Leum Documentation

Release 0.0.1

Tom Parker

Oct 04, 2018

Contents:

CHAPTER 1

Plugins with Leum reference

If the `foobar` plugin was to be loaded leum will look in `leum/plugins/foobar/` for a php file called `foobar.php`. Leum will import that file and create a new instance of the `foobar` class.

The best place/time to register hooks is in the `__construct()` method of the class.

CHAPTER 2

Setup

Installing leum is pretty straightforward.

It's broken down into a few steps.

1. Configure web-server.
2. Create symbolic links to media.
3. Set file and directory permissions.
4. Create and configure the database.
5. Modify the configuration files for leum.
6. Run `/leum/setup.php` using your browser.
7. Create an root account.

2.1 Configuring the web-server

If using symbolic links to store your media and thumbnails (highly recommended) you'll need to allow symbolic links in your site's configuration.

If you store all your media and thumbails in the content directory you can allow symlinks in only that directory.

2.2 Creating symbolic links

In `leum.conf.php` the `MEDIA_DIR` and `THUMB_DIR` settings set where leum stores it's media and thumbnails respectively. The content of these directories must bee accessible via the web server.

I strongly recommend using symbolic links.

My content directory looks like this.

```
tom@redcat:/var/www/leum$ ls -l content/
total 0
lrwxrwxrwx 1 tom tom 17 Sep 16 13:47 media -> /mnt/raid1/media/
lrwxrwxrwx 1 tom tom 24 Sep 16 13:47 thumbs -> /mnt/drive4/leum/thumbs/
```

Notice how media is pointing to `/mnt/raid1/media/` and thumbs to `/mnt/drive4/leum/thumbs/`. These locations are on different drives the www directory.

2.3 Permissions

Leum needs to have access to write and create in `/logs/` and `THUMB_DIR`. Some plug ins require write access in `MEDIA_DIR`. The web-server must be allowed to serve the files in `THUMB_DIR` and `MEDIA_DIR`.

2.4 Database

You'll need to create a database for leum a good practice is to also create a user with full permissions to this database as well.

The `conf/database.conf.php` is where you provide leum with the username password host and name for the database.

2.5 Configuration

All configuration files are available in `leum/conf/`. Leum loads the `leum.conf.file`. All other configuration files included from this file using PHP's include_once.

You can enable and disable plug ins with the `ACTIVE_PLUGINS` array.

2.6 Setup.php

Access the setup.php file through your browser. If all goes well the final message should be '*setup process complete with no errors*'.

If not. Read the errors or/and the logs provided in the logs directory.

2.7 Create user

You'll need to create a new user.

CHAPTER 3

Hooks

An object subscribes to a hook event. When the hook is invoked, every subscriber to that hook is invoked.

Anything can invoke, create or add to a hook event. Generally, if you did not create the hook, don't invoke it.

Other than a few important hooks, hooks are formatted as `[source].[type].[context]`. For example `leum.media.scan-new` means leum found new media to scan.

Internally leum stores hooks names as the key to an array (just a string). However later this might change and require more precise control. Hence the future proofing.

Hook events do not currently pass arguments.

To register a hook use `LeumCore\:\:AddHook()` and to Invoke a hook call `LeumCore\:\:InvokeHook()`.

3.1 Leum Hooks

initialize Invoked after leum is up and running. **Note:** this is after routes are created and resolved.

leum.setup Invoked when leum is being set-up via the setup.php script.

leum.front.routes Invoked when leunm's font has assigned the default routes.

leum.front.footer Invoked after the footer is outputted.

leum.front.header Invoked after the header is outputted.

CHAPTER 4

Log

Leum logs exceptions and PHP errors to the `logs/default.txt` file by default.

Custom messages can be logged to the default.txt or any other log file name using a simple `Log\:\:Write("Hello World");` call.

4.1 Log::Write

```
public static void Log::Write(string $message [ int $level = LOG::INFO, string
↪$logFile ] )
```

Logs the message to a log file. Timestamp and level is added automatically.

4.1.1 Parameters

$message The message to write to the log.

$level (optional) The log level of this message. By default it's `LOG::INFO`. See *Log Levels*.

$logFile (optional) The name of the file to log the message to. If set to **null** the log file default.txt is used.

4.1.2 Example

```
<?php
        Log::Write("This warning is important!", LOG::WARNING);
        Log::Write("The secret page was accessed", LOG::INFO, 'secret-access.txt');

        // Writes the following to the log/default.txt
        // 18:09:02 04:03:23 [warning] This warning is important!
        // And writes the following to the log/secret-access.txt file.
        // 18:09:02 04:07:36 [info] The secret page was accessed
```

(continues on next page)

(continued from previous page)

```
?>
```

4.2 Log Levels

LOG::EXCEPTION (int 0) This is reserved for uncaught exceptions. This is the lowest level at zero.

LOG::ERROR (int 1) Used for logging errors including PHP errors.

LOG::WARNING (int 2) Warnings, leum can still function however features or functionality might be lacking.

LOG::INFO (int 3) Information that is not all to important.

LOG::DEBUG (int 4) Debugging information, Currently not used as using echo is much easier for rapid prototyping.

CHAPTER 5

Dispatcher

The dispatcher is used to route pages to urls. The dispatcher currently supports index and slug variables.

5.1 Constants

Dispatcher::BAD_ROUTE Is used when a route was found but another error occurred.

Dispatcher::FOUND When a route has been found.

Dispatcher::NOT_FOUND Used when a route was not found like 404.

5.2 Route Variables

%index% An unlimited length of numerical characters. Regex equivalent: `([0-9]+)`.

Examples: `302`, `2202`, `129`

%slug% Characters `a to z`, `0 to 9` and `-`. Regex equivalent: `([-a-z0-9]+)`.

Examples: `the-grand-tour`, `mcmtv2`, `300`, `iron-man-2`

5.3 Dispatcher:AddRoute

```
static void Dispatcher::AddRoute(string $route, object $target)
```

Adds a route to the target.

5.3.1 Parameters

route The route string is a url like string. This string supports the variables listed below.

target The target object. Currently leum only supports a path to a page. It's capable of supporting anonymous functions however leum is not.

5.3.2 Example

```
<?php
        // /leum/testing/example/1498 goes to pages/testing.php.
        // 1498 is passed as an argument for the page.
        Dispatcher::AddRoute('testing/example/%index%', 'pages/testing.php');
        // /leum/fan-clubs/teen-titans-go will take you to clubs.php
        // while passing %slugs% to the page.
        Dispatcher::AddRoute('fan-clubs/%slug%', 'clubs.php');v
?>
```

5.4 Dispatcher:ResolveRoute

```
static void Dispatcher::ResolveRoute(string $request)
```

Attempts to resolve a route for the request.

5.4.1 Parameters

request The url or path that needs to be resolved.

5.4.2 Returns

Array with up to three elements.

state One of three constants above, This is always returned.

params Array of arguments. Only set if state is 0 (`Dispatcher::BAD_ROUTE`) or 1 (`Dispatcher::FOUND`)

target The target object provided with `AddRoute`. This is a string by anything is allowed.

5.4.3 Example

```
<?php
        // Add the only route, fan-clubs/%slug%
        Dispatcher::AddRoute('fan-clubs/%slug%', 'clubs.php');

        Dispatcher::ResolveRoute('fan-clubs/teen-titans-go');
        /* Returns
        [
                "state" => 1,
                "params"=> [ "teen-titans-go" ],
                "target" => "clubs.php"
```

(continues on next page)

(continued from previous page)

```
        ]*/

        Dispatcher::ResolveRoute('the-bad/egg');
        /* Returns
        [
                "state" => 2
        ]*/
?>
```

CHAPTER 6

MediaQuery

The media query class is used to query media in a slightly less insane way.

The query is broken into little bite modular chunks. Upon calling each method the a part of the query is added to the `sqlParts` array. If any PDO parameters are required they too get added as an array to the `argumentsParts` array.

Once the Query is built `Execute()` is called. It will return whatever the query returns.

6.1 Class Overview

```
MediaQuery
{
        /* Properties */
        protected array $sqlParts;
        protected array $argumentParts;
        protected array $allowedFields;
        protected array $allowedTables;

        /* Constants */
        const int FIELDS = 0;
        const int JOIN_TAGS = 3;
        const int WHERE = 5;
        const int GROUP = 6;
        const int ORDER = 7;
        const int PAGINATE = 8;

        /* Methods */
        public __construct (PDO $dbc)
        public void Fields (array $fields)
        public void Tags (array $tags)
        public void Order (string $field = "date", string $direction = "desc")
        public void Type (mixed $type)
```

(continues on next page)

(continued from previous page)

```
        public void Pages (int $page = 0, int $pageSize = PAGE_SIZE)
        public mixed Execute ()
}
```

6.2 Methods

Fields(array $fields) The fields method sets the columns to select from the database. If any item in the `$fields` array is not in `$allowedFields` an exception is thrown.

Tags(array $tags) Join the *tags* and *tag_map* tables and only return media with any of the provided tags. `$tags` is an array of tag slugs.

Order(string $field, string $direction) Order the query using the specified field and direction. If the field is not in the `$allowedFields` array an exception is thrown. If direction is anything other than *desc* or *asc* an exception is thrown.

Type(mixed $type) When type is a string the query will only get media with the same value in the media.type column. However when type is NULL the query will return media with no value set.

Pages(int $page, int $pageSize) Pagination. It's pretty self explanatory.

6.3 Properties

sqlParts An array that holds parts of an SQL query. The Constants are used as indexes for this array.

argumentParts An array that holds arrays of parameters for the query. The Constant are also used as indexes.

allowedFields This array contains all allowed fields for this query builder. PDO is unable to pass columns/fields and must be done in SQL. This array is used to check for invalid data like SQL injection attacks.

allowedTables This array contains all allowed tables for this query builder. For the same reasons as above.

6.4 Constants

The constants are used to help organize the indices of both the sqlParts and argumentParts.

FIELDS Fields are done first. It's used to create `SELECT media.* FROM media`.

JOIN_TAGS Used next to create inner joins for both the `tag_map` and `tags` table.

WHERE Creates the where parts. This is used in multiple places so both the `Type` and `Tags` components of the query must be able to append it's self to the existing query.

GROUP Groups the data on the `media.id` and is internally used on `JOIN_TAGS`

ORDER Done to order the data.

PAGINATE Finally, the data is paginated into little cute blocks of 250 (by default.)

CHAPTER 7

Indices and tables

- genindex
- search