p`.`title` LIKE "%Sample project%"
AND ( `p`.`id` IN (1, 2, 3) OR `u`.`id` = 1 )
ORDER BY `p`.`created` DESC
LIMIT 0, 20
```

13.6.3 Complex nested condition using OR/AND/OR

The following is an example for complex nested conditions using AND/OR:

```
$result = db_select('post', 'p')
    ->fields('p')
    ->fields('u', array('username', array('name', 'author')))
    ->join('user', 'u', 'p.user_id = u.id')
    ->leftJoin('category', 'c', 'p.cat_id = c.id')
```

(continues on next page)

(continued from previous page)

```
    ->orWhere(array(
        'p.title nlike' => 'Sample project',
        '$and' => array(
            'p.id' => array(1, 2, 3),
            'p.status <=' => 10,
            '$or' => array(
                'p.created >' => '2020-12-31',
                'p.deleted' => null
            )
        )
    ))
    ->orderBy('p.created', 'desc')
    ->limit(5)
    ->getResult()
```

It generates the following query:

```
SELECT `p`.*, `u`.`username`, `u`.`name` `author`
FROM `post` `p`
INNER JOIN `user` `u` ON `p`.`user_id` = `u`.`id`
LEFT JOIN `category` `c` ON `p`.`cat_id` = `c`.`id`
WHERE `p`.`title` NOT LIKE "%Sample project%"
OR (
    `p`.`id` IN (1, 2, 3)
    AND `p`.`status` <= 10
    AND ( `p`.`created` > "2020-12-31" OR `p`.`deleted` IS NULL )
)
ORDER BY `p`.`created` DESC
LIMIT 5
```

13.6.4 EXITS and NOT EXISTS

As of version 3.2, PHPLucidFrame added support for `EXISTS` and `NOT EXISTS` conditions. Here is an example.

```
$subquery = db_select('post_to_tag', 'pt')
    ->where()
    ->condition('post_id', db_raw('p.id'))
    ->condition('tag_id', 1)
    ->getReadySQL();

$qb = db_select('post', 'p')
    ->where()
    ->condition('deleted', null)
    ->exists($subquery);

$result = $qb->getResult();
```

It generates the following query:

```
SELECT `p`.* FROM `post` `p` WHERE `deleted` IS NULL
AND EXISTS (SELECT `pt`.* FROM `post_to_tag` `pt` WHERE `post_id` = `p`.`id` AND `tag_
↪id` = 1)
```

You can also use `notExists()` for `NOT EXISTS`.

```
$subquery = db_select('post_to_tag', 'pt')
    ->where()
    ->condition('post_id', db_raw('p.id'))
    ->condition('tag_id', 1)
    ->getReadySQL();

$qb = db_select('post', 'p')
    ->where()
    ->condition('deleted', null)
    ->notExists($subquery);

$result = $qb->getResult();
```

`orExists()` and `orNotExists()` are also available to add multiple `OR EXISTS` or `OR NOT EXISTS` statements to your query.

13.7 Grouping Results

You can use `groupBy()` to write the GROUP BY portion of your query:

```
$result = db_select('post', 'p')
    ->groupBy('p.cat_id')
    ->getResult();
```

You can use multiple `groupBy()` calls. This generates the following query:

```
SELECT `p`.* FROM `post` `p`
GROUP BY `p`.`cat_id`
```

13.8 HAVING Condition on Group Result

There are some methods available to create having conditions - `having()`, `andHaving()`, `orHaving()`. `having()` is an alias of `andHaving()`. You can use them with array parameter of conditional operators described in the previous section, for example,

```
$result = db_select('post', 'p')
    ->groupBy('p.cat_id')
    ->having(array(
        'p.cat_id >' => 10,
        'p.status' => 1
    ))
    ->getResult();
```

This generates the following query:

```
SELECT `p`.* FROM `post` `p`
GROUP BY `p`.`cat_id`
HAVING `p`.`cat_id` > 10 AND `p`.`status` = 1
```

You can create OR condition on having using `orHaving()` like this:

```
$result = db_select('post', 'p')
    ->groupBy('p.cat_id')
    ->orHaving(array(
        'p.cat_id >' => 10,
        'p.status' => 1
    ))
    ->getResult();
```

13.9 Ordering Results

You can use `orderBy('field', 'asc|desc')`. The first parameter contains the name of the field you would like to order by. The second parameter lets you set the direction of the result. Options are `asc` and `desc`. Default to `asc`.:

```
$result = db_select('post', 'p')
    ->fields('p', array('id', 'title', 'created'))
    ->orderBy('p.title', 'asc)
    ->orderBy('p.created', 'desc')
    ->getResult();

_pr($result); // array of results
```

This generates the following query:

```
SELECT `p`.`id`, `p`.`title`, `p`.`created` FROM `post` `p`
ORDER BY `p`.`title` ASC, `p`.`created` DESC
```

13.10 Counting Results

`db_count()` lets you determine the number of rows in a particular table.

```
$rowCount = db_count('post')
    ->where()->condition('deleted', null)
    ->fetch();

echo $rowCount;
```

This generates the following query:

```
SELECT COUNT(*) count FROM `post` WHERE deleted IS NULL
```

13.11 Limiting Results

`limit()` permits to limit the number of rows you would like returned by the query:

```
$result = db_select('post')
    ->limit(10)
    ->getResult();
_pr($result); // array of results
```

This generates the following query to return the first 10 records from the table `post`:

```
SELECT * FROM `post` LIMIT 10
```

You can also set offset to `limit()`:

```
$result = db_select('post')
    ->limit(0, 10)
    ->getResult();
```

The following query will be executed:

```
SELECT * FROM `post` LIMIT 0, 10
```

13.12 Aggregates

There are aggregate functions available - `db_min()`, `db_max()`, `db_sum()`, `db_avg()`.

13.12.1 MAX

Syntax: `db_max($table, $field, $alias = null)`

```
$max = db_max('post', 'view_count')->fetch();
// SELECT MAX(`view_count`) max FROM `post` `post`
```

13.12.2 MIN

Syntax: `db_min($table, $field, $alias = null)`

```
$min = db_min('post', 'view_count')->fetch();
// SELECT MIN(`view_count`) min FROM `post` `post`
```

13.12.3 SUM

Syntax: `db_sum($table, $field, $alias = null)`

```
$sum = db_sum('post', 'view_count')->fetch();
// SELECT SUM(`view_count`) sum FROM `post` `post`
```

13.12.4 AVG

Syntax: `db_avg($table, $field, $alias = null)`

```
$sum = db_avg('post', 'view_count')->fetch();
// SELECT SUM(`view_count`) avg FROM `post` `post`
```

13.12.5 Aggregate functions together

You can use aggregate function together like below:

```
$result = db_select('post', 'p')
    ->max('view_count', 'max')
    ->min('view_count', 'min')
    ->getResult();
```

This generates:

```
SELECT MAX(`view_count`) max, MIN(`view_count`) min FROM `post` `p`
```

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 14

Native Queries

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

- db_query() which peforms a query on the database.
- db_fetchAssoc() which fetchs a result row as an associate array.
- db_fetchArray() which fetchs a result row as a numeric array.
- db_fetchObject() which fetchs a result row as an object.
- db_fetch() which performs a query on the database and return the first field value only.
- db_fetchResult() which performa a query on the database and return the first result row as object.
- db_extract() which performs a query on the database and return the array of all results.

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:

```
$sql = 'SELECT * FROM ' . db_table('post') . ' ORDER BY title';
if ($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 title';
$posts = db_extract($sql); // The second or third argument can be given one of these:␣
↪LC_FETCH_OBJECT (default), LC_FETCH_ASSOC, LC_FETCH_ARRAY
_pr($posts);
```

(continues on next page)

(continued from previous page)

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

The following is an example to retrieve a single result:

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

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

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

CHAPTER 15

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.

15.1 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/
),
```

15.2 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('cat_id'),     // 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' => 'cat_id', // 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),
    'title'     => array('type' => 'string', 'null' => false),
    'title_en'  => array('type' => 'string', 'null' => true),
    'title_my'  => array('type' => 'string', 'null' => true),
    'body'      => array('type' => 'text', 'null' => false),
    'body_en'   => array('type' => 'text', 'null' => true),
    'body_my'   => array('type' => 'text', 'null' => true),
    'options' => array(
        'Pk' => array('id'), // 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'      => 'id',
            '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'      => 'id',
            'cascade'   => true,
```

(continues on next page)

(continued from previous page)

```
        ),
    ),
),
```

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`
`options.unique`		Unique index for composite fields `array('keyName' => array('field_name1', 'field_name2'))`
`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

Continued on next page

Table 1 – continued from previous page

Name	Default	Explanation
`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
`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`

15.3 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	2-byte integer values: • Unsigned integer with a range of **0** to **65535** • Signed integer with a range of **32768** to **32767**
mediumint	MEDIUMINT	3-byte integer values: • Unsigned integer with a range of **0** to **16777215** • Signed integer with a range of **8388608** to **8388607**
int/integer	INT	4-byte integer values: • Unsigned integer with a range of **0** to **4294967295** • Signed integer with a range of **2147483648** to **2147483647**
bigint	BIGINT	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)	A 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)	A 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	String data with a maximum length specified
char	CHAR	String data with a fixed length specified
binary	VARBINARY	Binary string data with a maximum length specified
tinytext	TINYTEXT	String data with a maximum length of 255 characters.
text	TEXT	String data with a maximum length of 6,553 characters.
mediumtext	MEDIUMTEXT	String data with a maximum length of 16,777,215 characters.
longtext	LONGTEXT	String data with a maximum length of 4,294,967,295 characters.
tinyblob	TINYBLOB	String data with a maximum length of 6,553 characters.
blob	BLOB	Binary string data with a maximum length of 6,553 characters.

15.4 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
```

15.5 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
```

15.6 Managing Schema Changes

As of version 2.2.0, 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(
    'facebook_url'  => array('type' => 'string', 'null' => true),
    'twitter_url'   => array('type' => 'string', 'null' => true),
    'instagram_url' => array('type' => 'string', 'null' => true),
    'linkedin_url'  => array('type' => 'string', '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,
        ),
    ),
),
```

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

```
$ php lucidframe schema:diff sample
PHPLucidFrame 3.3.0 by Sithu K.

```

(continues on next page)

(continued from previous page)

```
./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 3.3.0 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', 'null' => true),
    'twitter'   => array('type' => 'string', 'null' => true),
    'instagram' => array('type' => 'string', 'null' => true),
    'linkedin'  => array('type' => 'string', '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 3.3.0 by Sithu K.


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

Field renaming from `facebook_url` to `social_profile.facebook`: y
Field renaming from `twitter_url` to `social_profile.twitter`: y
Field renaming from `instagram_url` to `social_profile.instagrams`: y
Field renaming from `linkedin_url` to `social_profile.linkedin`: 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 3.3.0 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 16

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`.

16.1 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::getReference()

// 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::getReference('record-key-from-previously-executed-seed'),
    ),
    'record-2' => array(
        'field1'  => 'value2',
        'field2'  => 'value2',
        'field3'  => Seeder::getReference('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.

16.2 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',
        'name' => 'Technology',
    ),
    'category-2' => array(
        'slug' => 'framework',
        'name' => '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(
        'cat_id'    => Seeder::getReference('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',
        'title'     => 'Welcome to the LucidFrame Blog',
        'body'      => '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',
        'name' => 'PHP',
    ),
    'tag-2' => array(
        'slug' => 'mysql',
        'name' => '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(
        'post_id'   => Seeder::getReference('post-1'), // reference field to the␣
↪table "post"
        'tag_id'    => Seeder::getReference('tag-1'),  // reference field to the␣
↪table "tag"
    ),
    'post-to-tag-2' => array(
        'post_id'   => Seeder::getReference('post-1'),
        'tag_id'    => Seeder::getReference('tag-2'),
    ),
);
```

Note:

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

16.3 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 17

Middleware

As of version 2.0, PHPLucidFrame added middleware support. You can handle a HTTP request through middleware. You can define middlewares to execute before or after a HTTP request. For example, you can include a middleware that authenticates a user. If the user is not authenticated, the middleware will redirect the user to the login page, otherwise, it will allow the request to proceed. All middlewares should be located and defined in the `/app/middleware` directory.

17.1 Before Middleware

A *before* middleware is an event executed before a request. Here is its definition syntax:

```
_middleware(function () {
    // Do something before the page request
    // for example, checking bearer token from HTTP Authorization
    // $authorization = _requestHeader('Authorization')
});

_middleware(function () {
    // Do something before the page request
}, 'before');  // 'before' is optional and default
```

17.2 After Middleware

An *after* middleware is an event executed after a request. Here is its definition syntax:

```
_middleware(function () {
    // Do something at the end of the page request
}, 'after');
```

17.3 Order/Precedence for Middlewares

You can register precedence of the middleware execution by using the method `order()`, for example,

```
_middleware(function () {
    // ...
})->order(2);

_middleware(function () {
    // ...
})->order(1);
```

The middlewares will be executed in ascending order, so the second middleware will be executed first.

17.4 Assigning Middleware to Routes

A middleware is run during every HTTP request to your application by default. However, you can assign middleware to specific routes using the `on()` method. The first parameter to the `on()` method is filter name and the second parameter is any route paths or route names.

Filter Name	Description	Example
startWith	To run the middleware on the route starting with the given URI	`->on('startWith', 'api')`
contain	To run the middleware on the route containing the given URI	`->on('contain', 'api/posts')`
equal	To run the middleware on the route equal to the given route name or path	`->on('equal', 'lc_blog_show')`
except	To run the middleware on the routes except the given list	`->on('except', 'login')`

Note:

- You can check some example middlewares at https://github.com/phplucidframe/phplucidframe/blob/master/app/middleware/example.php

Let's create a middleware `/app/middleware/auth.php` that contains the following code:

```
<?php

/**
  * This is an example middleware running before every page request
  * that route starts with "admin"
  */
_middleware(function () {
    // set flash message and redirect to the login page if user is anonymous
    if (auth_isAnonymous()) {
        flash_set('You are not authenticated. Please log in.', '', 'error');
        _redirect('admin/login');
    }

    // otherwise, it will proceed the request
```

(continues on next page)

(continued from previous page)

```
})->on('startWith', 'admin') // executed on every page that URI starts with /admin
    ->on('except', array('admin/login')); // not be executed on /admin/login
```

It will be executed on every page that URI starts with `/admin` and it verifies the user is authenticated. It will not be executed on `/admin/login`. If the user is not authenticated, it will redirect to the login page.

CHAPTER 18

Form & Validation

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

18.1 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="your-form-name" id="your-form-id" 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="your-form-name" id="your-form-id" 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>
```

If you don't want to clear data in form inputs after submit, you can set `data-clear="off"` to the form tag.

```
<form name="your-form-name" id="your-form-id" data-clear="off" method="post">
```

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.

18.2 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="your-form-name" id="your-form-id" method="post" class="no-ajax">
    <div class="message error"></div>

    <!-- HTML input elements here as usual -->

    <?php form_token(); ?>
</form>
```

18.3 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 (_isHttpPost()) {
    $data = _post(); // Sanitize your inputs

    /** Form validation prerequisites here, for example */
    $validations = array(
        'title' => array(
            'caption'   => _t('Title'),
            'rules'     => array('mandatory'),
        ),
        'body' => array(
            'caption'   => _t('Body'),
            'rules'     => array('mandatory'),
        ),
    );

    if (form_validate($validations, $data)) {
        /**
        Database operations here
        */
```

(continues on next page)

(continued from previous page)

```
        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'));
    }
}

// Respond to the client
form_respond('your-form-id'); // 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('your-form-id', validation_get('errors'))` at the end of the form in `view.php` in order to show error messages correctly.

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

18.4 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. If it is not provided,␣
↪you need to provide all POST data to form_validate().
        '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'
```

(continues on next page)

(continued from previous page)

```
        'pattern'    => '', // The required property for the rule 'custom'
        'table'      => '', // The required property for the rule 'unique'
        'field'      => '', // The required property for the rule 'unique'
        'id'         => '', // The optional property for the rule 'unique'
        '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
            // 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($validations)) { // or validation_check($validations)
    // ...
}
```

You can also add POST data to the second parameter.

```
$data = _post();

$validations = array(
    // ...
);

if (form_validate($validations, $data)) {
    // ...
}
```

Note:

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

18.5 Sanitizing Form Inputs

You can sanitize form inputs using `_post()`.

```
if (_isHttpPost()) {
    $name = _post('name'); // $_POST['name']
```

(continues on next page)

(continued from previous page)

```
    $email = _post('email'); // $_POST['email']
}
```

You can also sanitize all inputs by calling `_post()` without parameter.

```
if (_isHttpPost()) {
    $post = _post(); // Array of $_POST will be sanitized
    // $post['name']
    // $post['email']
}
```

18.6 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.

18.6.1 alphaNumeric

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

18.6.2 alphaNumericDash

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

18.6.3 alphaNumericSpace

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

18.6.4 between

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

```
$validations = array(
    'vote' => array( // vote is HTML input element name or id
        'caption' => _t('Vote');
        'value'   => $valueToCheck,
        'rules'   => array('mandatory', 'between'),
        'max'     => 0,
        'max'     => 5,
    ),
); // The error message will be shown as "'Vote' should be between 0 and 5".
```

18.6.5 custom

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

```
$validations = array(
    'phone' => 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.
        ),
    ),
);
```

18.6.6 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(
    'date' => array(
        'caption'   => _t('Date');
        'value'     => $valueToCheck,
        'rules'     => array('date'),
        'dateFormat'=> 'd-m-y', // if not given, the default is y-m-d
    ),
);
```

18.6.7 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(
    'date' => array(
        'caption'   => _t('Date');
        'value'     => $valueToCheck,
        'rules'     => array('datetime'),
        'dateFormat'=> 'd-m-y', // if not given, the default is y-m-d
        'timeFormat'=> '24', // 12 or 24; if not given, default is both which␣
↪validates against both format
    ),
);
```

18.6.8 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.

```
$domain = array(
    'domain' => array(
        'caption'   => _t('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.".
```

18.6.9 email

This checks the field is a valid email address.

```
$validations = array(
    'email' => 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".
```

18.6.10 fileExtension

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

18.6.11 fileMaxSize

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

18.6.12 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(
    'logo' => 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',
    ),
);
```

18.6.13 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.

18.6.14 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.

18.6.15 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.

18.6.16 integer

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

18.6.17 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(
    'ip_addr' => array(
        'caption'  => _t('IP Address');
        'value'    => $valueToCheck,
        'rules'    => array('ip'),
        'protocol' => 'ipv4',
    ),
);
```

18.6.18 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(
    'name' => array(
        'caption'   => _t('Name');
        'value'     => $nameValueToCheck,
        'rules'     => array('mandatory'),
    ),
    'country' => array(
        'caption'   => _t('Country');
        'value'     => $countryValueToCheck,
        'rules'     => array('mandatory'),
        'messages'  => array(
            'mandatory' => _t('Country must be selected.') // this overwrites the␣
↪default message
        ),
```

(continues on next page)

(continued from previous page)

```
    )
);
```

18.6.19 mandatoryOne

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

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

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

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

18.6.20 mandatoryAll

This checks all fields of the field group is required.

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

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

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

18.6.21 max

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

```
$validations = array(
    'max_vote' => array(
        'caption' => _t('Max. Vote');
        'value'   => $valueToCheck,
```

(continues on next page)

(continued from previous page)

```
        'rules'   => array('mandatory', 'max'),
        'max'     => 5,
    ),
);
```

18.6.22 maxLength

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

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

18.6.23 min

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

```
$validations = array(
    'no_of_page' => 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.".
```

18.6.24 minLength

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

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

18.6.25 exactLength

This rule checks that a string/array's length is equal to the specific length. The required option - `length`.

```
$validations = array(
    'code' => array(
        'caption'   => _t('Code'),
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'exactLength'),
        'length'    => 6,
    ),
);
```

18.6.26 naturalNumber

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

18.6.27 notAllowZero

This ensures that the field is not zero.

18.6.28 numeric

It checks the field is numeric.

18.6.29 numericDash

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

18.6.30 numericSpace

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

18.6.31 positiveRationalNumber

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

18.6.32 rationalNumber

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

18.6.33 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(
    'time' => array(
        'caption'   => _t('Time');
        'value'     => $valueToCheck,
        'rules'     => array('time'),
        'timeFormat'=> '24',
    ),
);
```

18.6.34 url

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

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

18.6.35 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(
    'username' => array(
        'caption'   => _t('Username');
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'username'),
    ),
);
```

18.6.36 unique

The rule is used to check if any duplicate record exists for a specific field in the database.

```
$validations = array(
    'username' => array(
        'caption'   => _t('Username');
        'value'     => $valueToCheck,
        'rules'     => array('mandatory', 'username', 'unique'),
        'table'     => 'user', // table name to check in
        'field'     => 'username', // the field to be checked
        'id'        => $id, // Optional: id to be excluded in check
    ),
);
```

18.6.37 wholeNumber

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

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

18.7 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(
    'username' => 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);
    }
```

(continues on next page)

(continued from previous page)

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

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

    if ($qb->fetch()) {
        validation_addError('txtUsername', _t('Username already exists. Please try another one.'));
    } else {
        // No duplicate && success
    }
}
```

CHAPTER 19

File Helper

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

19.1 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).

```
# /inc/constants.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/');

# /app/inc/site.config.php
// $lc_photoDimensions - desired width and height for the uploaded photos
$lc_photoDimensions = array('400x300', '200x150'); // 3 photos will be uploaded␣
↪according to the defined dimensions
```

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
            /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.

19.2 Generic File Upload

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/photo/index.php */

$view = _app('view');

$pageTitle = _t('Photo Uploader');
_app('title', $pageTitle);

// You could have query here to retrieve the existing file upload data from db

$view->data = array(
    'pageTitle' => $pageTitle
);
```

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

```
<!-- app/photo/view.php -->
<form method="post" name="form-upload" id="form-upload" 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">
```

(continues on next page)

(continued from previous page)

```
        <tr>
            <td class="label"><?php echo _t('Photo'); ?></td>
            <td class="labelSeparator">:</td>
            <td class="entry">
                <input type="file" name="photo" id="photo" />
            </td>
        </tr>
        <tr>
            <td colspan="2"></td>
            <td class="entry">
                <button type="submit" id="btn-upload" name="btn-upload"><?php echo _t(
↪'Upload'); ?></button>
            </td>
        </tr>
    </table>
    <?php form_respond('form-upload', 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 */

if (_isHttpPost()) {
    $photo = $_FILES['photo'];

    $validations = array(
        'photo' => 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 (form_validate($validations)) {
        $file = _fileHelper();

        // set image dimension to resize
        $file->set('dimensions', _cfg('photoDimension'));

        // set file upload directory; this should be defined in /inc/constants.php
        $file->set('uploadDir', PHOTO_DIR); // optional; default to `/files/tmp/`
```

(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(
                'image' => $uploads['fileName'];
            );

            if (db_save('your_table', $data)) {
                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('photo', $error['message']);

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

19.3 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 `/app/inc/site.config.php` as described in the previous section *File Upload Form and File Handling*.

Create an instance of the class **AsynFileUploader** in `/app/photo/index.php` and pass it to view. For example, see /app/example/asyn-file-uploader/index.php.:

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

$view = _app('view');

$pageTitle = _t('AsynFileUploader');
_app('title', $pageTitle);

# The constructor argument
# string/array The input file name or The array of property/value pairs
$photo = _asynFileUploader('photo');

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

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

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

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

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

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

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

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

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

// You could have query here to retrieve the existing file upload data from db

# If there is any previously uploaded file, set it using setValue()
# @see /app/example/asyn-file-uploader/index.php
# @see /app/example/asyn-file-uploader/view.php
if (!empty($image)) {
```

(continues on next page)

(continued from previous page)

```
    $photo->setValue($image->pimgFileName, $image->pimgId);
}

$view->data = array(
    'pageTitle' => $pageTitle,
    'photo' => $photo
);
```

Call the instance 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="form-async-upload" method="post">
    <div class="message error"></div>
    <div class="table">
        <div class="row">
            <?php $photo->html() ?>
        </div>
        <div class="row">
            <input type="submit" id="btn-upload" name="btn-upload" 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 `#btn-upload` 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
/** app/photo/action.php */

if (_isHttpPost()) {
    $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
```

(continues on next page)

(continued from previous page)

```
        // $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 ###
        $data = array(
            'image' => $post['photo'];
        );

        if (db_save('your_table', $data)) {
            form_set('success', true);
            form_set('message', _t('The photo has been saved.'));
        }
    } else {
        form_set('error', validation_get('errors'));
    }
}

form_respond('form-async-upload');
```

19.4 PHP Hooks for AsynFileUploader

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

19.4.1 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}-{field_name}` 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.

For example, assuming that `post` table has `image` field which stores an uploaded image file name, that field will be updated when a new image is uploaded for an existing post by using `onUpload` hook as below:

```
// app/post/index.php
$post = db_find('post', $id);

$image = _asynFileUploader('image');
$image->setOnUpload('post_imageUpload');
$image->setHidden('postId', $post->id); // This will be available to the second␣
↪argument (array) to post_imageUpload() as key "image-postId"
if ($post->image) {
    $image->setValue($post->image, $post->id);
}

// app/helpers/file_helper.php
```

(continues on next page)

(continued from previous page)

```
function post_imageUpload($file, $post)
{
    if (isset($post['image-postId']) && $post['image-postId']) {
        # Save new file names in db
        db_update('post', array(
            'id' => $post['image-postId'],
            'image' => $file['fileName']
        ), $useSlug = false);

        return $post['photo-postId'];
    }

    return 0;
}
```

19.4.2 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.

For example, assuming that `post` table has `image` field which stores an uploaded image file name, that field will be nulled when the image is deleted by using `onDelete` hook as below:

```
// app/post/index.php
$post = db_find('post', $id);

$image = _asynFileUploader('image');
$image->setOnDelete('post_imageDelete');
if ($post->image) {
    $image->setValue($post->image, $post->id);
    // The second arugment to setValue() will be avaialble to the onDelete hook post_
↪imageDelete()
}

// app/helpers/file_helper.php
function post_imageDelete($id)
{
    if ($id) {
        return db_update('post', array(
            'id' => $id,
            'image' => null,
        ));
    }

    return false;
}
```

Note:

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

19.5 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

19.5.1 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

19.5.2 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

19.5.3 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 20

List with 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 `/app/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. There is a helper function *_pager()* to create a Pager instance. By default, the field name `page` is used in query string for the current page number. We can customize it by passing the name to the function such as `_pager('p')`.

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)

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

However, you can minimize your code when you don't need to customize your pagination settings. The following one-line is equivalent to a couple of lines aforementioned.

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

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`:

1. end.png
2. end_disabled.png
3. next.png
4. next_disabled.png
5. previous.png
6. previous_disabled.png
7. start.png
8. 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">
        <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.

20.1 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 `/app/post/index.php`,

```
$pageTitle = _t('Posts');

_app('title', $pageTitle);
_app('view')->addData('pageTitle', $pageTitle);
```

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

```
<p>
    <a href="<?php echo _url(_cfg('baseDir') . '/post/setup') ?>" class="button mini␣
↪green"><?php echo _t('Add New Post'); ?></a>
<p>

<div id="list"></div> <!-- The list will be displayed here -->
```

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

```
/** app/js/app.js */
LC.Page.Post = {
    List : {
        url : LC.Page.url(LC.vars.baseDir + '/post'), /* mapping directory */
        /* Initialize the post listing page */
        init : function() {
            LC.List.init({
                url: LC.Page.Post.List.url,
            });
        },
    }
};
```

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

```
<p>
    <a href="<?php echo _url(_cfg('baseDir') . '/post/setup') ?>" class="button mini␣
↪green"><?php echo _t('Add New Post'); ?></a>
<p>

<div id="list"></div>

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

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

# Prerequisite for the Pager
$pager = pager_ajax($rowCount);

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

<?php if ($qb->getNumRows()) { ?>
    <?php while ($row = $qb->fetchRow()) { ?>
        <p class="post">
            <h5>
                <a href="<?php echo _url('post', array($row->id, $row->slug)); ?>"><?
↪php echo $row->title; ?></a>
```

(continues on next page)

(continued from previous page)

```
            </h5>
            <p><?php echo $b->body; ?></p>
            <p>
                <a href="<?php echo _url('post', array($row->id, $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-container"><?php $pager->display() ?></div>

<?php } else { ?>
    <div class="no-record"><?php echo _t('There is no record.') ?></div>
<?php } ?>
```

20.2 Create an AJAX Listing Page with jQuery Dialog Form

Sometimes you need to add/edit an entity in a list page. PHPLucidFrame provides a convenient way to integrate a modal dialog with a form for adding or editing an entity of the list. The following would be the page structure.

```
/path_to_webserver_document_root
    /app
        /category
            |-- action.php
            |-- index.php
            |-- list.php
            |-- view.php
```

In `/app/category/index.php`

```
$pageTitle = _t('Categories');

_app('title', $pageTitle);
_app('view')->addData('pageTitle', $pageTitle);
```

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

```
<h4><?php echo $pageTitle; ?></h4>

<p>
    <button type="button" class="button mini green" id="btn-new">
        <?php echo _t('Add New Category'); ?>
    </button> <!-- #btn-new is a default button ID to launch the modal; you can␣
↪change to any ID but require to configure LC.List.init() in js -->
</p>

<div id="list"></div>
```

After then, you need to add HTML for jQuery modal dialog.

```
<!-- Category Entry Form -->
<div id="dialog-category" class="dialog" title="<?php echo _t('Category'); ?>">
    <form method="post" id="form-category"> <!-- form id is required to make it␣
↪working -->
```

(continues on next page)

(continued from previous page)

```
        <div class="message error"></div>
        <input type="hidden" id="id" name="id" />
        <table class="form fluid">
            <tr>
                <td class="label">
                    <?php echo _t('Name'); ?>
                    <span class="required">*</span>
                </td>
                <td class="label-separator">:</td>
                <td class="entry">
                    <input type="text" name="name" id="name" class="lc-form-input␣
↪fluid-100" />
                </td>
            </tr>
            <tr>
                <td colspan="2">
                <td class="entry">
                    <button type="submit" class="button jqbutton submit large green"␣
↪id="btn-save" name="btn-save">
                        <?php echo _t('Save') ?>
                    </button>
                    <button type="button" class="button jqbutton large" id="btn-cancel
↪" name="btn-cancel">
                        <?php echo _t('Cancel') ?>
                    </button> <!-- #btn-cancel is a default button ID to close the␣
↪modal dialog -->
                </td>
            </tr>
        </table>
        <?php form_token(); ?>
    </form>
</div>
```

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

```
LC.Page.Category = {
    url : LC.Page.url(LC.vars.baseDir + '/category'), /* mapping directory */

    /* Initialize the Category page */
    init : function() {
        LC.List.init({
            url: LC.Page.Category.url, // This will call /app/category/list.php
            formModal: '#dialog-category', // HTML id for the modal used in the view
            formId: 'form-category', // HTML id for the form used in the view
            editCallback: LC.Page.Category.edit, // define below
            // see more options at https://phplucidframe.readthedocs.io/en/latest/
↪ajax-and-javascript-api.html#lc-list-init-options
        });
    },

    /* Initialize the list */
    list: function() {
        LC.List.list();
    },

    /* Callback to set values when the dialog is open to edit an existing entry */
    edit : function($form, $data) {
```

(continues on next page)

(continued from previous page)

```
        $form.find('input[name=name]').val($data.name);
    }
};
```

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

```
<script>
    $(function() {
        LC.Page.Category.init();
    });
</script>
```

In `/app/category/list.php`,

```
<?php

list($qb, $pager, $total) = db_findWithPager('category', array('deleted' => null),␣
↪array('name' => 'asc'));

if ($qb->getNumRows()): ?>
    <table class="list table">
        <tr class="label">
            <td class="table-left" colspan="2"><?php echo _t('Actions'); ?></td>
            <td>Name</td>
        </tr>
        <?php
        $data = array();
        while ($row = $qb->fetchRow()):
        ?>
            <?php $data[$row->id] = $row; # data for the modal dialog form ?>
            <tr>
                <td class="table-left actions col-action">
                    <a href="#" class="edit edit-ico" title="Edit" rel="<?php echo
↪$row->id; ?>">
                        <span><?php echo _t('Edit'); ?></span>
                    </a> <!-- ".table .actions .edit" is a default class hierarchy␣
↪for edit button action -->
                </td>
                <td class="col-action">
                    <a href="#" class="delete delete-ico" title="Delete" rel="<?php␣
↪echo $row->id; ?>">
                        <span><?php echo _t('Delete'); ?></span>
                    </a> <!-- ".table .actions .default" is a default class hierarchy␣
↪for delete button action -->
                </td>
                <td class="colName">
                    <?php echo $row->name; ?>
                </td>
            </tr>
        <?php endwhile; ?>
    </table>

    <div class="pager-container"><?php $pager->display(); ?></div>

    <?php _addFormData('form-category', $data); # the first parameter is the form ID␣
↪for the form in the modal dialog defined in the view ?>
<?php else: ?>
```

(continues on next page)

(continued from previous page)

```
    <div class="no-record"><?php echo _t('There is no item found. Click "Add New␣
↪Category" to add a new category.'); ?></div>
<?php endif;
```

Lastly, add create/edit/delete handling code in `/app/category/action.php` because, by default, the form in the modal will be submitted to it.

```
<?php

$table = 'category';
$success = false;

if (_isHttpPost()) {
    $post = _post();

    if (isset($post['action']) && $post['action'] == 'delete' && !empty($post['id']))
↪{
        # DELETE category
        if (db_delete($table, array('id' => $post['id']))) {
            $success = true;
        }
    } else {
        # NEW/EDIT
        $validations = array(
            'name' => array(
                'caption'       => _t('Name'),
                'value'         => $post['name'],
                'rules'         => array('mandatory'),
                'parameters'    => array($post['id'])
            )
        );

        if (form_validate($validations)) {
            $data = array(
                'name' => $post['name']
            );

            if (db_save($table, $data, $post['id'])) {
                $success = true;
            }
        } else {
            form_set('error', validation_get('errors'));
        }
    }

    if ($success) {
        form_set('success', true);
        form_set('callback', 'LC.Page.Category.list()'); # Ajax callback
    }
}

form_respond('form-category');
```

20.3 Create a Generic Listing Page without AJAX

Sometimes, you may not want to use AJAX list. You can easily disable LucidFrame AJAX pagination option. In this case, you don't need to have `/app/post/list.php` like in the above example.

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

Retrieve your data in `index.php` and then render your HTML in `/app/post/view.php`. You don't need to write Javascript in this case.

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

$pageTitle = _t('Latest Posts');
$view = _app('view');

_app('title', $pageTitle);

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

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

# OR just one-line
// $pager = pager_ordinary();

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

# Pass data to the view layer
$view->data = array(
    'pageTitle'     => $pageTitle,
    'totalCount'    => totalCount
    'pager'         => $pager,
    'qb'            => $qb,
);
```

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

```
<!-- app/post/view.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->id, $row->slug)) ?>">
↪<?php echo $row->title; ?></a>
                </h5>
                <p><?php echo $b->body; ?></p>
                <p>
                    <a href="<?php echo _url('post', array($row->id, $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-container clearfix">
            <?php $pager->display(); ?>
            <div class="pager-records"><?php echo _t('Total %d records', $totalCount);
↪ ?></div>
        </div>
    <?php } else { ?>
        <div class="no-record"><?php echo _t('There is no record.'); ?></div>
    <?php } ?>
</div>
```

20.4 Customize Pagination Display

As of version 3.0, you can pass a callack function name to the `display()` method of Pager instance, for example,

```
<?php $pager->display('pager_custom') ?>
```

You need to define your custom pager function `pager_custom()` in `/app/helpers/pager_helper.php`. The function will receive an array parameter.

```
function pager_custom($result)
{
    # The outermost container must have "lc-pager" class for AJAX pagination
    // render HTML output for your custom pagination
}
```

The parameter to `pager_custom()` will have this array structure:

```
Array(
    [offset] => xx
    [thisPage] => xx
    [beforePages] => Array()
    [afterPages] => Array()
    [firstPageEnable] => xx
    [prePageEnable] => xx
    [nextPageNo] => xx
    [nextPageEnable] => xx
```

(continues on next page)

(continued from previous page)

```
    [lastPageNo] => xx
    [lastPageEnable] => xx
    [url] => xx
    [ajax] => 1 or 0
)
```

Here is an example to render boostrap-styled pagination:

```
function pager_bootstrap($result)
{
    # The outermost container must have "lc-pager" class for AJAX pagination
    ?>
    <ul class="pagination lc-pager">
        <li class="first">
            <?php if ($result['firstPageEnable']): ?>
                <a href="<?php echo _url($result['url']) ?>" rel="<?php echo $result[
↪'firstPageNo'] ?>"><?php echo _t('First') ?></a>
            <?php else: ?>
                <span><?php echo _t('First') ?></span>
            <?php endif ?>
        </li>

        <li class="prev">
            <?php if ($result['prePageEnable']): ?>
                <a href="<?php echo _url($result['url']) ?>" rel="<?php echo $result[
↪'prePageNo'] ?>">«</a>
            <?php else: ?>
                <span>«</span>
            <?php endif ?>
        </li>

        <?php if (!empty($result['beforePages'])): ?>
            <?php foreach ($result['beforePages'] as $pg): ?>
                <li>
                    <?php if ($result['ajax']): ?>
                        <a href="<?php echo _url($result['url']) ?>" rel="<?php echo
↪$pg ?>"><?php echo $pg ?></a>
                    <?php else: ?>
                        <a href="<?php echo _url($result['url'], array($this->
↪pageQueryStr => $pg)) ?>" rel="<?php echo $pg ?>"><?php echo $pg ?></a>
                    <?php endif ?>
                </li>
            <?php endforeach; ?>
        <?php endif ?>

        <li class="active">
            <a href="#"><?php echo $result['thisPage'] ?></a>
        </li>

        <?php if (!empty($result['afterPages'])): ?>
            <?php foreach ($result['afterPages'] as $pg): ?>
                <li>
                    <?php if ($result['ajax']): ?>
                        <a href="<?php echo _url($result['url']) ?>" rel="<?php echo
↪$pg ?>"><?php echo $pg ?></a>
                    <?php else: ?>
                        <a href="<?php echo _url($result['url'], array($this->
↪pageQueryStr => $pg)) ?>" rel="<?php echo $pg ?>"><?php echo $pg ?></a>
```

(continues on next page)

(continued from previous page)

```
                    <?php endif ?>
                </li>
            <?php endforeach; ?>
        <?php endif ?>

        <li class="next">
            <?php if ($result['nextPageEnable']): ?>
                <a href="<?php echo _url($result['url']) ?>" rel="<?php echo $result[
↪'nextPageNo'] ?>">»</a>
            <?php else: ?>
                <span>»</span>
            <?php endif ?>
        </li>

        <li class="last">
            <?php if ($result['lastPageEnable']): ?>
                <a href="<?php echo _url($result['url']) ?>" rel="<?php echo $result[
↪'lastPageNo'] ?>"><?php echo _t('Last') ?></a>
            <?php else: ?>
                <span><?php echo _t('Last') ?></span>
            <?php endif ?>
        </li>
    </ul>
    <?php
}
```

Note:

- PHPLucidFrame 3.0 included a pagination helper `pager_bootstrap()` in `/app/helpers/pager_helper.php`. You can use it to display boostrap-styled pagination or you can see the code as reference for your custom pagination callback function.

CHAPTER 21

Authentication & Authorization

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
$lc_auth = array(
    'table' => '', // table name, for example, user
    'fields' => array(
        'id'    => '',  // PK field name, for example, user_id
        'role'  => ''   // User role field name, for example, user_role
    ),
    'permissions'  => array(
        // permissions allowed
        // role name => array of permission names
        // For example,
        // 'admin' => array('post-list', 'post-add', 'post-edit', 'post-delete'),
        // 'editor' => array('post-list', 'post-add', 'post-edit') // editor is not␣
↪allowed for post deletion
        // If you store permissions in your db, implement auth_permissions($role) in /
↪app/helpers/auth_helper.php
        // to return the permission list from your db
    ),
);
```

According to the sample database `/db/schema.sample.php`,

1. `$lc_auth['table']` would be `user`.
2. `$lc_auth['fields']['id']` would be `id` (i.e., `user.id`).
3. `$lc_auth['fields']['role']` would be `role` (i.e., `user.role`).

21.1 Encrypting Passwords

Encrypting passwords are always required in every secured web application. When user data are inserted in user registration process, it is advisable to encrypt 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.

21.2 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()`.

```
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 as an object accessible from `_app('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 from `_app('auth')`, it also contains session id, token and permissions:

- `_app('auth')->sessId` - The session id for the current session returning from `session_id()`.
- `_app('auth')->token` - The generated token of the authentication object
- `_app('auth')->permissions` - The permissions allowed according to the defined role and perms 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 = _app('auth');
$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.

Note:

- For login sample code, see https://github.com/phplucidframe/admin-boilerplate/tree/master/login.
- For logout sample code, see https://github.com/phplucidframe/admin-boilerplate/blob/master/logout/index.php.

21.3 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
}
```

21.4 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()` or multiple user roles by using `auth_roles()`, for example,

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

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

And it also allows you to check the user is accessible to a particular page or section by using `auth_can()`, for example,

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

if (auth_can('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() {
```

(continues on next page)

(continued from previous page)

```
    return auth_role('editor');
}
```

You can also check the URL route path or name to prevent the user from being accessed to a page or a function. You can implement this as middleware. The following middleware will be invoked in all routes under `/admin` except `/admin/login` and `/admin/logout`

```
// app/middleware/auth.php

$baseDir = _cfg('baseDir'); // Let says _cfg('baseDir') is '/admin'

_middleware(function () {
    if (auth_isAnonymous()) {
        flash_set('You are not authenticated. Please log in.', '', 'error');
        _redirect$baseDir . '/login');
    }
})->on('startWith', $baseDir)
    ->on('except', array($baseDir . 'login', $baseDir . 'logout'));
```

The following example is to allow post delection for admin only.

```
// app/middleware/auth.php

_middleware(function () {
    if (!auth_role('admin')) {
        _page403();
    }
})->on('equal', 'post_delete');
```

The following example is to allow users section (all routes containing a URI segment "users") for admin only.

```
_middleware(function () {
    if (!auth_role('admin')) {
        _page403();
    }
})->on('contain', 'users');
```

21.5 Working with Permissions in Your Database

Sometimes, you may have user roles and permissions (ACL) in your database. Let's say for example, you have the following data structure in your database.

role

id	name
1	Admin
2	Editor

role_permission

id	role_id	name
1	1	post-create
2	1	post-update
3	1	post-delete
4	2	post-create
5	2	post-update

user

id	role_id	username
1	1	admin
2	2	dummy

You would need to add this function in `/app/helpers/auth_helper.php` to override `auth_permissions()` in `/lib/helpers/auth_helper.php`. The function should return the list of permissions by the given role id.

```
/**
* Get the permissions of a particular role
* @param string|int $role The user role name or id
* @return array|null Array of permissions of the role
*/
function auth_permissions($role)
{
    $result = db_select('role_permission')
        ->where()->condition('role_id', $role)
        ->getResult();

    return array_column($result, 'name');
}
```

Then, set `role_id` to `$lc_auth['fields']['role']` in `/inc/config.php`.

```
# $lc_auth: configuration for the user authentication
# This can be overidden by defining $lc_auth in /inc/site.config.php
$lc_auth = array(
    'table' => '', // table name, for example, user
    'fields' => array(
        'id'    => 'id',  // PK field name, for example, user_id
        'role'  => 'role_id'   // User role field name, for example, user_role
    ),
    'permissions'  => array(
        // permissions allowed
        // role name => array of permission names
        // For example,
        // 'admin' => array('post-list', 'post-add', 'post-edit', 'post-delete'),
        // 'editor' => array('post-list', 'post-add', 'post-edit') // editor is not␣
↪allowed for post deletion
        // If you store permissions in your db, implement auth_permissions($role) in /
↪app/helpers/auth_helper.php
        // to return the permission list from your db
    ),
);
```

Since you use the `role_id` field for `$lc_auth['fields']['role']`, you will have to use role id when calling `auth_role()` or `auth_roles()`

```
if (auth_role(2)) {
    // if user is editor, do something
} else {
    // redirect to the access-denied page
}

if (auth_roles(1, 2)) {
    // if user is admin or editor, do something
} else {
    // redirect to the access-denied page
}
```

CHAPTER 22

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()`.

22.1 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.

22.2 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.

22.3 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));
```

22.4 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 23

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 3.3.0 by Sithu K.

3.0.0
PHP Version: 7.3.5
The MIT License
Simple, lightweight & yet powerful PHP Application Framework
Copyright (c) 2014-2021, phplucidframe.com
```

23.1 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 3.3.0 by Sithu K.

Usage:
  secret:generate [options]

Options:
  -h, --help     Display the help message

Help:
  Generate a secret key
```

If you want to generate a new secret key, run,

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

You can check all available command list by running

```
$ php lucidframe list
```

23.2 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!
```

As of version 2.2, it supports class to add command definition. You can create a class in `/app/cmd/classes/`. Let's say `/app/cmd/classes/Greet.php`

```
<?php

use LucidFrame\Console\CommandInterface;
use LucidFrame\Console\Command;

class Greet implements CommandInterface
{
    public function execute(Command $cmd)
    {
        $name = $cmd->getArgument('name');
        $msg = $name ? 'Hello ' . $name : 'Hello';
        $msg .= '!';

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

        _writeln($msg);
    }
}
```

Then you need to connect that class to the command definition. Set class name to `setDefinition()` in `/app/cmd/GreetCommand.php` you created above.

```
_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('Greet')
    ->register();
```

It is useful when your command has more complicated functions and you can write logic neatly in your class.

CHAPTER 24

Useful Helpers

PHPLucidFrame provides some useful helper functions.

24.1 _app($name, $value = null)

Get/Set a global variable/object.

Parameters:

Name	Type	Description
`$name`	string	The name of the variable/object
`$value`	mixed	(Optional) The value or the variable or object

When the second parameter is ommited, it is a getter function, otherwise it is a setting function.

Return Values:

The value stored globally

Getter Usage	Description
`_app('db')`	To get the globally connected database object
`_app('view')`	To get the global View object
`_app('auth')`	To get the global authentication object
`_app('title')`	To get the current page title

Setter Usage	Description
`_app('db', new \LucidFrame\Core\Database($name, create);`	To create a new global database object (it is rarely used)
`_app('view', new \LucidFrame\Core\View())`	To set a new View object (it is rarely used)
`_app('auth', $auth)`	To set/update the authentication object globally
`_app('title', 'Home')`	To set the current page title

24.2 _cfg($key, $value = '')

Get/Set a config variable

Parameters:

Name	Type	Description
`$key`	string	The config variable name without prefix
`$value`	mixed	(Optional) The value to set to the config variable; if it is omitted, it is Getter method.

When the second parameter is ommited, it is a getter function, otherwise it is a setting function.

Return Values:

The value of the config variable

24.3 _cfgOption($name, $key)

Get the value of the array config variable by its key

Parameters:

Name	Type	Description
`$name`	string	The config array variable name without prefix
`$key`	string	The key of the config array of which value to be retrieved

Return Values:

The value of a single column of the config array variable

24.4 _env($name, $default = '')

Get the parameter value by name defined in `/inc/parameter/parameter.env.inc`

Parameters:

Name	Type	Description
`$name`	string	The parameter name in dot annotation format such as `prod.db.default.database`
`$default`	mixed	The default value if the parameter name doesn't exist

Return Values:

The value defined in `/inc/parameter/parameter.env.inc`

24.5 _p($name = 'env')

Get the parameter value by name defined in `/inc/parameter/(development|production|staging|test).php`

Parameters:

Name	Type	Description
`$name`	string	The parameter name defined as key in `/inc/parameter/(development|production|staging|test).php`. The file development, production, staging or test will be determined according to the value from `.lcenv`. If `$name` is `env` (by default), it returns the current environment setting from `.lcenv`.

You can use dot notation format for the name parameter such as `db.default.database` for multi-level array keys.

Return Values:

The value defined `/inc/parameter/(development|production|staging|test).php`

24.6 _baseUrlWithProtocol()

Get site base URL with protocol according to the config

24.7 _arg($index = null, $path = null)

Return a component of the current path. When viewing a page http://www.example.com/foo/bar and its path would be "foo/bar", for example, `_arg(0)` returns "foo" and `_arg(1)` returns "bar"

Parameters:

Name	Type	Description
`$index`	mixed	The index of the component, where each component is separated by a '/' (forward-slash), and where the first component has an index of 0 (zero).
`$path`	string	A path to break into components. Defaults to the path of the current page.

Return Values:

The component specified by `$index`, or `null` if the specified component was not found. If called without arguments, it returns an array containing all the components of the current path.

24.8 _entity($table, $dbNamespace = null)

Get default entity object from the schema

Parameters:

Name	Type	Description
`$table`	string	The mapped table name without prefix
`$dbNamespace`	string|null	The current db namespace

Return Values:

The empty stdClass object with field names as properties

24.9 _addJsVar($name, $value = '')

Passing values from PHP to Javascript making available to `LC.vars`

Parameters:

Name	Type	Description
`$name`	string	The JS variable name that will be available to `LC.vars`, e.g., if $name is 'foo', it will be available as `LC.vars.foo` in JS.
`$value`	mixed	The value for the JS variable

24.10 _addHeadStyle($file)

Add CSS file to be included in head section

Parameters:

Name	Type	Description
`$file`	string	An absolute file path or file name only. The file name only will be prepended the folder name `css/` and it will be looked in every sub-sites `css` folder

24.11 _addHeadScript($file)

Add JS file to be included in head section

Parameters:

Name	Type	Description
`$file`	string	An absolute file path or file name only. The file name only will be prepended the folder name `js/` and it will be looked in every sub-sites `js` folder

24.12 _json(array $data, $status = 200)

Header sent as text/json

Parameters:

Name	Type	Description
`$data`	array	Array of data to be encoded as JSON
`$status`	int	HTTP status code, default to 200

24.13 _requestHeader($name)

Get a HTTP request header by name

Parameters:

Name	Type	Description
`$name`	string	The HTTP header name

For example, to get the value from HTTP Authorization in header:

```
$authorization = _requestHeader('Authorization');
```

Return Values:

The HTTP header value from the request

24.14 _r()

Get the current routing path, for example,

Note:

- `example.com/foo/bar` would return `foo/bar`
- `example.com/en/foo/bar` would also return `foo/bar`
- `example.com/1/this-is-slug` (if accomplished by RewriteRule) would return the underlying physical path

---

Return Values:

The route path starting from the site root

24.15 _rr()

The more realistic function to get the current routing path on the address bar regardless of RewriteRule behind, for example,

---

Note:

- `example.com/foo/bar` would return `foo/bar`
- `example.com/en/foo/bar` would also return `foo/bar`
- `example.com/foo/bar?id=1` would also return `foo/bar`

- `example.com/1/this-is-slug` would return `1/this-is-slug`

Return Values:

The route path starting from the site root

24.16 _url($path = null, $queryStr = array(), $lang = '')

Get the absolute URL path

Parameters:

Name	Type	Description
`$path`	string	Routing path such as "foo/bar"; null for the current path
`$queryStr`	array	Query string as key/value array
`$lang`	string	Language code to be prepended to $path such as "en/foo/bar". It will be useful for site language switch redirect.

Return Values:

The absolute URL path

24.17 _redirect($path = null, $queryStr = array(), $lang = '', $status = null)

Header redirect to a specific location

Parameters:

Name	Type	Description
`$path`	string	Routing path such as "foo/bar"; null for the current path
`$queryStr`	array	Query string as key/value array
`$lang`	string	Language code to be prepended to $path such as "en/foo/bar". It will be useful for site language switch redirect.
`$status`	int	The HTTP status code

24.18 _page404()

Redirect to 404 page

24.19 _shorten($str, $length = 50, $trail = '...')

Shorten a string for the given length

Parameters:

Name	Type	Description
`$str`	string	A plain text string to be shortened
`$length`	int	The character count. Default is 50.
`$trail`	string	To append `...` or not. `null` to not show

Return Values:

The shorten text string

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

Format a date

Parameters:

Name	Type	Description
`$date`	string	A date to be formatted
`$format`	string	The date format; The config variable will be used if it is not passed

When both parameters is not provided, the current formatted date will be returned.

Return Values:

The formatted date

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

Format a date/time

Parameters:

Name	Type	Description
`$dateTime`	string	A date/time to be formatted
`$format`	string	The date/time format; The config variable will be used if it is not passed

When both parameters is not provided, the current formatted date/time will be returned.

Return Values:

The formatted date/time

24.22 _randomCode($length = 5, $letters = array(), $prefix = '')

Generate a random string from the given array of letters.

Parameters:

Name	Type	Description
`$length`	int	The length of required random string
`$letters`	array	Array of letters from which randomized string is derived from. Default is a to z and 0 to 9.
`$prefix`	string	Prefix to the generated string

Return Values:

The random string of required length

24.23 _slug($string, $table = '', array $condition = array())

Generate a slug of human-readable keywords

Parameters:

Name	Type	Description
`$string`	string	Text to slug
`$table`	string	Table name to check in. If it is empty, no check in the table
`$condition`	array	Condition to append table check-in, e.g, `array('fieldName !=' => value)`

Return Values:

The generated slug

24.24 _isHttpPost()

Check if HTTP request method is POST and has request data

Return Values:

`TRUE` if HTTP request method is POST and it has request data, otherwise `FALSE`

Note:

- See more helper functions at http://www.phplucidframe.com/api-doc/latest/

CHAPTER 25

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. Moreover, `LC` has there core objects - `LC.Page`, `LC.Form` and `LC.List`. They all provide utilities to make your pages, forms and lists easier.

PHPLucidFrame recommends the code organization in a Javascript file so-called `app.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/app.js` in the release.

```
// /app/js/app.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('app.js'); ?>
```

25.1 The Page

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

25.1.1 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

25.1.2 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.

25.1.3 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() { .. } }`

25.1.4 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.

25.1.5 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')`.

25.1.6 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.

25.2 The Form

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

25.2.1 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.

25.2.2 jQuery Datepicker

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

25.2.3 jQuery Button

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

25.2.4 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

25.2.5 LC.Form.getFormData(formId, id)

Get the embedded JSON form data.

Parameters:

Name	Type	Description
`formId`	integer	HTML Form ID
`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

---

Note:

- You can check the completed example source code at https://github.com/phplucidframe/admin-boilerplate

---

25.2.6 LC.Form.slug(str)

Generate slug value from the given string

Parameters:

Name | Type | Description
`str` | string | The string to generate slug

Returns:

`<string>` The slug value

25.2.7 LC.Form.beforeSubmit

Bind beforeSubmit hook to AJAX form

Parameters:

Name	Type	Description
`formId`	string	HTML form id
`callback`	function	A callback function that handles the event before the form is submitted.

25.2.8 LC.Form.afterSubmit

Bind afterSubmit hook to AJAX form

Parameters:

Name	Type	Description
`formId`	string	HTML form id
`callback`	function	A callback function that handles the event after the form is submitted.

25.3 The List

As of PHPLucidFrame 3.0, `LC.List` is added to help create AJAX list easily.

25.3.1 LC.List.init(options)

Initialize an AJAX list

Parameters:

Name	Type	Default	Description
`options`	object		The object with the following properties
`options.url`	string	`LC.Page.url(LC.vars.baseDir)`	The URL for AJAX request to render the list
`options.params`	string	`{}`	Optional parameters to `options.url`
`options.id`	string	`list`	HTML id for the list container element
`options.formModal`	string	`#dialog-item`	HTML id of jQuery dialog modal to create/edit an entity
`options.formModalCancelButton`	string	`#btn-cancel`	HTML id of the button to close the dialog modal
`options.formId`	string	`dialog-form`	HTML id of the form in the dialog modal
`options.confirmModal`	string	`#dialog-confirm`	HTML id of the confirm modal to delete an entity
`options.confirmModalTitle`	string	Confirm Delete	Title of the confirm modal to delete an entity
`options.confirmModalMessage`	string	Are you sure you want to delete?	Message of the confirm modal to delete an entity
`options.createButton`	string	`#btn-new`	HTML id of the button to launch the dialog modal to create a new entry
`options.editButtonClass`	string	`.table .actions .edit`	HTML id/class of the button to launch the dialog modal to edit an entry
`options.deleteButtonClass`	string	`.table .actions .delete`	HTML id/class of the button to delete an entry
`options.createCallback`	string	null	A callback function to be invoked before the dialog modal is opened for creation
`options.editCallback`	string	null	A callback function to be invoked before the dialog modal is opened for editing
`options.deleteCallback`	string	null	A callback function to be invoked after the entry is deleted

25.3.2 LC.List.list(listId, [arg1, arg2])

Load the list by AJAX

Parameters:

When three parameters are provided,

Name	Type	Description
`listId`	string	HTML id of the list container; default is `list`
`arg1`	string	URL to request
`arg2`	object	Parameters to URL

When two parameters are provided and the second parameter is string,

Name	Type	Description
`listId`	string	HTML id of the list container; default is `list`
`arg1`	string	URL to request

When two parameters are provided and the second parameter is object,

Name	Type	Description
`listId`	string	HTML id of the list container; default is `list`
`arg1`	object	Parameter to URL to request

Note: When no parameter is provided to the function, the first parameter `listId` would be `list` by default.

25.3.3 LC.List.create(listId)

Launch the dialog to create a new entity

Parameters

Name	Type	Description
`listId`	string	HTML id of the list container; default is `list`

25.3.4 LC.List.edit(id, listId)

Launch the dialog to edit an existing entity

Parameters

Name	Type	Description
`id`	mixed	ID (from db) to edit (required)
`listId`	string	HTML id of the list container; default is `list`

25.3.5 LC.List.remove(id, listId)

Launch the dialog to confirm for deleting an entity

Parameters

Name	Type	Description
`id`	mixed	ID (from db) to edit (required)
`listId`	string	HTML id of the list container; default is `list`

25.3.6 LC.List.doDelete(listId)

Do delete action upon confirmation (after approval of the dialog launched by `LC.List.remove()`)

Parameters

Name	Type	Description
`listId`	string	HTML id of the list container; default is `list`

25.4 LC.DependentUpdater

Change another select dropdown (child) upon one select dropdown (parent) change using AJAX request.

HTML Attributes for The Parent <select> Element:

Name	Required	Description
`data-dependency`	Yes	The HTML element (`select`) ID of the child element
`data-url`	Yes	URL for AJAX request to retrieve data by the selected id of the parent element
`data-callback`	No	A js callback function to invoke after the AJAX request is completed.

HTML Attributes for The Child <select> Element:

Name	Required	Description
`data-value`	No	The value to be selected after the AJAX request is completed.

The example scenario is to generate townships by a region selected. Here is an example code:

```
<!-- parent element -->
<select class="form-control" id="region" name="region"
        data-dependency="#township"
        data-url="<?php echo _url('api/townships') ?>">
    <option value=""><?php echo _t('Select Region') ?></option>
    <option value="1">Yangon</option>
    <option value="2">Mandalay</option>
</select>

<!-- child element -->
<select class="form-control" id="township" name="township">
    <option value=""><?php echo _t('Select Township') ?></option>
</select>
```

You need to create `/app/api/townships.php` (the URL you would provide in the `data-url` attribute) with the following similar code to return a json response.

```
$regionId = _arg('parentId');

$qb = db_select('town', 't')
    ->fields('t', array('id', 'name'))
    ->where()
        ->condition('region_id', $regionId)
    ->orderBy('name');

$result = array();
while ($row = $qb->fetchRow()) {
    $result[$row->id] = $row->name;
}

_json($result);
```

Then, the township dropdown will be re-generated whenever the region dropdown is changed.

25.5 LC.eval

Evaluates JavaScript code represented as a string.

Parameters

Name	Type	Description
`statement`	string	The javascript expression as string to evaluate

25.6 LC.getKeyByValue

Get a key in an object by its value

Parameters

Name	Type	Description
`object`	object	The object to search in
`value`	mixed	The value to search in the first parameter object

CHAPTER 26

Hooks And Overrides

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

26.1 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.

26.1.1 __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`.

26.1.2 __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.

26.1.3 __validation_messages()

Define customer validation messages by rule as key. It can be defined in `/app/helpers/validatio_helper.php`. For example,

```
/**
* Hook for custom validation messages
* @return string[]
*/
function __validation_messages()
{
    return array(
        # rule => message
        'validate_emailRetyped'     => 'Your re-typed email address does not match.',
        'validate_confirmPassword'  => '"%s" does not match.',
    );
}
```

26.1.4 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.

26.1.5 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.

26.1.6 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.

26.1.7 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

26.2 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.

26.2.1 __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`.

26.2.2 __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`.

26.2.3 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`.

26.2.4 auth_getUserInfo($id)

Overrides the function `auth_getUserInfo()` in `/lib/helpers/auth_helper.php`. Get user record from db to create auth session. It can be defined `/app/helpers/auth_helper.php`.

26.2.5 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`.

26.2.6 auth_roles($role, ...)

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

26.2.7 auth_permissions($role)

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`.

26.2.8 auth_can($perm)

Overrides the function `auth_can()` 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`.

26.2.9 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`.

26.2.10 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`.

26.2.11 _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`

26.2.12 _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`.

26.2.13 _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`.

26.2.14 _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`.

26.2.15 _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`.

26.2.16 _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`.

26.2.17 _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`.

26.2.18 _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`.

26.2.19 _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`.

26.2.20 _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`.