



The Lenses Framework

a PHP Framework

NOTE This is a "working" document.

By Chris F Ravenscroft

Version 0.1 "beta 1"

Date 10-Oct-2008

Document History

Date	Author	Revision	Description of Change

Copyright Notice

© Chris F. Ravenscroft. 2008 – USA.

Legal Information

To the extent permitted by law, the author accepts no responsibility and excludes all liability whatsoever in respect of any person's use or reliance on this publication or any part of it

Acknowledgements

[This section needs to be filled properly - CFR]

LENSES

CONTENTS

1	Introduction	2
1.1	About this document.....	2
1.2	About Lenses	2
1.2.1	Overview.....	2
1.2.2	Why this framework?	2
1.2.3	Audience.....	3
1.3	Product components.....	3
1.4	References and training	3
2	Installation	5
2.1	Licence agreement.....	5
2.2	Folders	5
2.3	Installation	5
2.3.1	Stand-alone PC	5
2.3.2	Remote web server	6
2.3.3	Preparing the database	6
3	Using Lenses	7
3.1	The Configuration File	7
3.2	Contexts & Namespaces.....	8
3.3	Info/Error Loggers	12
3.4	Controllers	13
3.5	AJAX.....	14
3.6	Views	15
3.7	Active Records.....	16
3.8	Variables Lifecycle And Visibility	18
3.9	PHP Libraries and Helpers	19
3.10	Presentation Helpers.....	19
3.11	CSS Helpers.....	21
3.12	Admin Control Panel and Preferences	23
3.13	Modules	24
3.14	Users Accounts and Sessions Management	25
3.15	Please.....	26
3.16	Migrations.....	27
3.17	Coding Style	28
4	Troubleshooting.....	29

1 Introduction

1.1 About this document

This manual is in its very early stage. I tried to cover as much as possible but it is very likely that I forgot to document some important framework pieces. Please let me know about any such omission.

1.2 About Lenses

1.2.1 Overview

I created this framework for my own needs. I hope you find it suitable.

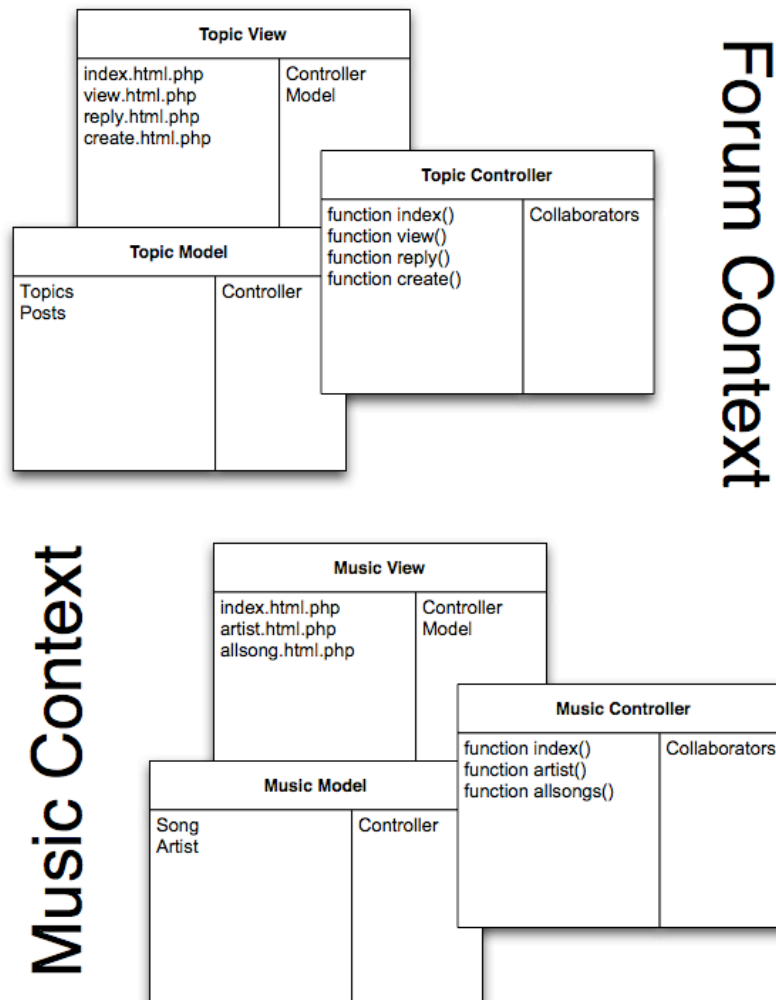
Note that, to run correctly, this framework needs PHP 5. Command-line access is also recommended, albeit not mandatory.

1.2.2 Why this framework?

HMVC

The Lenses Framework is a web development framework that follows a HMVC development paradigm. A HMVC, if you are comfortable with MVC development, aka “Model-View-Controller”, requires a very short learning curve: the H stands for ‘Hierarchical.’

The Lenses Framework can be used to develop following the PAC architectural pattern, or you can decide to use its hierarchical dimension to create isolated MVC contexts



The Library

The library is kept as small as possible. The idea is not to re-invent the wheel. You can find dozens of PHP scripts that will fit perfectly your needs and that is why, rather than forcing you to memorize a whole library, Lenses will try to 'stay out of your way'.

1.2.3 Audience

This document is targeted at web developers with medium or better level of proficiency in programming. Knowledge of web technologies is a must, but you do not need to be an expert in any of these technologies. That's where Lenses helps you!

1.3 Product components

Lenses can be downloaded as a simple .zip file. This file contains the libraries, base classes and helpers as well as a command-line tool. Unzipping the .zip file will automatically create the files you need.

1.4 References and training

At this time, there is no auto-generated framework cross-reference document.

Lenses

You may, however, visit our online board at <http://www.nextbbs.com> for support.

2 Installation

2.1 Licence agreement

This is a preview of a framework as-of-yet unreleased. Permission is granted to you, the user, to use this framework on as many machines as desired. Permission is **not** granted, however, to redistribute this product or any derived work.

2.2 Folders

Lenses's folder hierarchy is as follows:

- .
- app/
 - app/\$context_name/helpers/
 - app/\$context_name/models/
 - app/\$context_name/controllers/
 - app/\$context_name/views/
- helpers/
- models/
- controllers/
- views/
- libs/
 - libs/base/
 - libs/models/
 - libs/modules/
- tools/
- migrations/

2.3 Installation

2.3.1 Stand-alone PC

- Download Lenses .zip file
- Unpack the .zip file in your local web server documents directory
- Create a database for your project
- Edit config.php with the database settings

2.3.2 Remote web server

The steps to install on a remote server are similar to a local install, with the additional requirement that the files need to be installed remotely. This can be done either by unzipping them all locally then uploading them using a file transfer utility, or if you have a remote shell access, uploading the .zip file, then unzipping it remotely.

2.3.3 Preparing the database

After unzipping the PHP and Javascript files, you will need to prepare your database for framework use: run 'please' and type 'migrate up' and you should be all set!

3 Using Lenses

3.1 The Configuration File

In `config.php`, a class, called `Config`, is declared.

Here is what each variable in this class means:

<code>developer</code>	Your email address. This is where Lenses will send distress email messages if configured to do so.
<code>path</code>	The web path to your application. For instance, an application found at <code>http://www.example.com/myapp/</code> will have a path of <code>‘/myapp/’</code>
<code>dblayer</code>	Which database layer are we using? At the moment, only one layer is supported: <code>‘native’</code>
<code>dbengine</code>	Database engine, selected by <code>AdoDb</code> . For instance: <code>‘mysql’</code>
<code>dbhost</code>	This is the hostname or the IP address of the server where our database can be accessed.
<code>dbname</code>	Database name, eg. <code>‘myapp’</code>
<code>dbuser</code>	Database user name
<code>dbpassword</code>	Database user password
<code>salt</code>	Do not forget to change this value. The salt must remain a secret as it is part of all Sha1 encryption performed by Lenses.
<code>cookie</code>	Your website cookie, stored in your users’ browsers.
<code>fullemailcheck</code>	When validating an email address, if this variable is <code>‘true’</code> , Lenses will actually talk to that email address’ declared mail server to check its validity.
<code>logger</code>	Select which conduit to use when writing log messages.
<code>debugger</code>	Reserved.
<code>notifyonerror</code>	Attempt to send an email notification to you when something goes wrong?
<code>altroutes</code>	An array of routes mappings, to change contexts on-the-fly.

Note:	Modifying the value of <code>‘path’</code> is not enough if you are using URL rewriting (highly recommended!) You will also need to fix the last line of your <code>.htaccess</code> file.
-------	--

3.2 Contexts & Namespaces

Contexts

By default, models, views, controllers and helpers are expected to exist in various contexts that are located under `app/`

For instance, if you wish to work with a context named 'dmv', you will store that context's classes in `app/dmv/`

Let's imagine that you create a model, a view, a controller and add a couple helper classes for good measure, all these in the 'dmv' context. Your directory structure will be as follows:

- `app/dmv/`
- `app/dmv/models/`
- `app/dmv/controllers/`
- `app/dmv/views/`
- `app/dmv/helpers/`

You may wish to create a simple application and using contexts seems overkill. In that case, you can tell Lenses that some classes are context-free by mapping them to an empty context in `config.php`. For instance:

```
$altroutes = array(
    'myclass' => "");
```

Classpath

The Classpath is a list of root directories that Lenses will use as a starting point to locate classes as they are instantiated.

Models and Helpers can be found through a scan of the Classpath.

By design, Controllers and Views are not in the Classpath. This is to prevent their direct invocation.

What does this mean? Simply that every time that you access a context, including the default (empty) context, its `models/` and `helpers/` directories are added to the list of directories to be scanned for classes.

Namespaces

Namespaces exist both at the application root (main context) and in the contexts that you declare.

Models are automatically available in the Classpath, therefore they do not need to be imported in the controllers.

Helpers, however, have to be "namespaced" and as such they need to be imported.

An example:

You are the owner of a Las Vegas casino called 'The Lucky House'.

You create a context called 'casino' and decide to add a model called 'Roulette' and a helper 'Croupier'

Your directory structure will look like this:

- `app/casino/models/roulette.php`
- `app/casino/helpers/com/theluckyhouse/casino/croupier.php`

The namespace shown here is arbitrary. You could decide, for instance, to simplify it to:

- `app/casino/helpers/casino/croupier.php`

In your controller, found in `controller_casino.php`, you can use Roulette directly because models are in the classpath by default:

```
$roulette = new Roulette();
```

To use a 'Croupier' helper, however, you need to import it first:

```
import('com.theluckyhouse.casino.croupier');  
$croupier = new Croupier();
```

Tip:	Let's say that the Croupier helper offers a method for rendering a roulette table. You may want to make the \$croupier instance available to your view. It is as easy as actually making it an instance variable. Your code will then look like:
------	--

```
$this->croupier = new Croupier();
```

Router

The built-in URL router will automatically invoke the controller that matches your URL structure. The first field after your application's path will be the current context. The next field will be the controller name. The next field will be the name of the controller's action to invoke. All subsequent fields are arguments.

- url/controller/action
- url/controller/action/arg1/...
- url/context/controller/action
- url/context/controller/action/arg1/...

Avoid or substitute context by specifying in config.php in Saltroutes

Language Extensions

import(\$path)

Import the content of the given \$path; for instance:

```
import('com.voilaweb.core.files')
```

will look through your whole classpath for com/voilaweb/core/files.php

```
import('com.voilaweb.core.*')
```

will look for your whole classpath for files in com/voilaweb/core/

include_here(__FILE__, 'filename')

This is a convenience function that you should almost never need to invoke. It allows you to split big segments of code across multiple files without having to worry about absolute path versus relative path.

Let's imagine that you are refactoring a big source file, called example.php, by splitting it in smaller files.

Original file :

```
<?php  
function func1() {....}  
function func2() {....}  
function func3() {....}  
?>
```

Refactored file:

```
<?php  
include_here(__FILE__, 'example_func1.php');  
include_here(__FILE__, 'example_func2.php');  
include_here(__FILE__, 'example_func3.php');  
?>
```

In this example, you do not have to worry whether you are refactoring a model, a helper or another class type: the included files are expected to be in the same directory as your master file and that is your only constraint.

Mixins

According to Wikipedia: "a mixin is a class that provides a certain functionality to be inherited by a subclass, but is not meant to stand alone. Inheriting from a mixin is not a form of specialization but is rather a means to collect functionality. A class may inherit most or all of its functionality by inheriting from one or more mixins through multiple inheritance."

Here is an example, using Lenses:

```
class MixMe  
{  
    function a($c)  
    {  
        print "B($c)<br />";  
        return "You passed $c";  
    }  
}  
  
class TestController extends ApplicationController  
{  
    function testmixin()  
    {  
        $this->mixin(MixMe);  
        print $this->a('hello');  
    }  
}
```

Visiting `/test/testmixin/` will display:

```
B(hello)
You passed hello
```

3.3 Info/Error Loggers

Select your logger by changing the value of 'logger' in config.php

NullLogger (value: 'null')

Log messages will be ignored. This is the default logger.

FirePHPLogger (value: 'firephp')

Log messages will be displayed in your browser's status window.

To use this logger, you need to use the Firefox web browser with the FireBug and FirePHP extensions installed.

TmpLogger (value: 'tmp')

Log messages will be appended to /tmp/fw.log

EchoLogger (value: 'echo')

Log messages will be displayed in the current page.

Currently, the logger API is very simple, if not simplistic:

```
Logger::instance()->write($str)  
Logger::instance()->info($str)  
Logger::instance()->error($str)
```

3.4 Controllers

Controllers are the bricks, in your design, that connect models with views: they get the information from the models and pass it to the views for display purpose.

Tip:	You can, anywhere in your controllers, throw <code>GoodException</code> to pop out of the current controller's code and move to processing of the view.
------	---

3.5 AJAX

You can export AJAX methods directly from your controllers:

In the controller file, create a static method.

After your class definition, call

```
ajaxExport(ClassName, methodName)
```

Sample code:

```
class UserController extends ApplicationController  
{  
  static function checkUsernameAvailable($username)  
  {  
    return ($username != 'chris');  
  }  
}  
ajaxExport(UserController, checkUsernameAvailable);
```

In the view file, add this javascript code:

```
ClassName$methodName(parameters, callback)
```

We need a callback because we cannot test the result of our call directly: it is being performed asynchronously, therefore the result will be returned later (to the callback function!).

Sample code:

```
function check_availability()  
{  
  UserController$checkUsernameAvailable('john', check_availability_cb);  
  return false;  
}  
function check_availability_cb(avail)  
{  
  alert(avail ? 'Available' : 'Not available');  
}
```


3.6 Views

The first thing you need to know about views is that their display is decided in the controllers.

In your controllers, you can set the content of a couple arrays and this will affect what views are displayed.

\$page_parameters

action	Used to ask Lenses to change the action, as far as the view is concerned: action = 'redirect': use redirect() rather than display a view action = 'notify': real-time update of the view notification area using Javascript action = 'api': use this value when providing a web service
--------	--

\$options

controller	Get view from another controller
view	Display different view page
header	Display alternate header <i>Existing header pages:</i> header.html.php (default) admin_header.html.php (default for admin pages) partial_header.html.php include style sheets and javascript and message, if any
body	Display alternate page body <i>Alternate body:</i> empty.html.php
footer	Display alternate footer <i>Existing footer pages:</i> footer.html.php (default) admin_footer.html.php (default for admin pages) partial_footer.html.php send minimal footer

3.7 Active Records

All model classes inherit from an active model, thus providing easy CRUD ('Create, Read, Update, Delete') access. Currently two default active models are available:

- **ActiveRecord** for easy access to many database engines (MySQL, Sybase, ODBC...)
- **ActiveWSRecord** provides the same level of encapsulation for web services.

Both implement the **ApplicationModel** interface. Any model you use should eventually implement this interface,

Note:	You can choose not to inherit from an active model when creating your models. Subclassing an active model makes using CRUD functions easier but is not a requirement.
-------	---

ActiveRecord:

Relationships: *belongsTo* and *hasMany*

ActiveWSRecord:

This class offers convenience methods for accessing Restful web services.

Requests can be performed using GET or POST.

Results can be parsed using JSON or XML.

From conversation with John Lim, author of AdoDb:

“

Let's consider the example of a message board: your users can create topics and each topic will contain one or more posts.

If you look at the definitions of Topic and Post, you will see that their 1->n domain relationships are defined in their respective constructor.

The syntax is very simple: a post *belongsTo()* a topic because each topic *hasMany()* posts. Sometimes 'n' is '1', for instance a topic *belongsTo()* a member.

If we wanted, in listtopics.php, to display a quick preview of all the posts contained in each topic, as a topic is listed, it would be easy to get this information if we had declared in Topic's constructor that topic *hasMany()* posts.

OK, this sentence was a tad confusing. A more palatable example: let's say that you create a page that will display all the topics created by a user (member). You could simply tell Member that it *hasMany()* posts; each time we retrieve this Member instance, we would also retrieve the member's topics list. Well, at least we would have the possibility to retrieve it, see below: 'join v lazy v worker'.

None of this has any effect on the database's internal structure, therefore we can decide to apply these relationships or not, depending on the current context.

join v lazy v worker:

- If you use the 'join' mode, you will retrieve a *lot* of rows because, under the hood, the relationships will be expressed using join statements. OK, it's many programmers' favourite way of doing things ("Yay! I saved a query!"). Of course, there is a price to pay, in terms of query complexity, performance and having to clean up the redundant rows afterwards. Oh, and don't think of using LIMIT.
- If you use the 'lazy' method, you're actually going to have your code issue a lot of queries to the database. There are many use cases when that's what you would want to do. For instance, why retrieve a complete relation tree when you don't know in advance what the

user will really want to see? Yes, you may use more queries but the effort can be spread over a fairly long period of time, as opposed to the brute force method used by join.

- Finally, my favourite: worker (abbreviated 'work'). This method, which is the last one I implemented, is nice because you can tell in advance how much it is going to cost you. The number of queries that your code will issue is 1 + number of expressed relationships. For instance, a post *belongsTo()* a topic and it also *belongsTo()* a member. Therefore, when you retrieve a list of posts for a given topic, the number of queries is going to be 3. Sure, this is more than 1 but if you issued 1 query, using the join method, it would be a query that involves three tables, consumes more memory, may or may not use covering indices and you will find it difficult to page correctly. So, yes, that's why it's my favourite method and I've made it the default method in the Adodb AR class.

Of course, more relationships could be implemented, such as many-to-many (*HasManyAndBelongsTo...*) but I would think that these two guys fit the 80% approach.

3.8 Variables Lifecycle And Visibility

All instance variables declared in a controller are made available to the corresponding view. Syntactically, this means that a variable referred to using `$this->varname` will be available to the view as `$varname`.

Arguments

When in a controller, you can retrieve all the current URL information through:

```
$this->_controller  
$this->_action  
$this->_args
```

In a view, you have access to:

```
$controller  
$action  
$args
```

Note:	Whenever possible, you should limit access to <code>\$args</code> to the header and footer elements.
-------	--

3.9 PHP Libraries and Helpers

If you look into the `libs/` directory you will find the libraries used by Lenses; these are also available to you.

You may be interested in such libraries as:

Inflector, Validator, Session...

The helpers that ship with Lenses can be found in `helpers/com/voilaweb/core/`

Currently available helpers are:

- Files management ('neuter', upload, create thumbnails...)
- Dates formatting
- Text handling (escape, format...)
- Modules management (load)
- Undo pattern
- Security (captcha)
- Pagination

3.10 Presentation Helpers

These functions are available to controllers and views and allow you to create web pages using semantic meanings rather than HTML tags. Of course, the generated code will be HTML.

```
function url_for($url)
function link_to($text, $uri, $options = array())
function button_to($name, $uri, $options = array())
function mail_to($name, $uri, $options = array())
function tag($name, $content = null, $options = array(), $type = TAG_SELF_CONTAINED)
function form_tag($url = "", $options = array())
function label_for($id, $label, $options = array())
function input_tag($name, $value = null, $options = array())
function input_password_tag($name = 'password', $value = null, $options = array())
function input_hidden_tag($name, $value = null, $options = array())
function input_file_tag($name, $options = array())
function textarea_tag($name, $content = "", $options = array())
function rich_textarea_tag($name, $content = "", $options = array())
function submit_tag($value = 'Save changes', $options = array())
function submit_image_tag($uri, $options = array())
function reset_tag($value = 'Reset', $options = array())
function click_tag($text, $callback)
function img_tag($uri, $options)
function click_img_tag($uri, $callback)
function constrained_img_tag($uri = "", $options = array())
function options_for_select($options = array())
function select_tag($name, $option_tags = null, $options = array())
function radiobutton_tag($name, $value, $checked = false, $options = array())
function checkbox_tag($name, $value = '1', $checked = false, $options = array())
function util_checkbox_tag($name, $value=1)
function enable_select_multi_tag($name)
function reveal_tags_tag($name, $label = null, $options = array())
function flash_upload_form_tag($uri = "", $options = array())
function attachments_tag($frame)
function emoticons_tag($frame)
```

Additionally, Lenses provides these three functions that you will most likely only call from your controllers:

Lenses

redirect(\$url, \$redirect_message)

Issue redirect headers that will take your user to the url specified in \$url; if your headers are defined to display messages, the content of \$redirect_message will be displayed.

addheader(\$html)

The string passed to this function will be added to the <header> area of your web page. You can call this function as many times as you wish.

addjs(\$js)

The string passed to this function will be added to the <header> area of your web page, allowing you to run Javascript before parsing the <body> area. You can call this function as many times as you wish.

3.11 CSS Helpers

You will find, in equal numbers, professional web designers who advocate the use of a CSS framework for reasons such as consistency, ease of use, etc., and professional web designers who will tell you of the evilness of CSS frameworks, since they introduce their own quirks, make designers lazy, etc.

I believe that providing a simple CSS framework is the right approach: no strings attached, ease of use are the two characteristics I wish to define the CSS helpers described in this chapter.

THE LENSES FRAMEWORK's CSS smarts are located in `views/assets/utills.css`

Take a look at the content of this file, it is pretty self-explanatory.

Some of the features provided:

- Color message boxes
- CSS Reset
- Layout: integrates in existing page
- Column layouts: 10%, 20%, 25%, 30%, 33%, 40%, 50%, 60%, 66%, 70%, 75%, 80%, 90%, 100%
- Column layouts: float left, float right
- Rows layout: row_top to 'float' content to the top
- Flow break

You may have noticed that a feature is glaringly absent: typography support. This may be added in a future release but I do not see a need for it at this point.

Unit testing of the CSS framework is performed using a page that is a copy of the demo page provided by the author of the CSS framework 'Typogridphy'

Here is what typical CSS code would look like (extracted from that unit test):

```
<div class="column_20_left">
  <h2>20 percent</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. In accumsan diam vitae
velit. Aliquam vehicula, turpis sed egestas porttitor, est ligula egestas leo, at interdum leo
ante ut risus. Cras ut nunc. Phasellus imperdiet, urna in volutpat venenatis, mauris pede
euismod pede, malesuada euismod turpis enim at.</p>
</div>
<div class="column_80_right">
  <h2>80 percent</h2>
  <blockquote class="column_50_right row_top"><p class="row_top">&ldquo;This is
a blockquote, with a right float and a width of 50 percent. Even <em>this</em> maintains
typographical flow!&rdquo; &mdash; <cite>Harry Roberts</cite></p></blockquote>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. In accumsan diam vitae
velit. Aliquam vehicula, turpis sed egestas porttitor, <strong>est ligula egestas leo</strong>,
at interdum leo ante ut risus. Cras ut nunc. Phasellus imperdiet, urna in volutpat venenatis,
mauris pede euismod pede, malesuada euismod turpis enim at arcu. Sed placerat accumsan
mi. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.
Aliquam suscipit, quam at placerat interdum, quam turpis placerat pede, eget sagittis arcu
sapien vel mauris. Nulla faucibus turpis eget tellus. Integer mattis dapibus lorem. Donec
pretium lorem vel ligula. Fusce semper consequat dui. Integer ac mi. Cras faucibus nulla quis
ipsum. Nunc augue turpis, tristique et, hendrerit id, hendrerit eu, lacus.</p>
</div>
<div class="break"></div>
<div class="column_100">
  <h2>Full Width</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. In accumsan diam vitae
velit. Aliquam vehicula</p>
</div>
```



3.12 Admin Control Panel and Preferences

Lenses comes with its own admin control panel that you can customize to support any kind of settings.

Using the '`please`' utility it is possible to create new settings that the control panel will automatically know how to handle.

You can add new forms to the admin control panel using simple definitions.

3.13 Modules

Modules are classes that allow you to offer alternative pieces for given functionality. For instance, the text editor used in a textarea can be handled as a module.

Modules are declared in the database tables ‘modules’

The table format is:

type	A letter <i>Currently used letters:</i> ‘E’ for editor
path	Module name – will be used to locate the module management class in libs/modules/\$name/\$name.mod.php
default	A Boolean value: when true, this module is considered the default module for the given type.

Writing a Module

Quite often, a module will be used to wrap an existing third-party application. For instance, the ‘nicEdit’ module is a wrapper for the ‘nicEdit’ rich-text editor package.

Store your whole package in libs/modules/\$name/

Create your module wrapper class: libs/modules/\$name/\$name.mod.php

Lenses, when invoking your wrapper, will expect to find a function called `getModule_$name()` which will return the wrapper class.

Let’s have a look at nicEdit:

It is located in libs/modules/nicEdit/

The wrapper code is in libs/modules/nicEdit/nicEdit.mod.php

nicEditor.mod.php provides a function called `getModule_nicEdit()` which returns an object that implements `Editor`. That object provides a single method: `present($areaId)` which will be invoked to add content to the current web page.

3.14 Users Accounts and Sessions Management

You can register new accounts. Registration can be open or require an invitation. Of course, a mechanism for requesting invitations is also provided.

Members can log in and their session is automatically maintained in the database.

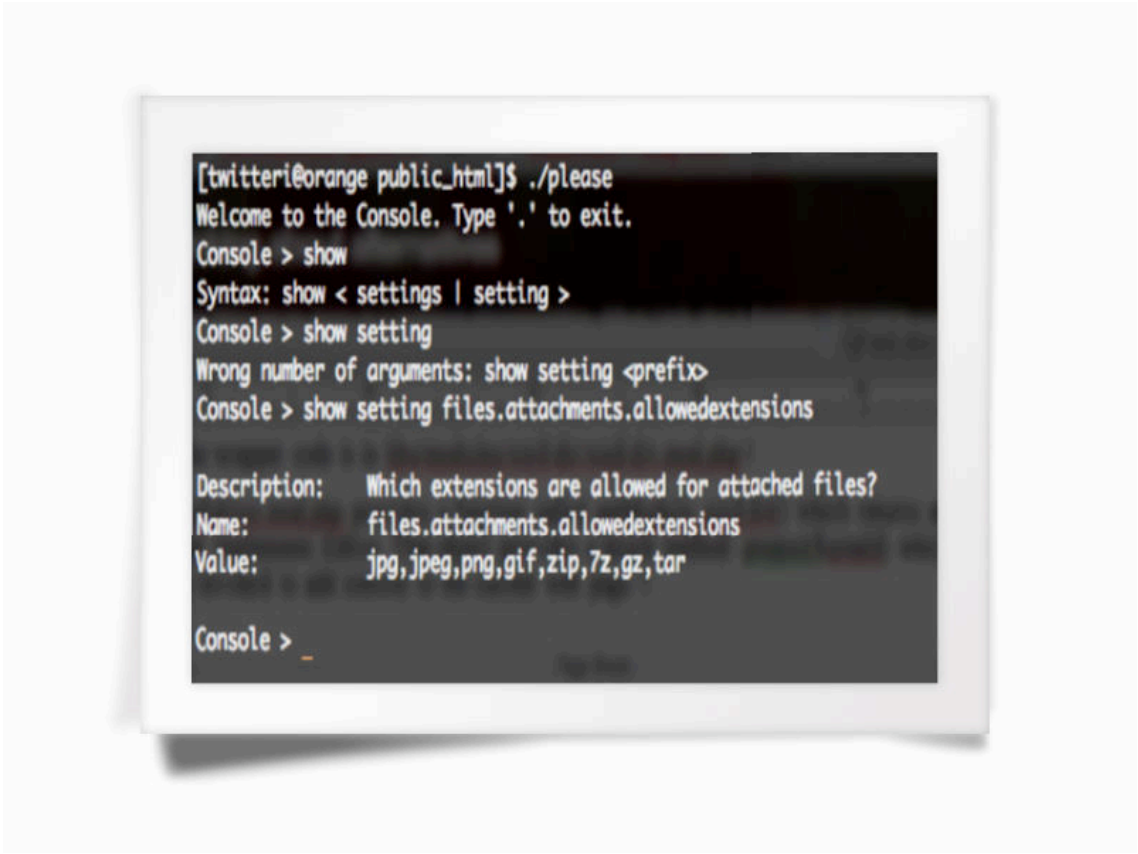
3.15 Please

'please' is an interactive console shell that speeds up the manipulation of entities.

Here is a list of commands it understands:

create		
	model	Create a new model class
	controller	Create a new controller class
	view	Create a new view directory and a default index page
	mvc	Create a model, a controller and a view
	helper	Create a new helper
	setting	Create a new setting
delete		
	setting	Delete a setting
set		
	setting	Set a setting's value
show		
	setting	Show a setting's value
	settings	Show all settings
.		Exit console

Tip: You can hit [Return] or [Enter] before entering all the parameters to get more information on a command's syntax.



```
[twitteri@orange public_html]$ ./please
Welcome to the Console. Type '.' to exit.
Console > show
Syntax: show < settings | setting >
Console > show setting
Wrong number of arguments: show setting <prefix>
Console > show setting files.attachments.allowedextensions

Description: Which extensions are allowed for attached files?
Name:       files.attachments.allowedextensions
Value:      jpg,jpeg,png,gif,zip,7z,gz,tar

Console > _
```

3.16 Migrations

Lenses comes with a mechanism for migrating your database data. By ‘migrating’ I mean “upgrade from version n to version $n+1$.”

Migration files are stored in migrations/ and are named 001.yml, 002.yml, etc. The .yml extension is used because these files are authored using YAML syntax. It is a very intuitive syntax that allows you to define your schema as you jot it down.

Upon install, you will notice that a file called 001.yml already exists. It is a file that you must use to ‘migrate’ your database after installing Lenses.

3.17 Coding Style

This section will obviously need some work. However, as already stated, this framework's purpose is to “stay out of the way” most of the time, therefore this document will not dictate strict guidelines.

So, what do we have so far?

- Whenever possible use "null objects" (eg `array()`) rather than null values.
For instance:

```
function myfunction($arg = array())
```

rather than:

```
function myfunction($args = null)
```

- When creating a link, it should end with a forward slash (`/`)

4 Troubleshooting

New material will be added to this section as common problems surface.

Error! No text of specified style in document.