FALCODE WEB DEVELOPMENT FRAMEWORK

USER GUIDE

Table of Contents

Introduction

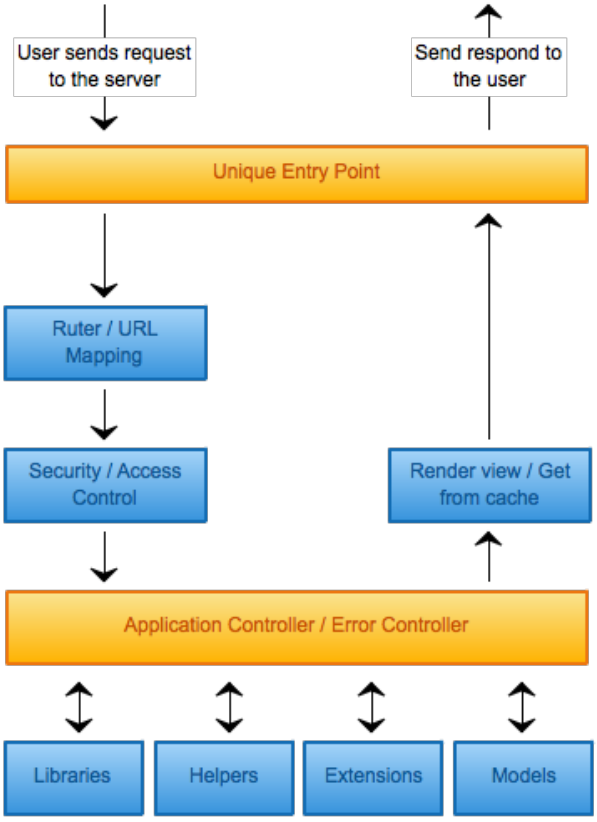
FALCODE is a build-in web development framework that implements the Model View Controller programming architecture for creating robust web applications.

FALCODE is built on top of the leading web development technologies such as PHP, MySQL, Apache, HTML 5 and Javascript.

The following user guide and tutorial assumes that you:

- Have basic knowledge of HTML and CSS
- Have intermediate to advanced knowledge of PHP
- Have basic knowledge of the Apache configuration (.htaccess)
- Have intermediate knowledge of Javascript
- Have basic knowledge of the jQuery Javascript Framework
- Have basic knowledge of SQL

Backend Process Architecture



1. Unique Entry Point: This is the index.php file, it is the only framework file that can be accessed directly, and all requests come through this file.
2. Router/Mapping: The system analyzes and parses the HTTP request and determines which module needs to be executed.
3. Access control: Security determines if the current user has access to the requested module and/or section of the application.
4. Application/Error Controller: If the user has access, the module controller is executed. If not, an exception is raised and a specific error controller is called to handle the request. The controller has access to all the system libraries (core), models, helpers, drivers and extensions explained later on.
5. The controller can specify a view, which will be rendered and displayed to the user. The view can be cached for further requests.

Front-end Process Architecture

Once the request is served and sent to the user browser, the front-end process goes as follows.

1. Load the requested CSS files
2. Declare the framework JavaScript global objects
3. Load the core JavaScript libraries and extension libraries required for the view
4. Load and execute the global JavaScript event handler file
5. Load and execute the module specific JavaScript file

Terms and Definitions

- MVC: Model View Controller. A programming architecture for separating application logic from application views.
- Model: An object representation of a database entity as specified in the database abstraction layer standard.
- View: An html file that is going to be displayed to the user. Can be a whole web page of can be a section of a web page. The standard output format is HTML but can be any other format such as an XML, JSON, a PDF file or any other file format.
- Controller: A controller is a piece of code, which is executed as a result of a HTTP request.
- Module: For better navigation and organization, the application is divided into several “sections” or “modules”. Each of which has specific controllers and views.
- Action: Each module has one or more “sub-sections” or “actions”
- Helper: A helper is a library of plain functions of certain topic used as “toolkit” and that is not used in all the modules.
- Extensions: All the libraries that are not part of the system core are called Extensions and can be called from every controller.
- Drivers are system libraries that can vary depending on the software being used in the web server.
- The dates in FALCODE are handled always as GMT dates. By default, all dates stored in the database are previously converted to GMT. This is very important as the server first determines the time zone based on the user’s browser configuration and then makes the corresponding dates transformation for displaying accurate date calculations. When having more than one server hosting your application it’s very important to prevent date time inconsistency.

URLS

In FALCODE by default all URLs are generated to be human readable and search engine friendly. Therefore, all URLs used in the system are strictly absolute.

The URLs have the following syntax:

http://[subdomain].domain.com/[module[.controller]][/id][/path_1]...[/path_n][/var_1,val_1]...[/var_n,val_n]

For example lets say you have a module named “news” and you want to “view” the article with the id number 10 and pass a get variable “referrer” with the value “home”:

http://www.domain.com/news.view/?id=10&referer=home

Or we can pass the GET variables as follows:

http://www.domain.com/news.view/id,1/referer,home/

Or if “view” section inside “news” module is the default action, then:

http://www.domain.com/news/id,1/referer,home/

Or if the “id” variable is numeric, we can pass it as follows:

http://www.domain.com/news/1/referer,home/

If you don’t want to use SEO friendly URLs you can always use plain simple GET syntax as follows:

http://www.domain.com/?i=news/view&id=10&referer=home

As you can see you can pass the module and control through the “i” variable. You can pass it in POST as well. The syntax when sending via “i” variable is i=module/action

Hash or URL segments

The URL segment or HASH is the part of the URL which is right to the # symbol.

As you may already know, hash urls are now allowed to be passed to the server, mostly because the hash is mainly used in the frontend by javascript or HTML anchors. This means that if you access the following URL http://www.google.com/#hashvalue, google servers (supposing they are Apache servers) will never know that you passed a hash tag “hashvalue” in the URL.

Nevertheless, many times we may want to pass the server the hash value in our current URL. This can be done by calling the addToForm() function of the Hash javascript super object: Hash.addToForm(). We'll cover the super objects later on.

This what mainly does is add a __hash GET var into the form. And that is because FALCODE detects if there is a REQUEST var named "__hash" and if it does, it appends it's value as the hash part of the URL inside $_SERVER['HTTP_REFERER'] superglobal.

Note: You can always pass the __hash directly using POST or GET instead of calling the Hash.addToForm() function.

Generating absolute URLs

It would be very painful and time consuming to generate all the absolute URLs manually, that's why there are several methods to generate absolute URLs in FALCODE which will be explained later on.

Controllers And Modules

What is a Module?

In order to keep everything within you app more organized, your code needs to be divided in functional parts, for example if you are designing a blog application, you will probably have a "posts" module and a "users" module. Each of which needs to perform several specific actions such as: create a post, show a post, show the list of posts, create a new user, validate the user, and so on.

Can you imagine a code structure in which all functions or "actions" or the whole application are stored in the same location? It would certainly have a very poor scalability and maintainability. That's why we encourage the use of modules.

What is a Controller?

The controller is simply a class that hosts all the possible "actions" or "functions" a certain module can perform. For example in the "users" module, we can imagine that basic "functions" of that module would be: create a new user, delete an existing user, show the users list, etc. Each of those functions needs to be stored in methods or "functions" inside the controller class.

Take for instance the following example. Create the folder "test" inside controller/default/. Then create a PHP file inside it and name it ModuleController.php with the following content:

```php
<?php
class ModuleController extends Controller{
    public function __construct(){
        // Do nothing
    }

    public function hello(){
        echo "Hello world!";
    }
}
?>
```

Now save the file and access the following URL:

http://yourinstallation.com/test.hello/

If everything is OK you will se "Hello world!" on your browser.

The class must always be named "ModuleController" and must always extend to the parent class Controller.

Actions

In the previous example we have already created and accessed one action named “hello”. As you can see, creating actions is no more than creating functions inside the module controller class.

As you have already noticed, the action name must coincide with the function name (it IS case sensitive so be careful)

Important: The “__construct” function cannot be accessed through a HTTP request be MUST ALWAYS be present in the class. If you need some code to be executed regardless the action that was called, you can place that code inside the constructor function.

In cases your action name matches with a reserved PHP world (for example “list”), you can always add the suffix “Action” to your function name. For example instead of:

public function list(){} // This will give a PHP error

You can use:

public function listAction(){} // This is OK

NOTE: The suffix “Action” must not be added when calling the action in the HTTP request.

Defining a Default Action

You can tell the system to execute a default action when the action is not specified in the request, for example using the previous controller if you try the following url you will get an error:

http://yourinstallation.com/test/

This is because we haven´t specified a default action to execute when no action is specified.

When no action is passed, the system will always try and execute the main() function inside the controller.

```php
<?php
class ModuleController extends Controller{
    public function __construct(){
        // Do nothing
    }
    public function main(){
        echo "This is the default action!";
    }
    public function hello(){
        echo "Hello world!";
    }
}
?>
```

Now we can access the main() function by entering:

http://yourinstallation.com/test/

Or

http://yourinstallation.com/test.main/

Alternative method

You can define a default action to execute other than using the main function method. You can do this by calling the $this->setDefaultAction("function_name") function inside the __construct() function of your ModuleController.

This way you can define a series of actions and simply tell FALCODE which function to execute if there is no action set.

NOTE: In both methods, the action will still be empty.

Getting URL Paths and ID

URL paths are all URL segments that are neither module or action or id or GET vars. They are simply words separated by slashes. In the following example, I'll explain to you all the URL segments.

http://www.domain.com/news.list/510/archive/may-2012/order,date/?search=bin+laden#listing

- Module
- Action/Control
- ID (Must always be numeric, else will be considered a path)
- URL Paths
- GET vars (In this case are passed in two different ways)
- HASH

Getting URL paths

There are two ways of getting the url paths.

First method

The first one is getting them right from the Router class as so:

Router::$path

This is a one dimensional static array inside the Router class which contains the paths (if they exists) passed through the URL. They are added in order of appearance. For the example above, the $path array would be:

Router::$path = array('archive','may-2012');

So if you want to access the "archive date" information passed through the URL path then you would:

$date = Router::$path[1];

Second Method

The second method is much more simple and more straight forward. It consists of getting it directly as the action function arguments

```
public function test($id,$archive_type,$date){
	echo $id; // This will only be passed if there is a numeric path
	echo $archive_type;
	echo $date;
}
```

In this example, the variable $archive_type will be assigned the value "archive" and $date will be asigned "may-2012".

In case you expect more path variables, you can always just create more arguments in your action function each of which will be populated with the corresponding value. Remember that the values will be assigned in order of apearance.

Note that ALWAYS the first argument of the action function will be populated with the GET or POST var named "id" OR the FIRST numeric path, which will be considered an ID and therefore an "id" http request var will be created. The logic is as follows:

1. Is there a POST var named id? If so, $id = $_POST['id']
2. If not, is there a GET var named id? If so, $id = $_GET['id]
3. If not, is there a numeric path in the URL? If so, $_GET['id'] = [that path],$_POST['id'] = [that path], $_REQUEST['id'] = [that path], $id = $_GET['id'].
4. Else, $id = null

Private Functions

You can always create private function inside your ModuleController. This functions WON'T be served from the URL request. The only requirement is that you set the function as private.

Reverved Action Names

There are several function named that will cause conflict if used. They will be listed later on the Reserved Names section.

Error Controllers

The error controllers are a variation or a normal controller except instead of module actions, it defines exception actions to be executed in case errors are raised throughout the normal execution process. The exception triggers are explained later on the Controller class chapter.

The default error controller is located in controller/default/ErrorController.php, and has the following structure:

```
<?php
class ErrorController extends Controller{
    public function ModuleNotFound($message,$module_requested){ }
    public function ActionNotFound($message,$action_requested){ }
    public function AccessDenied($message){ }
    public function Validation($arr){ }
    public function UnderMaintenance($message){ }
    public function CustomError($message){ }
    public function UnknownError($message){ }
}
```

ModuleNotFound($error_message,$requested_module)

This action is going to be executed when the requested module the user was trying to access does not exist in the application. This is the equivalent of a 404 http error. $error_message variable will have a system error default message and $requested_module has the invalid module name.

ActionNotFound($error_message,$requested_action)

This is similar to the previous one just that in this case the requested action is the one that does not exist.

AccessDenied($error_message)

This action gets executed when the user tries to access a module or action he's not allowed to access.

Validation($error_fields_array)

This action is executed when the input fields failed a validation process.

UnderMaintenance($error_message)

This action is executed when raised a specific exception triggered manually by the normal module controller

CustomError($error_message)

This one gets executed when raised the exception manually from the normal module controller

UnknownError($error_message)

Triggered by the system when an unknown error is raised

Overriding The Default Error Controller File

The default error controller can be overriden at a module level by creating an ErrorController.php class with the same structure inside the module folder. This one can have all the action functions except ModuleNotFound().

Error Controller Views

All error views are located inside content/templates/[active_template]/_errors/. There you can create your custom error view and load them using:

$this->load->view("error_view.html");

The error views use for default the same layout you where using at the moment the exception was raised in the module controller but you can change it at any time. In fact you can create a specific layout for error views and load it:

$this->load->layout("error_layout.html");

Rising and Handling an Exception

Lets take for instance the following ModuleController located in controller/default/secure_area/ModuleController.php

```
<?php
class ModuleController extends Controller{
       public function __construct(){}
       public function secret(){
              if(has_access() == false){
                     $this->throwAccessDenied("You dont have access");
                     // Which is exactly the same as doing the following:
                     // throw new ControllerException("You dont have
access",ControllerException::ACCESS_DENIED);
                     echo "Test"; // ← This line is NEVER going to be executed
              }
       }
```

Let's say that the has_acces() is a function that evaluates if the user has acces to the location based on a session check… Now if the user doesent have access a ControllerException exception will be raised, the first line is a shortcut to the second line which is raising manually the complete exception. Note that in case the exception is raised, NO FURTHER LINE IN THE MODULECONTROLLER IS GOING TO BE EXECUTED. At this point the execution jumps to the error

function handler action inside the ErrorController. That's why in the comment of the "echo" line, it says that line doesent gets executed ever.

Now the system would determine which error action handler function to execute, in this case it will first look for an ErrorController.php file inside the current module. If not found it will try and execute the AccessDenied() function inside the default ErrorController class.

Reserved Names

Controller Action Names

- setDefaultAction
- title
- blank
- render
- includeController
- memcached
- load
- config
- lang
- system
- session
- user
- db
- request
- response
- throwAccessDenied
- throwUnderMaintanance
- throwCustomError
- throwValidationError
- locateView

Class Names

- AccessControl
- Acl
- ActiveRecord
- Auth
- Cipher
- Config
- Controller
- ControllerException
- Core
- Db
- Image

- Lang
- Layout
- Loader
- Mail
- Map
- Paginator
- Request
- RequestVar
- Response
- Router
- Session
- SiteMap
- SiteNode
- Sys
- TaskPipeline
- TemplateEngine
- Tpl
- Uploader
- User
- UserRole
- UserRoleRegistry

Global Function Names

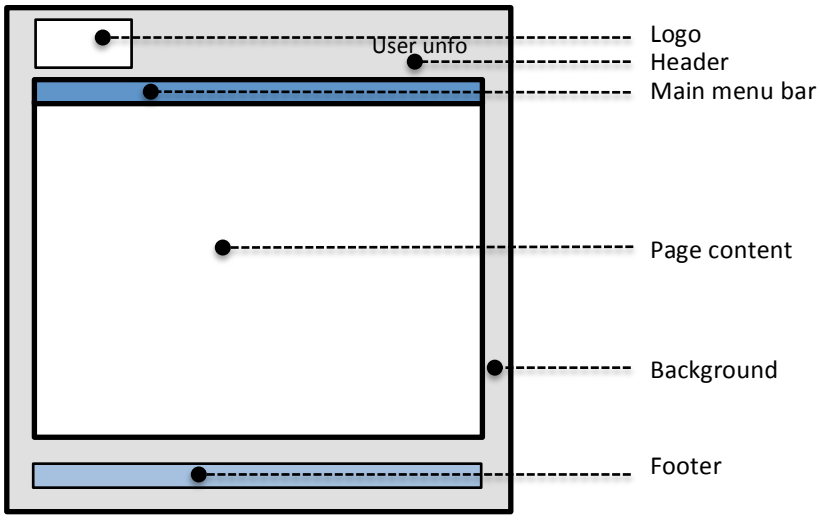
- get_include
- redirect
- get
- not_null
- is_empty
- url
- now_gmt
- time_gmt
- date_to_gmt
- date_to_local
- log_event

Views

A view is everything that is passed out as an output to the user's browser. Most views consist of formatted HTML but can also be an XML report, a RSS feed or even a JSON formatted asociative array.

HTML views

The HTML views are the most important as most of them are of this kind. Before getting in how to load a view and send it as an output, it's very important to study the structure of a view.



It's safe to say that at least 90% of the web pages today have a similar approach to the structure above. And you will also notice that in most of the cases the ONLY thing that changes thoughout the navigation in the page is the "Page content" section. That means that the logo, header, menu bar and footers remain constant in all of the pages.

Now, if we were to make a web page with the described structure for a client that needs will host 20 different pages, do we need to copy and paste those static elements in every single page? Or do we need to include the header and footer in every page? It wouldn't be very efficient.

And what if we needed to present a mobile version of our web page? Do we need to code the whole site again? The answer is not. If the only thing that needs to be changed is your views, then nothing else should be re-coded. That's why in FALCODE we created ***Templates***.

Templates

A template is a series of HTML views, CSS and Javascript that present your site in certain fashion or look & feel. It is the "theme" of your site. As we know, there are many ways in which you can present the same data, in fact, the actual data doesen't change, the only thing that changes is HOW you present it.

FALCODE allows you to have many templates for a single site, each template needs to be inside content/templates and has to have this minimum structure:

- html
    - module_name
        - action_view.html
        - another_action_view.html
    - another_module
        - another_action.html
- images
- css
- js
- layout.html

The html folder contains all the HTML views that are going to be loaded from the controllers. The subsequent folders within are grouped and named after the module name in the controller. It's important that the names coincide.

Not all modules need to have a html folder and the action file (.html) may or may not coincide with the action name inside de ModuleController, it's optional.

Changing the default template

In FALCODE the default template is named "default" (That one was hard, right?). But there are some cases in which you will need to user other templates and to do so you can simply call:

$this->load->template("template_name");

inside your module controller.

Or if you want to permanently change the name of the default folder, you can easily do so by changing the $config['tpl_default'] variable inside the default configuration file in engine/conf/config.php

Layouts

In FALCODE we will consider those static elements explained earlier, the ones that remain the same from page to page, to be the page ***Layout***. A complex web development can certainly have more than one page Layout and thousands of views.

Every template must have at least one layout file.

A layout is also a view, but an "incomplete" view. It is a view that needs *another view* within to present a complete view.

By default, FALCODE has a black layout, that means that it doesent have any header, footer, logo or anything, and therefore the page content will be the complete view presented to the user.

Changing the default layout

By default, FALCODE uses the layout named layout.html which is located in content/templates/[template]/layout.html

If you were to load a different layout other than layout.html, you can do by calling the following inside your module controller:

```
$this->load->layout("layout_file.html");
```

If you want to permanently change the default layout you can do so by changing the $config['tpl_default_layout'] variable inside engine/conf/config.php

Loading Views

To load views from the ModuleController action you can do so by calling the following code:

```
$this->load->view("action_view.php");
```

FALCODE will now search for the following file content/templates/[active_template]/html/[active_module]/action_view.php

Generating Dynamic Views

Now comes the fun part where we can pass variables to our view. Take for example the following html view content/templates/default/html/test/main.php:

```
<h1>My First View</h1>
<p>Hi, <?php echo $name ?>, welcome to my site!</p>
```

Now we need to open ModuleController.php inside controller/default/test/ and put the following code:

```
<?php
class ModuleController extends Controller{
	public function __construct(){}
	public function main(){
		$data['name'] = $_GET['name'];
		$this->load->view("main.php",$data);
```

```
        }
?>
```

As you can see, all we need to do is pass an array with the variables needed in the view.

But we could have achieved the same result by coding:

```
$this->load->view("main.php");
$this->template->name = $_GET['name'];
```

Complex Dynamic Views

Let's say we want to build a more complex view. This is the HTML:

```
<h1><?php echo $title ?></h1>
<h2>Item listing</h2>
<ul>
<?php foreach($item as $value): ?>
        <li><?php echo $value ?></li>
<?php endforeach; ?>
</ul>
```

Now the PHP inside the controller:

```
<?php
class ModuleController extends Controller{
        public function __construct(){}
        public function main(){
                $title = "Foo bar";
                $items = array('apples','bananas','oranges');
                $this->load->view("main.html",get_defined_vars());
        }
?>
```

Note that in this case we used the function get_defined_vars() instead of the $data array. This is a more natural approach because this way you have access to all variables defined in the action function scope inside the loaded view. get_defined_vars() return all defined vars in the current context as an associative array and that's why it is perfect.

Including Views Inside Views

In some cases, we have HTML or code fragments or snippets that we use without change inside many views. The best thing to do in those cases is separate the repeating code, put it in separate files and then call these files when needed. We can call these snippets right from inside each view.

Let's say for example you have the following view main.php in the "test" module:

```
<h1>Page title</h1>
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">Logout</a></li>
</ul>
<p>Content</p>
```

And you notice that the menu item (the <ul> tag) repeats in several pages. So you can separate that HTML code and put it in content/templates/default/html/navigation/menu.php:

```
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">Logout</a></li>
</ul>
```

Finally call this snippet from inside your main view main.html:

```
<h1>Page title</h1>
<?php $this->load->view( "../navigation/menu.php")->render(); ?>
<p>Content</p>
```

Note that the first argument of loadSnippet() receives the path of the view relative to content/templates/[active template]/

Snippet Scope

The snippet we just included may me just plain HTML or may be a dynamic view. By default, the snippet will have a different scope from the parent view who loaded it, that means it won't have access to the variables of the parent view. In fact, by default, the view will have access only to the global and superglobal vars.

Note that you can load a snippet inside another snippet. Be careful not to load recursevely a snippet or you will get an infinite loop and an overflow error.

Passing the Parent View Scope

If we want the snippet to have access to the parent's view variables, then we need to pass the context to the snippet:

```
<?php $this->load->view( "../navigation/menu.php",$this->template->getContext())-
>render(); ?>
```

Note that the first parameter "../navigation/menu.php" needs to start with a double colon because by default, the function assumes that you want to load a view that is inside the current's view html directory, that's why we need to get up one level.

Remember that $this->template is the referer of the current view (the parent view), and getContext() only return the defined variables in that view. You can pass any array of variables you want as the context.

Defining A Separate Snippet Controller

If the snippet needs a scope other than the parent's view scope, then we can pass a context that's defined in a separate PHP file. In the previous example, suppose that menu items vary from user to user, then we would need a dynamic view that queries the user level and based on that it would render the appropiate menu items.

menu.php:

```
<ul>
<? if($level == “member”): ?>
	<li><a href=”#”>My account</a></li>
	<li><a href=”#”>Reports</a></li>
<?php elseif($level == “admin”): ?>
	<li><a href=”#”>Sales</a></li>
	<li><a href=”#”>Users</a></li>
<?php endif; ?>
</ul>
```

And in the controller:

```
public function main(){
	$level = get_user_level(); // This code is for the snippet
	// […] All other code for the parent view
}
```

We could easily define the $level variable in the parent's view controller function, but we would need to do this every time the snipped is required, so why not put this piece of php code in a separate file and reuse it. So create the file controller/default/navigation/menu.php:

```
<?php
$data[‘level’] = get_user_level();
```

The above snippet controller must only populate a $data array, whose elements will be passes through to the snippet view, so $data[‘level’] will be accessible in the snippet like this: $level.

Now call the snippet along with it's controller directly from main.php view:

```
<h1>Page title</h1>
<?php $this->load->view( “../navigation/menu.php”,”navigation/menu.php”)->render(); ?>
<p>Content</p>
```

So the second parameter in $this->load->view() can either be an array of variables or a path to the snippet's controller RELATIVE to controller/default/

Other View Formats

Many times we need to send the browser data in other format different from HTML. For example when we make an AJAX call and expect a JSON object as a response from the server.

Sending JSON response

If we want to send a JSON response we don't need to load any HTML view, it's very straight forward as all we need to do is echo the json formatted array so we may be tempted to do the following:

```
<?php
class ModuleController extends Controller{
	public function __construct(){}
	public function main(){
		$response = array(‘foo’ => ‘bar’);
		echo json_encode($response);
```

```
        }
?>
```

And it is OK as long as your layout.html is a blank template! If that's not the case, FALCODE will echo your JSON object INSIDE the page content of the default Layout, and that's not what we are expecting obviously. That's why we need to tell FALCODE we are going to send a JSON response and we can do so in two ways:

First: Using the blank.html layout, which is simply a layout without header, footer or any other element. We can do so by calling

```
$this->blank();
```

which is exactly the same as calling

```
$this->load->layout("blank.html");
```

Second: The right way would be to tell FALCODE exactly what our response is going to be:

```
$this->response->isJSON();
```

This way we are not only telling FALCODE to use a blank layout but also to send the propper http headers in the response (Content-type).

Views Using Template Engine

As we have seen until now, view are not actually plain HTML views but HTML files with embeded PHP. By using a template engine, your views are not actually written in PHP, they use a template engine syntax for handling the dynamic content passed from your controller, and therefore all your views are plain HTML.

This approach has many advantages over using PHP:

- Your code gets a lot more compact
- It prevents designers from screwing with your PHP code
- It's all plain old HTML
- Prevents you from putting application logic inside a view
- Your code is more readable
- Every view is compiled back to PHP and cached so there is no performance problem
- The syntax of the template engine is easy to learn and is similar to javascript object manipulation

All views and snippets can me written in PHP or TemplateEngine syntax. In fact, you can have a TemplateEngine view load a PHP snippet and vice-versa.

Template Engine Syntax

The syntax is very simple, every variable or structure is surrounded by curly braces. Variables must start with $, every other thing will be treated as a structure.

```
<html>
<head>
<title>{$page_title}</title>
</head>
<body>
<h1>{$page_title}</h1>
<ul>
{loop:$rows}
<li>{$row.title}</li>
{/loop}
</ul>
```

Loading a Template Engine View

The procedure is exactly the same as loading an ordinary view just that the extension MUST BE .html or .tpl.

This means that if you execute:

```
$this->load->view("view.php");
```

The system will assume your view is a PHP view but if:

```
$this->load->view("view.html");
```

The system will assume your view is a TemplateEngine view and therefore it will parse it with the template engine class.

Variables

This is a comparisson of a PHP variable in PHP vs TemplateEngine

Variable	PHP	Template Engine
$title	<?php echo $title; ?>	{$title}
$items[0]	<?php echo $items[0]; ?>	{$items.0}
$blogs['first']['date']	<?php echo $blogs ['first']['date']; ?>	{$blogs.first.date}
$Row->title	<?php echo $Row->title; ?>	{$Row->title}

NOTE: By default all variables in the template engine will encode the html tags found in the strings. That is to prevent user inputs from alter the HTML layout. If you want to disable this feature in all the site, change TemplateEngine::ESCAPE_TAGS_IN_VARS to false.
If you only want to disable this for a single var, add the modifier preventTagEncode()

Structures

The template engine supports two basic control structures: if/elseif/else and foreach. And several helper structures.

The variables inside the structures must not be surrounded by curly braces.

Foreach structure

Foreach structure if perfect for looping or iterating arrays:

PHP	TemplateEngine
 <?php foreach($blogs as $post): ?> <?php echo $post; ?> <?php endforeach; ?> 	 {loop:$blogs,item=post} {$post} {/loop}
 <?php foreach($blogs as $i => $post): ?> <?php echo $post['date']; ?> <?php endforeach; ?> 	 {loop:$blogs,key=i,item=post} {$post.date} {/loop}

Note that when assigning the loop the “key” and “item” vars, they DO NOT HAVE the dollar symbol prefixed but when used inside the structure they have it.

You can code nested loops.

If Structure

For conditionals

PHP	TemplateEngine
<?php if($number < 100): ?> Less than 100 <?php elseif($number >= 100 && $number < 200): ?> Between 100 and 200 <?php else: ?> Greater than 200 <?php endif; ?>	{if:$number < 100} Less than 100 {elseif:$number >= 100 && $number < 200} Between 100 and 200 {else} Greater than 200 {/if}

URL Structure

It is a helper function to generate absolute URLs from whithin your view, it uses the global url() function. Suppose you want to do the following:

```
<a href=”http://yourdomain.com/blog.view/10/from,home”>Click here</a>
```

With this modifier you can do it like this:

```
<a href=”{url:blog/view?id=10&from=home}”>Click here</a>
```

Everything that goes after url: accepts the exact same syntax as the url() function.

Include Snippet Structure

This is the equivalent on calling $this->load->view() directly from the view to load a snippet.

It’s basic syntax is:

```
{include:navigation/menu.html}
```

Note that the first parameter after “include:” is the file to include relative to the template’s html directory.

If you need to include a separate controller:

```
{include:navigation/menu.html,controller=navigation/menu.php}
```

As both the snippet and it’s controller are located in the same navigation folder and have the same name (excluding their extension), we could do the following:

```
{include:navigation/menu.html,find_controller}
```

And then the TemplateEngine will assume that it's snippet is located in: controller/default/navigation/menu.php.

Note that you can indeed include a PHP snippet from a TemplateEngine view or snippet as such:

```
{include:navigation/menu.php,find_controller}
```

Modifiers

The modifiers are functions to transform the ouput of a variable and that are very useful in views.

Take a look at the following example

PHP	Template Engine
<?php echo (empty($post['title']) ? "NA" : (strlen($post['title']) > 20 ? strtolower(substr($post['title'],0,20)) : strtolower($post['title']))); ?>	{$post.title.lower().truncate(20).default("NA")}

Both of them give us the same result… Which one do you like best?

Note that modifiers can be chained and the result of the first modifier will be passed to the second one and so on. All modifiers must contain open and close parenthesis even if they do not accept parameters.

In case the modifiers have one or more parameters, if the parameters are not numbers, they must be surrounded by double quotes ("). NOTE: DO NOT use single quotes ('), in the template engine they are not the same.

upper()

Transforms the string to upper case. Equivalent in PHP to strtoupper()

lower()

Transforms the string to lower case. Equivalent in PHP to strtolower()

capitalize()

Capitalices the first letter of each world. Equivalent in PHP to ucwords()

abs()

Returns the absolute value of the number. Equivalent in PHP to abs()

print_r()

Equivalent in PHP to print_r()

truncate($length)

Truncates the string to if it's length is larger than the first parameter. If it's longer, will truncate and suffix "…"

Parameter	Type	Description
$length	numeric	Sets the maximum length of the string

length()

Returns the length of a string of the number of items in an array.

count()

Alias of length()

toLocal()

Converts the datetime to local timezone. It uses the date_to_local() global function.

toLocalTime()

Alias of toLocal()

toGMT()

Converts the datetime to GMT time. Uses the date_to_gmt() global function.

date($format)

Converts the date to the given format. Assumes that the variable's format is Y-m-d H:i:s

Parameter	Type	Description
$format	string	All of the formats that are accepted in PHP date() function are valid

nl2br()

Equivalent to PHP's nl2br(). Converts white spaces to

stripSlashes()

Equivalent to stripSlashes() in PHP.

sum($value)

Adds $value to the variable

Parameter	Type	Description
$value	numeric	Value to add

substract($value)

Substract the given value to the variable

Parameter	Type	Description
$value	numeric	Value to substract

multiply($value)

Multiplies $value by the variable

Parameter	Type	Description
$value	numeric	Value tu multiplicate

divide($value)

Divide the variable by $value

Parameter	Type	Description
$value	numeric	Value to divide by

addSlashes()

Equivalent in PHP to addSlashes()

encodeTags()

Encode HTML tags from the string. Equivalent in PHP to htmlspecialchars($var,ENT_NOQUOTES)

decodeTags()

Opposite of encodeTags(). Equivalent in PHP to htmlspecialchars_decode()

stripTags()

Equivalent in PHP to strip_tags()

urlDecode()

Equivalent in PHP to urldecode()

urlEncode()

Equivalent in PHP to urlencode()

urlFriendly()

Returns the string as a readable url encoded string

trim()

Equivalent in PHP to trim()

sha1()

Equivalent in PHP to sha1()

numberFormat($decimals)

Returns the number with thousands separated by commas and rounded to $decimals positions

Parameter	Type	Description
$decimals	numeric	Set the decimal positions.

lastIndex()

Return the array's last key or index

lastValue()

Return thr array's last item

jsonEncode()

Equivalent in PHP to json_encode()

substr($start*[,$length]*)

Equivalent in PHP to substr()

Parameter	Type	Description
$start	numeric	Start searching in position
$length	numeric	Stop in position. Defaults to string length

join($glue)

Equivalent in PHP to implode() or join()

Parameter	Type	Description
$glue	string	Join the array with this string

split()

Equivalent in PHP to explode()

preventTagEncode()

Prevents the default tag encoding performet to all the variables.

randString($length)

Returns a random string of length $length. This modifier alters the content of the variable.

Parameter	Type	Description
$length	numeric	Sets the length of the string

replace($search,$replace)

Replaces de $search for the $replace stringas

Parameter	Type	Description

$search	string	String to search
$replace	string	String to repleace with

default($string)

Set the default string to show in case the variable has no value or is empty

Parameter	Type	Description
$string	string	Default value

ifEmpty($value*[,$else_value]*)

If the variable's value is empty, then the variable is asigned the value $value. If $else_value is set and the value of the variable is not empty, it will be assigned $else_value.

Parameter	Type	Description
$value	mixed	The value to assign the variable if it's empty
$else_value	mixed	The value to assign if the variable is not empty

if($compare_with,$value_true*[,$value_false[,$compare_logic="equals"]]*)

Makes an if comparation with the value of the variable. The variable's value is compared with the $compare_with variable. If the comparisson returns true, then the variable is assigned the $value_true. If $value_false is set and the comparisson returns false, then the variable is assigned $value_false.

Parameter	Type	Description
$compare_with	mixed	Value to compare the variable with
$value_true	mixed	Value to assign the variable if the comprisson is true
$value_false	mixed	Value to assign the variable if the comparisson is false
$compare_logic	string	The comparisson logic. It defaults to "equals" but can be: • equals • smaller • smaller or equal • greater • greater or equal • == • < • > • <= • >=

Example:

```
{$number.if(10,”Greater or equal 10”,”Smaller than 10”,”>=”)}
```

Would be the same as code:

```
if($number <= 10){ echo “Greater or equal 10” } else { echo “Smaller than 10” }
```

Modifiers in Structures

All modifiers can be applied in structures:

```
{if:$blog.name.length() < 100}Some HTML{/if}
```

Template Engine Global Variables

Every view and snippet loaded with the template engine has access to a global variable named $system that has the following structure.

- system
  - now: current date in Y-m-d H:i:s
  - get: $_GET
  - post: $_POST
  - request: $_REQUEST
  - session: $_SESSION
  - session_id: session_id()
  - server: $_SERVER
  - const: Array containing all defined constants
  - user: Current user’s login array
  - user_role: Current user role
  - pathway: URL pathways
  - module: Current module
  - control: Current control
  - action: Same as control, current action
  - path
    - http: http site path
  - is_logged: Returns true if the user is logged
  - files
    - css: CSS autoloads
    - js: JS autoloads
  - dates
    - months: Array containing 0-12
    - years: Array containing last 100 years

        - future_years: Array containing future 25 years
        - days: Array from 0-31
    - lang: Current language dictionary array

This means that every variable with name “system” passed to the view will be overridden by the $system super global variable.

Models

What is a model?

A model is the abstraction of a database table into an object. All model classes reside in engine/model and all of them are autoloaded on-demand at the time of creating the object.

Creating a model is very simple you just need to create a class inside the models folder with the following characteristics.

```
<?php
class Table extends ActiveRecord{
	protected static $table_name = “table”; // Mandatory, the name of the table
	protected static $primary_key = “id_table”; // Mandatory, primary key field name
	protected static $prefix = “t”; // Optional, table name alias
}
```

Note that the $table_name variable needs to be exactly the same as the table name the model is refering, and the primary_key needs to be the column name of the primary key field. It’s important that the class extends always the ActiveRecord system class.

Conventions

For the autoloading of modules to work, the class file name needs to be exactly the same as the class name. For example if you are about to create a model for the table “user” inside your database you would first need to create the file engine/model/User.php and your class code would be as follows (assuming the primary key for the table is the column “id”).

```
<?php
class User extends Model{
	protected static $table_name = “user”;
	protected static $primary_key = “id”;
	protected static $prefix = “u”;
}
```

Advanced Model Relationships

In order to emulate the relational nature of the databases we need to be able to code their relationships in our models. Note that coding these relationships if completely optional.

In FALCODE we can do so by defining the relationships of each model. Take for instance the following database structure.

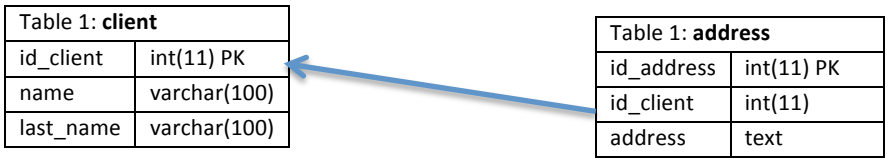


Table 1: **client**	
id_client	int(11) PK
name	varchar(100)
last_name	varchar(100)

Table 1: **address**	
id_address	int(11) PK
id_client	int(11)
address	text

*The line represents a “1 to n” relationship, being the arrow pointer the “1” side.

We know that a single client can have one or more addresses, right? So we can create the following 2 models for this db architecture.

First the client table:

```
class Client extends ActiveRecord{
    protected static $table_name="camiseta";
    protected static $primary_key="id_camiseta";
    protected static $prefix="c";
    protected static $fk = array(
        'Address' => array(  // This needs to be the same as the referenced model class
name
            'table' => 'address', // The referenced table
            'local_key' => 'id_client', // The key column name in the client table
            'foreign_key' => 'id_client',  // The key column name in the address table
            'rel_type' => Db::REL_1_TO_N
        )
    );
}
```

Second the address model:

```
class Address extends ActiveRecord{
    protected static $table_name="address";
    protected static $primary_key="id_address";
    protected static $prefix="a";
    protected static $fk = array(
        'Client' => array(  // This needs to be the same as the referenced model class
name
            'table' => 'client', // The referenced table
            'local_key' => 'id_client', // The key column name in the client table
            'foreign_key' => 'id_client',  // The key column name in the address table
            'rel_type' => Db::REL_N_TO_1
        )
    );
}
```

Database Conventions

There are several database design conventions that we encourage to follow in order to get the best of your application.

Database normalization

We highly recommend to have at least a Boyce-Codd normal form.

Keys

We recommend that every single table have at least a one-column primary key. Even if the table posseses a composite-key we encourage to…

Miscelaneous Functions

FALCODE comes with a series of built-in functions available at any time in the system. This are the same as helper functions with the exception that this ones don't need to be loaded manually as helper functions are.

json_encode($array)

This function is created just in the case the original php built-in function is not available. Same arguments as the original one, same result.

now_gmt()

Return the current GMT date time. It automatically calculates the offset based on the server timezone configuration.

Return the value un the format Y-m-d H:i:s

time_gmt()

Return the timestamp of the current GMT datetime

date_to_gmt($date,$format="Y-m-d H:i:s")

Transforms the given date from the user local timezone to GMT timezone and returns it in the given format. The format accepts exactly the same ones as the php date() function. And the $date variable must be a timestamp or a standard time format(Y-m-d H:i:s)

This function supposes the original date given is in the same timezone as the server configuration.

date_to_local($date,$format="Y-m-d H:i:s")

This is the opposite of date_to_gmt() as it transforms the GMT datetime to the user local datetime.

url($uri,$subdomain=NULL,$port=80)

This is used to generate the absolute URLs with the FALCODE URI syntax. All anchors and redirects in the HTML y PHP must by generated by this function.

The $uri parameter has the following syntax:

module/control?var1=val1&var2=val2&_path=path1&_path=path2

The module is the only required part in the uri.

The control is optional, if set it must be separated from the module with a slash

Get vars are optional and must be written in the http GET standard syntax.

In case you want to put URL paths, you need to pass them each one as a _path GET variable.

For example let's say you want to generate the following url:

http://yourdomain.com/articles.list/news/order,latest/

Then the function would be as:

```
url("articles/list?_path=news&order=latest");
```

redirect($url,$populate_get=false,$populate_post=false,$post=NULL)

This function redirects the user's browser to the determined url. In most cases the $url variable is generated with the url() function.

For example let's say you want to redirect the user to another page temporaly you can do so by

```
public function test(){
	redirect(url("login")); // Redirect all users to the login page
}
```

You can also forward the current GET variables by setting $populate_get to true.

If you also want to forward the current POST variables you can do so by setting the third parameter to true. If so, you can manually determine what post is going to be forwarded through the $post parameter. If not set, it will default to $_POST.

get_include($file)

This function includes the given file and returns it's output as a string.

get($method=”get”,$except=array(),$return=”httpquery”,$no_empty=true)

This function returns the current http input vars.

$method: Which request method to user, can be “get” or “post”.

$except: An array containing variable names to exclude from the returning array

$return: How the variables are going to be returned. Can be “httpquery” or “hidden”. If set to hidden it will return HTML hidden inputs.

is_empty($variable)

This is a more accurate empty() function variation

Extensions

In FALCODE, extensions are external libraries or plugins. All extensions reside in engine/lib/extensions.

Architecture

Single-file Extensions

If the extension is a single-file php class then it can be loadaded automatically on-demand just in the case the file name equals the class name.

If the class name does not coincide with the file name of there are more than one class in the file, or if the extension is not a class, but it still is a single-file, then we can load it:

```
$this->load->extension("extension.php");
```

We can make this call from every module controller action function. And it assumes that the extension.php is in engine/lib/extensions/extension.php.

Multiple-file Extensions

If your extension has more than one file then you need to create a new directory inside the extensions directory with the name of the library: engine/lib/extensions/multiple_file_extension.

Then you need to drop all the library files within this new directory you just created. Now we need to the FALCODE which files to load for the library to work properly and in what order. Let's say our library "FooBar" has the following structure inside engine/lib/extensions/FooBar/:

- main_file.php
- config.php
- bin
    - init.php
    - helper.php
- lang
    - es.php
    - en.php

And for the plugin to work properly the main_file.php needs to be loadad first, then inside the plugin directory inside extensions, an _autoload.php file needs to be created.

When loading the extension, FALCODE will search and execute that _autoload.php file, so then we can put in there what we need to load, so _autoload.php will have the following code (this is the code needed for the previous example):

```
<?php
require_once(“main_file.php”);
```

That’s it! Now if we would like to use the library, we just do the following:

```
<?php
class Test extends Controller(){
	public function __construct(){}
	public function main(){

	public function __construct(){}
	public function main(){
		//$FooBar = new FooBar(); // This will raise error, as extension has not
been called yet
		$this->load->extension(“FooBar”);
		$FooBar = new FooBar(); // This is OK
	}
}
```

Helpers

A helper is a repository of functions of a certain topic that are not necessarily required in all the modules or actions of the application. FALCODE comes with a series of built-in helpers such as date, string and upload, which are functions that make certain tasks a little bit easier.

All helpers are single files and are located in engine/lib/helpers.

Helpers are not classes, they are plain functions. If you were to make a helper class you should probably create an extension instead.

Loading a Helper

To load a helper you need to use:

```
$this->load->helper("helper");
```

Note that you don't need to include the file extension as it will assume it's .php. In the previous example the system will try and include the file engine/lib/helpers/helper.php.

Auto-loading Helpers

If you want a helper to load automatically, then you can prefix you helper name with an undersore (_) as such: _auto_loading_helper.php.

Executing a Helper Function

Once you load the helper, all of it's functions are automatically available and you can use them as any other normal functions.

Naming Conventions

As helper functions are "global", it's important to be carefull your function name won't conflict with a system function or other helper function. We advise you to use a prefix for all your helper functions in order to prevent this.

URL Mapping

Sometimes you need some URL to be “masked” to another URL for some particular reasons, most of the times because of making a more SEO friendly approach.

Let’s say you have an “articles”module that has a “view” action, and to access a particular article archive you would typically access by giving it’s id through a GET var. Articles are divided in “news” and “stories”. To obtain all the “news” articles ordered by publication date you would access this url:

http://yourdomain.com/articles.list/category,news/order,latest/

It’s a long URL right? Wouldn’t it be better ir you could just access via:

http://yourdomain.com/latest-news/

To accomplish that, you would normally need to create a “latest-news” module and write some more code in that module. But it would be easier if the last URL is an “alias” or the long URL, and we can do so by defining a custom URL map.

Creating a new URL map

URL maps are defined in engine/conf/map.php. And to define the map we use the static Router class which is described in detail in the Router class chapter.

To map the previous example, the map.php file would need to have:

```
<?php
Router::map(“latest-news”,”articles.list/category,news/order,latest”);
```

Now you would be able to access both ways the same result.

Advanced URL Mapping

Let’s say you want to map some URL but also get some value from the mapper url and pass it to the mapped URL. Lets say we want to view a news article:

http://yourdomain.com/latest-news/new-breaking-news-article/

And normally you would access via:

http://yourdomain.com/articles.view/category,news/title,new-breaking-news-article/

You would map both as:

```
<?php
Router::map(“latest-news/:alpha”,”articles.view/category,news/title,$1”,true);
Router::map(“latest-news”,”articles.list/category,news/order,latest”);
```

First you will notice that in the first declaration we used a “:alpha” wildcard which simply matches every alphanumeric character in that position and passes it to the mapped URL, you can access that match through the “$1” variable.

You can use any regular expression to match certain string format. If you need to match more than one value, you can access them in the second parameter using $1,$2,…,$n.

For a more detailed information about Router::map() function, checkout the complete Router class reference.

Configuration Files

Configuration files are stored in engine/conf/. The default configuration file is config.php. This one is loaded automatically. The config files structure is:

```
<?php
$config['variable_name'] = "value";
```

This file should only declare variables in the $config array.

Creating Custom Configuration Files

If you don't want to alter the main configuration file, you can go ahead and create other files. These files must have the same structure as config.php. Beware of not re-write a variable name previously defined in config.php, this could cause the system to not work properly.

The configuration vars MUST NOT HAVE SPACES or symbols other than letters, numbers and underscores. Other way they won't be accesible.

Loading Custom Configuration Files

You can load the custom config file you've just created using the following function:

```
$this->load->config("custom_config");
```

Accessing Config Variables

Once the configuration files are loadad the variables can be accesed from the controller (take the previous example):

```
echo $this->config->variable_name; // This would output "value"
```

Constants

In FALCODE all system constants are defined in engine/conf/definition.php

This are the available constants at all time

Constant	**Definition**
APP_NAME	Change this to the name of your application
APP_VER	Current version of your application
HTTP	The HTTP root of your app
HTTP_CONTENT	HTTP URL to the contents directory
HTTP_CONTENT_FILES	HTTP URL to the folder of file uploads
HTTP_CONTENT_TMP	HTTP URL to tmp folder
PATH_SYSTEM	The absolute path to the application root
PATH_CACHE	Directory to the cache folder
PATH_CONTENT	Path to content directory
PATH_CONTENT_TEMPLATES	Path to templates folder
PATH_CONTENT_FILES	Path to _files folder
PATH_ENGINE	Path to engine directory
PATH_ENGINE_MODEL	Path to model directory
PATH_ENGINE_LIB	Path to lib folder
PATH_CORE	Path to core lib folder
PATH_EXTENSIONS	Path to the extensions folder
PATH_ENGINE_CONF	Path to the configuration directory
PATH_ENGINE_LANG	Path to the lang folder
PATH_CONTROLLER	Pah to the controller root

Once the system is fully loaded, some more constants are defined on the run:

Constant	Definition
PATH_CONTROLLER_MODULES	Active controller set directory
SUBDOMAIN	The current subdomain
DSP_MODULE	The current module
DSP_CONTROL	The current action or control
DSP_DIR	The current module execution directory
DSP_FILE	The file to execute

Language

FALCODE gives you the ability to create a multi-language application. To do that you first have to create dictionaries for each language.

Structure

Language dictionaries reside in engine/conf/lang. Each file there is a language dictionary. Each of the language files must be named after the ISO 639-1 convention (http://en.wikipedia.org/wiki/ISO_639-1). For example if I want to create an english and spanish dictionary I would first create the file engine/conf/lang/en.php with the following code:

```
<?php
$lang['hello'] = "hello";
$lang['name'] = "name";
$lang['welcome_msg'] = "Hello %1, today is %2.";
```

Then the es.php file:

```
<?php
$lang['hello'] = "hola";
$lang['name'] = "nombre";
$lang['welcome_msg'] = "Hola %1, hoy es %2.";
Now inside the controller action:
public function test(){
	echo $this->lang->hello . " Richard";
	echo "<br/>";
	echo $this->lang->get("welcome_msg","Richard",date("Y-m-d"));
}
```

This would output :

In english:

hello Richard

Hello Richard, today is 2012-12-15

In spanish:

hola Richard

Hola Richard, hoy es 2012-12-15

Getting a Language Variable

When the language var does not have placeholders inside it (%1,%2…), it can be accessed directly as an object property:

```
$this->lang->hello;
```

But when it has one or more placeholders inside the string, values need to be passed so they can be replaced for the placeholders, and each value needs to be passed as an argument, these will be replaced in order of apearance.

```
$this->lang->get('welcome_msg','Ricardo',date('Y-m-d'));
```

Changing the Default Language

By default, FALCODE will try to get the language preferences of the user by fetching the browser header information. If language exists for the user's preferences then that language will be loaded if it hasn't been overrided by one of this methods:

The default language can be changed manually from the controller:

```
$this->lang->load("es.php");
```

Or by passing a "lang" POST or GET var with the language code.

http://yourdomain.com/index/lang,es/

Currency and Geolocation

FALCODE has a 3rd party geolocation database at a city-level that allows it to get the current user's currency preferences and aproximate location.

All this is done automatically by the system at app startup and saved in ISO 3166 2 format.

For example if I access from any part within the US:

```
echo $_SESSION['currency']; // Will output "USD"
```

Default currency is established in engine/conf/config.php

```
$config['base_currency'];
```

Currency Conversion

Using a 3rd party webservice that provides up-to-date USD based currency converion rates, FALCODE can provide currency conversion through the currency helper explained later on.

Image Manipulation

FALCODE uses the WideImage Open Source PHP library (http://wideimage.sourceforge.net/) for image manipulation. It has a built-in module named “image” which is used to handle and display user-uploaded images.

Where Are Uploaded Images Stored?

All uploaded files are stored in content/_files. In particular, uploaded images are stored in content/_files/images.

Folders inside content/_files/images/ represent the module from where the image was uploaded. It is a way of organizing them.

All uploaded images and other files should be renamed for security reasons. They sould maintain their original name in database so the user can download the file with their original name.

Display An Image

To display an image you just need to access the image module right from the browser or from the src attribute of an image HTML tag. All parameters must be provided either as GET variables or as URL paths.

Let’s say the user uploaded his profile picture. We know that in this particular site user pictures are stored in content/_files/images/user/ (this could have been user_image, user_images, etc.) and this particular one was saved with the name 001.jpg inside that folder.

If we would like to show the user his profile pic later on we could just access the image directly but rarely you will require the user image in it’s original size. Here is where the image module comes in handy.

Get the image in it’s original size

http://yourdomain.com/image/user/001.jpg

image: This is the image module, a standard built-in module
user: This is the folder name inside content/_files/images/ where the image is stored
001.jpg: This is the actual file name

Resize And Display The Image

There are more optional parameters you can send the module to get different image output. Note that this are all optional.

http://yourdomain.com/image/user/200x150/inside/001.jpg

Proportional Resizing

If you want to resize to a certain with or height and resize proportionally the image you can do:

http://yourdomain.com/image/user/200xauto/001.jpg
This will generate an image with a width of 200px and the height will be generated by the resizing calculations. No image portion will be lost

http://yourdomain.com/image/user/autox200/001.jpg
This will generate an image with a height of 200px and the width will be generated by the resizing calculations. No image portion will be lost

Fixed Resizing With Cropping

If you need the image to have certain specific width and height you must set both with and height in the size parameter. Note that this module prevents at all costs the image stretching! The original image aspect ratio will be preserved at all times.

http://yourdomain.com/image/user/200x150/001.jpg
// This will generate a 200x150px image. In case the image ratio coincides with the new size ratio, then no image portion will be lost, but in case the aspect ratios are different, the image will be resized until the whole new size is covered and overflow image portions will be cropped out

Fixed Resizing Without Cropping

If you must show the whole image and therefore image cropping is not an option, but at the same time you must do a resize to specific dimensions, you can do the following:

http://yourdomain.com/image/user/200x150/inside/001.jpg
// Will generate a 200x150px image. If aspect ratios do not coincide, the image will be resized to the minimum size where the whole image is visible and the empty spaces will be filled with white.

The “inside” parameter determines wheather the image should be cropped or not, it can be “inside” or “outside”. It defaults to “outside”.

Security

FALCODE has a robust built-in multi-level authentication system.

Adding Roles

The user roles and permissions are defined in the Access Control List located in engine/lib/core/Acl.php.

Roles must be defined in the constructor function of Acl class.

```
public function __construct(){
	$this->addRole(new UserRole("public"));
	$this->addRole(new UserRole("member"));
	$this->addRole(new UserRole("admin"));
}
```

Child and Parent Roles

A children role is a normal role who inherits all of it's parent's permissions, but not the other way around. That means that a child can inherit some permissions and override them with permissions of it's own.

This is how you add a parent and child role:

```
public function __construct(){
	$this->addRole(new UserRole("public"));
	$this->addRole(new UserRole("parent"));
	$this->addRole(new UserRole("child"),"parent");
}
```

Adding Permissions

Permissions are added also in the constructor function of Acl class AFTER the user role definition.

```
$this->deny($role_name,$module,$action,$subdomain);
$this->allow($role_name,$module,$action,$subdomain);
```

So if we were to deny all access on all modules to all roles or creating a blacklist (all denied unless access is granted):

```
$this->deny(NULL,NULL,NULL); // Or $this->deny();
```

In contrast, in order to create a "whitelist" (all allowed unless access is revoked):

```
$this->allow();
```

Permissions are overridding as in order of declaration.

Users and Permissions Stored on Database

If you don't want to write directly on the app all the user roles and permissions because of maintainability, you can save them on database using 3 tables.

```
CREATE TABLE `usuario_rol` (
  `id_usuario_rol` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `id_padre` int(11) unsigned DEFAULT NULL,
  `nombre` varchar(255),
  `descripcion` varchar(255),
  PRIMARY KEY (`id_usuario_rol`)
)

CREATE TABLE `usuario_permiso` (
  `id_usuario_permiso` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `id_usuario_rol` int(11) unsigned DEFAULT NULL,
  `subdominio` varchar(100) DEFAULT NULL,
  `modo` enum('deny','allow') DEFAULT 'deny',
  `modulo` text,
  `accion` text,
  PRIMARY KEY (`id_usuario_permiso`)
)

CREATE TABLE `usuario` (
  `id_usuario` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `id_usuario_rol` int(11) unsigned,
  `usuario` varchar(255),
  `pass` varchar(255),
  PRIMARY KEY (`id_usuario`),
  KEY `id_usuario_rol` (`id_usuario_rol`) USING BTREE,
  KEY `codigo_act` (`codigo_act`) USING BTREE,
  CONSTRAINT `usuario_ibfk_1` FOREIGN KEY (`id_usuario_rol`) REFERENCES
`usuario_rol` (`id_usuario_rol`) ON DELETE SET NULL ON UPDATE SET NULL
)
```

On the usuario_rol we have some default roles:

- public
- admin
- super_admin

The system works with a black or white permission list which is defined in the usuario_permiso table.

Each role can be granted of denied access to certain module/action combination. Each record in the table overrides the previous one if necessary.

Let's take for example the following user_role structure:

id_usuario_rol	id_padre	nombre
1	[NULL]	public
2	[NULL]	member
3	[NULL]	super_admin

By default, the super_admin role is the ONLY ONE that has access to EVERY SINGLE MODULE AND ACTION in the application, and this is independant of the permission structure un usuario_permiso, and this can't be overridden.

In this example, as our application is super secret we need to be sure no one access the system without previous authorization. Our permission structure is this:

id_usuario_permiso	id_usuario_rol	subdominio	modo	modulo	accion
1			deny		
2			allow	login	
3			allow	home	view
4	2		allow	secure	list,view

Some points to consider about the previous structure:

- As the first record doesn't has an id_usuario_rol associated, it applies to ALL of the roles.
- As the first record doesn't has a subdomain defined, it applies to ALL subdomains, including www and no-subdomain.
- As the first record doesn't has a module or action defined, it applies to all modules and all actions of all modules
- Then, the first record can be read: Deny the access on all actions of all modules on every subdomain to all user roles.
- As the permission rules override each other from upp to buttom, and as the second one grants access to all users to all actions of the login module, a small part of the first rule got overridden automatically.
- The rule #4 grants access on list and view actions of the secure module only to the role id #2 (member).

The required code used for fetching the roles and permissions from the database is commented on the Acl.php file. To activate it just uncomment those lines.

Activating Access Control

The access control is defined in the main config.php file:

```
$config['login_required'] = true;
```

When set to true, it will enable automatically on startup.

Access Denied

When the user tries to access some module/action combination to which he does not has access granted, an AccessDenied controller exception rises and therefore the AccessDenied error controller function is executed. There you can determine the propper action to be taken.

Authentication

FALCODE has a robust built-in user authentication system.

Database Tables

The authentication system relies in the Security system tables plus the user table:

```
CREATE TABLE `usuario` (
  `id_usuario` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `id_usuario_rol` int(11) unsigned,
  `usuario` varchar(255),
  `pass` varchar(255),
  PRIMARY KEY (`id_usuario`),
  KEY `id_usuario_rol` (`id_usuario_rol`) USING BTREE,
  KEY `codigo_act` (`codigo_act`) USING BTREE,
  CONSTRAINT `usuario_ibfk_1` FOREIGN KEY (`id_usuario_rol`) REFERENCES
`usuario_rol` (`id_usuario_rol`) ON DELETE SET NULL ON UPDATE SET NULL
)
```

Note that you can add as many fields to this table as you want.

User Authentication

FALCODE comes with a built-in module for authentication called “login” which has all this code implemented.

To Authenticate we use the Auth static class:

```
try{
	Auth::attemptLogin(array(
		‘user’ => ‘testuser’,
		‘pass’ => ‘test123’
	));
	echo “User correct”
}catch(Exception $e){
	die(“Login failed with message: “.$e->getMessage());
}
```

User and password fields as well as the SQL query used to authenticate the user are defined in config.php.

```
$config['user_table'] = "usuario";
$config['user_username_field'] = "usuario";
$config['user_password_field'] = "pass";
$config['user_lvl_field'] = "rol";
$config['user_login_query'] = "SELECT u.*,ur.nombre as rol FROM usuario u INNER JOIN
```

```
usuario_rol ur USING(id_usuario_rol) WHERE usuario = '{0}' AND pass = '{1}'"; // This is
the SQL query that's going to be executed to determine if the user exists
```

Once the user is logged, all fields returned by $config['user_login_query'] are populated the $_SESSION['login'] array.

Setting Login Cookie

If you want your user to keep logged in, you need to set a cookie. If the $_POST['set_cookie'] == 1 then a cookie will be passed to the user's web browser and session will not end automatically when closing the browser.

Set Cookie Duration And Name

Cookie duration and name are both determined in the config.php file in vars:

```
$config['login_set_cookie'] = true;
$config['login_cookie_expire_days'] = 30;
```

Use Encoded Passwords

FALCODE encourages the developers to save user passwords hash-encoded so their private information is not compromised during a security breach.

In FALCODE you can easily enable hash encoded passwords by setting $config['login_encode_pass'] = true inside the config.php file.

Note that this method uses a one-way encription algorithm sha-1 so once passwords are encoded there is no way to decode them.

If you are going to use encoded passwords YOU MUST ENCODE THEM FIRST WHEN CREATING OR UPDATING THE USER PASSWORD. This is not done automatically.

Access Login User Details

User login details can be accessed directly using the User static class or the controller $this-> object.

Checking If A User Is Logged

```
User::isLogged(); // Returns true or false
$this->user->logged(); // Alias of the above function used from inside the
modulecontroller
```

Get User Login Variable

```
User::get('name'); // Returns the "name" field of the sql login query
$this->user->name; // Same result as the above function used from inside the module
controller
```

Get Current User Role Name

```
User::getRoleId(); // If the user is not logged will return “public”
$this->user->role(); // Alias from the above function from inside the module controller
```

Logout

To logout is the equivalent of destoying the current user’s session. But sometimes the session not only holds the login information so it would be unnecessary to wipe it all out if we just want to delete the current user’s login details.

```
public function logout(){
	Auth::logout();
}
```

User Configuration Variables

User configuration variables come in handy when you need fast access to a single variable than can be used to determine for example the user layout prefernces of determine if it’s the first time a user visit vertain page.

First we need a usuario_config table that will complement the security and authentication tables:

```
CREATE TABLE `usuario_config` (
  `id_usuario_config` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `id_usuario` int(11) unsigned DEFAULT NULL,
  `var` varchar(20) DEFAULT NULL,
  `val` text,
  PRIMARY KEY (`id_usuario_config`),
  UNIQUE KEY `id_usuario` (`id_usuario`,`var`)
)
```

Storing a User Configuration Variable

You can store a configuration variable directly from the User class or from the controller:

```
$this->user->config(‘menu_position’,’top’); // Only from within the module controller functions
User::configuration(‘menu_position’,’top’); //  Same as above
```

If the variable “menu_position” already exists, it’s value will be updated to “top”, if not, it will be created.

Getting a User Configuration Variable

Getting a user configuration variable is very easy:

```
echo $this->user->config(‘menu_position’); // Result: “top”
echo User::configuration(‘menu_position’); // Result: “top”
```

Input Validation

In FALCODE, the input validation takes place in the frontend but as the frontend methods can be cheated, it's very important to have a robust backend validation method.

The validation uses the Request and the RequestVar classes.

Required

This validation prevents the user for submiting empty values.

```
$this->request->post('name',true)->required()->errorMsg("Please enter your name");
// Will validate that $_POST['name'] is not empty and if so will set error message
$this->request->validate();
```

Numeric

If we need to validate if the input is a numeric value:

```
$this->request->post('number',true)->numeric("Enter a numeric value");
// Checks if $_POST['number'] is a numeric value
$this->request->validate();
```

Email Address

Use this to verify if the input is a valid email address:

```
$this->request->post('email',true)->email("Enter a valid email address");
$this->request->validate();
```

Greater Than, Smaller Than

This ones are used to tell if the numeric representation of the input is greater or smaller than the compared value

```
$this->request->post('age',true)->greaterThan(20,"You must be at least 21 to enter");
// Verifies if $_POST['age'] > 21
$this->request->validate();
$this->request->post('age',true)->lessThan(22,"You must be at the most 21 to enter");
// Verifies if $_POST['age'] < 22
$this->request->validate();
```

Min Length Max Length

Are used to establish a maximum length and minimum length of a string.

```
$this->request->post('username',true)->maxLength(10,"Your username cannot be more than 10 characters long");
$this->request->validate();
$this->request->post('password',true)->minLength(6,"You password must be at least 6 chars long");
$this->request->validate();
```

Check If Value Exists In Table

To use this one you must have a database connection configured. It checks if the given value exists in the column of the table.

```
$this->request->post('email',true)->unique('user','email',$_SESSION['email'],"The email already exists in our database");
// This one checks the following query:
// SELECT email FROM usuario WHERE email = '$_POST[email]' AND email != '$_SESSION[email]'
// If the query returns at least one row then the validation fails
```

Equals

Check if the input coincides whith another value

```
$this->request->post('pass_confirm',true)->equals($_POST['pass'],"Your passwords do not match");
// Check if $_POST['pass_confirm'] == $_POST['pass']
$this->request->validate();
```

Regular Expression

If you need to match a custom regular expression, do the following:

```
$this->request->post('name',true)->regex('/[a-z]{5}/',"Your name must be 5 words all lower case");
$this->request->validate();
```

Custom Function Validation

```
If you need to check or validate against the result of a custom function.

// This function must return a boolean
function check_username($uname){
	return strlen($uname) > 10;
}
```

```
$this->request->post('username',true)->func('check_username',"Invalid username");
$this->request->validate();
```

File Uploading

File upload is a very common task in web applications. FALCODE has a built-in integration with the open source uploader library Plupload (http://www.plupload.com/) which is a great library for handling asyncronous file uploading.

Asyncronous Uploading

Several libraries such as Plupload must have a backend script that handles temporal uploads and receive then sometimes in chunks, as well as sending feedback to the ajax handler to maintain the user updated about the uploading progress.

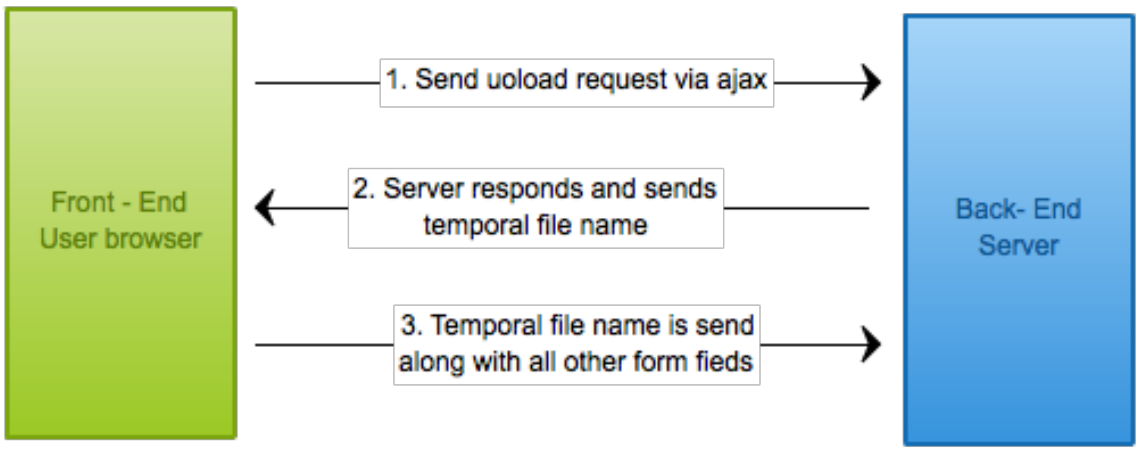
FALCODE comes with a built-in module designed for handling asyncronous uploads called “upload” with a “tmp” action, so all the asyncronous uploads from Plupload must be set to upload in “upload/tmp”.

Ell temporary files will be saved in content/tmp/. Once the file is uploaded the server will return a JSON object with the following structure:

```
{“jsonrpc”:”2.0”, “result”:null,”id”:”file_name”}
```

Where “file_name” will be replaced for the temporal file name which resides in content/tmp/ so it can be send back to the server via a POST.

This is the process:



Save The Temporal File

Once the file is saved in the temporal files folder and the form is submited, then we need to move the temporal file to a permanent folder. We can do this with the upload helper:

```
$this->load->helper("upload");
$newname = move_tmp_file("tmpfilename.pdf","upload/docs","new_file");
// This will return "new_file.pdf" which will be stored in
content/_files/upload/docs/new_file.pdf
```

HTTP Uploading

If you prefer using the old school http uploading FALCODE helps with that task with the Uploader class which will be fully explained later on. You can also perform complex uploading along with image resizing and saving with the WideImage class to upload large images, resize them and store a resized version of the image.

General Programming Conventions

Files

In FALCODE all PHP file classes are written cammel-cased. Example

Normal name	Cammel-cased
php mailer	PhpMailer
my_name_is_RICK	MyNameIsRick

All other file names are written underscore-cased. Example

Normal name	Cammel-cased
php mailer	php_mailer
my_name_is_RICK	my_name_is_rick
PhpMailer	php_mailer

File Format

All files should be utf-8 encoded to avoid the BOM (Byte Order Mark).

Classes

All classes in FALCODE are written cammel-cased as well, and the name of the class should match the name of the file. Typically, we should write one single class per file.

The class methods, functions and properties should be written cammel-cased with the first letter being lower-cased. Example

```
class Foo{
	private $firstVar;
	public function fooBar(){ }
}
```

Variables

All non-object variables should be written underscore-cased to identify them fromt he object-variables. This include function parameters.

Constants

All constants should be written in uppercase and spaces separated with undersores.

Functions

Global functions should be written underscore-cased.

PHP Closing Tag

The PHP closing tag in PURE PHP files should be avoided to prevent unwanted output that leads to “headers already sent” errors in PHP level.

One line statements

Use only one statement per line. Using multiple statements in one line can lead to readability issues.

SQL

Use UPPERCASE in the reserved words or your SQL queries and break the statements:

Incorrect

```
$this->db->fetch(“select name,last_name,id from user where id_user = 10 and name =
‘rick’”);
```

Correct

```
$this->db->fetch(“SELECT name,last_name,id
	FROM user
	WHERE id_user = 10
		AND name = ‘rick’”);
```

Separate Files For ModuleController Actions

When your action handler function in the ModuleController gets too big, it’s better to move the function content to a separate file.

For example, this is the file controller/default/test/ModuleController.php:

```
class ModuleController extends Controller{
	public function __construct(){}

	public function main(){
		include(‘main.php’);
	}
}
```

And now create controller/default/test/main.php:

```
<?php
$this->title(“Test”);
NOTE that main.php file has the same scope as the main() function. And therefore you are
able to use all the controller functions and the $this object.
```

Front-end Resources

As explained earlier in this tutorial, the front-end resources are loated in the template directory.

There are some resources such as CSS files an Javascript files that are generally loaded manually in the HTML. This is fine, but in FALCODE there is a way we can tell the system which files to load automatically and therefore include them in the current layout.

Javascript Files

Autoloading Javascript Files

Javascript files, located in the javsacript directory inside the current template can be autoloadad by adding an underscore (_) prefix to the name of the file. Every file prefixed that way will be automatically added to the layout file.

Also, if there is a javascript file which name (without extension) coincides with the current module, it will be automatically loaded.

Let's say we have this javascript files

- _jquery.js
- _api.js
- articles.js
- users.js

By default, the files _jquery.js and _api.js will be autoloaded. If the articles module is acceses via:

http://yourdomain.com/articles.list/

Or by any other call refering to the articles module, the file articles.js will be autoloaded automatically.

System Javascript Files

By default and in order for the FALCODE framework to work properly several files need to be included in certain order.

- __jquery.js (Core jQuery framework library)
- _jquery.FALCODE.js (Core FALCODE framework library)
- _misc.js (Miscelaneous functions)
- _z_general.js (Global event handlers)

Loading Javascript From the Controller

You can also tell FALCODE directly from the controller if you need certain javascript file to be loaded with the function:

```
$this->load->js("load_this.js");
```

Note that this file will be loaded after loading all the autoloaders.

CSS Files

Autoloading CSS files

CSS files work pretty much the same as the Javascript files. For a file to be autoloadad it must be prepended an underscore (_). If a css file is found whose name coincide with the current module name, it will be loaded automatically.

Loading CSS from the Controller

FALCODE can be told from the controller if certain CSS file needs to be loaded:

```
$this->load->css("load_this.css");
```

Loading Order

In all cases the files will be loaded in strict alphabetical order, so in case you need to load some file strinctly before another, you can ad a prefix to alter the loading order or your file.

The files loaded via the controller are always loaded at last. And are loaded in the order they where called.

Enabling AutoLoading

As the javascript and CSS loading is done all in the front-end and through the HTML, we cannot auto-load directly from the backend, that's why several snippets need to be called directly from the layout.

All of this snippets are located in the "nav" directory inside the template html and controller folders. Each one has to load it's own snippet controller, this code must be added in all the HTML layouts of your application inside your <head> tag:

If you are using template engine:

```
{include:nav/mod_meta.tpl,find_controller}
{include:nav/mod_css.tpl,find_controller}
{include:nav/mod_js.tpl,find_controller}
```

*Template engine syntax will be covered in the TemplateEngine chapter

If you are NOT using template engine:

```
<?php echo $this->load->view("nav/mod_meta.tpl","nav/mod_meta.php"); ?>
<?php echo $this->load->view("nav/mod_css.tpl","nav/mod_css.php"); ?>
<?php echo $this->load->view("nav/mod_js.tpl","nav/mod_js.php"); ?>
```

Javascript Conventions

Functions

All functions in javascript are groupped using objects. This is done to prevent naming conflicts amongst all javascript libraries. It is used as a namespacing in javascript. This way, instead of coding:

```
function foobar(){
	return false;
}
```

We would do:

```
Foo = {};
Foo.bar = function(){
	return false;
}
```

Embeded Scripts vs JS Files

In order to fully separate application views from application logic, all javascript needs to be written in separate .js files within the js directory. Embed javascript is only allowed when certain variable value needs to be obtained from the view. For example:

```
<script type="text/javascript">
Mod.Articles.id = "<?php echo $id ?>";
</script>
```

Then we can operate separately with this value inside the articles.js file:

```
Mod.Articles = {}
Mod.Articles.read = function(){
	Mod.Articles.fetch(Mod.Articles.id);
}
```

Event Handlers

For the same reason as the mentioned above, all html objects event handlers must be written separately in the .js file. Built-in HTML attribute event-handler should be avoided.

For example this click handler inside a view of the "articles" module:

```
<a href=”#” id=”first_link” onclick=”alert(1);”>Click me</a>
```

Should be replaced for

```
<a href=”#” id=”first_link”>Click me</a>
```

And inside articles.js file:

```
$(function(){
	$(“#first_link”).click(function(){
		alert(1);
	});
});
```

When dealing with hundreds of event handlers inside a view, it’s far more maintanable having all of them in a JS file instead of having then embeded in the actual view.

Global Javascript Objects

There are 5 global javascript objects in FALCODE:

- Misc
- Mod
- Hash
- Sys
- Lang

The global objects will be explained later on.

Module Objects

Each module must have it’s own object, which belongs in the Mod global object. Generally, the module objects are declared in the module javascript file (which is auto-loaded).

Let’s say we want to create the module object for the “blog” module. This module needs several functions and event handlers for it’s views so the blog.js file would be like this:

```
Mod.Blog = {} // First create the module object
// Then define all the module specific functions
Mod.Blog.loadPosts = function(){
	$(“#blog_container”).load(Sys.path.http+”?i=blog/getPosts”);
}
// Finally set all the event handlers on the doc-ready event
$(function(){
	$(“a.load-blog”).click(function(){
		Mod.Blog.loadPosts();
	});
});
```

Global Javascript Event Handler

In FALCODE the global javascript event handler is the _z_general.js file. This is where event handlers that apply in all the application are defined.

In fact, the global javascript event handler has it's own module object, and it is: Mod.G

If you were to apply an event handler to some element inside your layout, and as your layout appears in all the pages, it is likely that you use this file for that.

Handling URI Hash From Javascript

URL hashs come in very handy when you are developing a very dynamic or AJAX-based web application.

In FALCODE, URL hashs have the same syntax as http GET vars:

http://yourdomain.com/#var1=val1&var2=val2

This way we can handle several variables in a single hash.

Adding and Modifying Hash Variables

We can easily add or update a value stored in the URL hash. We can tell FALCODE to add a key/value pair, in case the key exists, it updates it's value but if not, it creates the key and updates the URL.

We'll assume that before this function gets executed, the user's URL is:

http://yourdomain.com/

```
$("a#test").click(function(){
       Hash.append("tab","details");
       return false;
});
```

NOTE: In the previous case if the anchor's http atrubute is "#", it is very important for this to work to always return false, because otherwise the anchor's http attribute will override the hash with a simple "#".

After the execution, the user's URL is:

http://yourdomain.com/#tab=details

Removing a Hash Key/Value Pair

This is very straight-forward:

```
$("a#test").click(function(){
	Hash.remove("tab");
	return false;
});
```

Hash Changed Event Handler

It wouldn't be useful to have the URL hash changing if we aren't able to capture each time the hash changes. We can add an event listener to record every time the URL hash changes.

```
$(window).bind("hashchange",function(e){
	alert("The current Tab is: " + Hash.get("tab") );
});
```

You can add several "hashchange" event listeners inside your module javascript file.

Sys Global Object

The Sys global javascript object would be the equivalent for $system global variable inside the TemplateEngine context.

This one has the following structure:

- Sys
  - session_id: current PHP session
  - path
    - http: Absolute http path
    - img: Absolute images path for the current template
    - swf: Absolute path for the current template swf
    - js: Absolute js path for the current template
  - user
    - logged: True/false if the current user is logged or not
    - pic: Current user pic if available
  - module: Current module
  - control: Current action
  - flash: Flash messages sent by the server

Getting Language Variables In Javascript

The global javascript Lang object is automatically populated with all the current language dictionary so it is available at any moment from any javascript.

Let's say we want to alert a message to the user with the 'module_not_found' string defined in the language dictionary.

Mod.G.alertError = function(){

```
        alert( Lang.module_not_found );
}
```

In the case the doctionary key had wildcards (%1..) then we can use:

```
alert( Lang.get('module_not_found','wildcard replace') );
```

PHP Class Reference

Acl

The Acl or Access Control List is where the user roles and permissions are set in the system.

All definitions should be set in it's constructor class.

ActiveRecord

All the models extend this class. This class applies the active record architectural pattern.

Instanciating a Model

ActiveRecord is an abstract class and as such it can be instanciated directly. The only way to do this is by instanciating a model class which inherits all of it's methods.
In this example we have a model that represents the "blog" database table:

id_blog	title	date	id_author
1	First blog	2012-12-10	1
2	My Second Blog	2012-12-17	2
3	The third blog	2012-12-18	2

Then we have the author table:

id_author	name	last_name
1	Rick	Gamba
2	John	Perez

We have 2 models, Blog.php and Author.php, and each one has defined the relations between each table with the id_author. We know one author can have many blogs but a blog may only be written by a single author.

Generating SQL Queries

The ActiveRecord provides several ways to emulate a natural SQL query with an object oriented approach.

Data can be fetched as simple as this:

```
$Blog = new Blog();
$Blog->execute(); // This gets all the rows from the table
```

execute($execute_query = true)

This function complies the SQL based on the selector functions such as select(), where() and join() and then if $execute_query is set to true, it executes the query on the db.
No SQL is executed or compiled until this function is called. Note that the order in which the selector functions are called does not affect the result of the SQL query

select($fields = "*")

Tells which fields to select from the query. It generates the SELECT part of the SQL Query. Of ommited, it defaults to "*". Note that calling this function is totally optional.

```
$Blog = new Blog();
$Blog->select("title,date")->execute();
// Generates: SELECT title,date FROM blog b
```

where($sql*[,$replace_1[,$replace_n]]*)

Sets the WHERE part of the SQL. The $sql parameter establishes the comparisson. Inside $sql, we can set several wildcards or placeholders. For each placeholder, a new argument needs to be added to the function.

The benefits of using wildcards is that the variables are escaped.

```
$Blog->where("id_blog = 1")->execute();
```

Or using wildcards:

```
$Blog->where("id_blog = {0}",1)->execute();
// Generates: SELECT * FROM blog b WHERE id_blog = 1
```

Or several wheres:

```
$Blog->where("id_blog = {0}",1)->where("title = '{0}'","Test")->execute();
// Generates: SELECT * FROM blog b WHERE id_blog = 1 AND title = 'Test'
```

Or where and OR combined:

```
$Blog->where("(id_blog = {0} OR id_blog = {1})",1,2)->where("title = '{0}'","Test")-
>execute();
// Generates: SELECT * FROM blog b WHERE (id_blog = 1 OR id_blog = 2) AND title = 'Test'
```

join($sql[,$inner=true])

Generates the joins of the SQL. Several joins can be chained.

```
$Blog->join("author a USING(id_author)")->select("b.*,a.name")->execute();
// Generates: SELECT b.*,a.name FROM blog INNER JOIN author a USING(id_author)
```

oderBy($fields*[,$order=true]*)

Generates the ORDER BY sql command.

```
$Blog->orderBy("date");
// Generates: SELECT * FROM blog b ORDER BY date ASC
```

```
$Blog->orderBy("date","DESC");
// Generates: SELECT * FROM blog b ORDER BY date DESC
```

Or by multiple columns:

```
$Blog->orderBy("date,title");
// Generates: SELECT * FROM blog b ORDER BY date,title ASC
```

groupBy($field)

Generates the GROUP BY sql command.

```
$Blog->groupBy("id_author")->execute();
// Generates: SELECT * FROM blog b GROUP BY id_author
```

limit($sql)

Generates the MySQL limit query command.

```
$Blog->limit("10");
// Generates: SELECT * FROM blog b LIMIT 10
```

Or with offset

```
$Blog->limit("10,10");
// Generates: SELECT * FROM blog b LIMIT 10,10
```

having($sql)

Generates the SQL HAVING command.

ActiveRecord::find($mixed)

The find static function is a shortcut to other selector functions and is useful when you want to filter by the table's primary key.

The first parameter can be "all", "last" or any number.
If it has more than one parameter, it will search the primary key that matches at least one of these parameters.

For example:

```
$Blog = new Blog();
$Blog->where("id_blog = 1 OR id_blog = 2 OR id_blog = 3")->execute();
```

Is the same as:

```
$Blog = Blog::find(1,2,3);
```

ActiveRecord::find_by_[field]()

This is a static function whose name can change for creating complex queries.
All the function name right to "find_by" is dynamic, that means it can change to whatever column name you want.

For example:

```
$Blog = Blog::find_by_id(1);
//$Blog = Blog::find_by_id_blog(1); // Both lines are the same
// Generates: SELECT * FROM blog b WHERE id_blog = 1
```

Or by multiple fields

```
$Blog = Blog::find_by_fecha_and_title('2012-12-17','Test');
// Generates: SELECT * FROM blog b WHERE fecha = '2012-12-17' AND title = 'Test'
```

Or with Ors

```
$Blog = Blog::find_by_fetcha_or_title('2012-12-17','Test');
// Generates: SELECT * FROM blog b WHERE fecha = '2012-12-17' OR title = 'Test'
```

This function creates the query and calls the execute() function.

findById($id)

This one filters by primary key and executes the query

```
$Blog = new Blog();
$Blog->findById(10); //  There is no need to ->execute()
```

Chaining Select Functions

Select functions can be chained to build complex queries.

```
$Blog = new Blog();
$Blog->select('b.title')->join("author a USING(id_author)")->where("title =
'{0}'",'Test')->orderBy('date','DESC')->limit(10)->execute();
```

Or without consecutive chaining, remember function order does not affect the result:

```
$Blog = new Blog();
$Blog->select('b.title')->orderBy('date','DESC');
if($condition)
	$Blog->join("author a USING(id_author)");
foreach($where_filters as $filter)
	$Blog->where($filter);
$Blog->execute();
```

Getting the Generated SQL

Once the the query is executed with execute() you can get the generated sql by accesing the SQL property:

```
$Blog = new Blog();
$Blog->select('b.title')->join("author a USING(id_author)")->where("title =
'{0}'",'Test')->orderBy('date','DESC')->limit(10)->execute();
echo $Blog->Sql;
// Will output: SELECT b.title FROM blog b INNER JOIN author a USING(id_author) WHERE
title = 'Test' ORDER BY date DESC LIMIT 10
```

Fetching Data

Once the SQL query has been generated and executed with execute() function, now we can start fetching the results using one of these functions.

next($advance = true,$use_foreign_relations = true)

This functio fetch the next row of the result. It returns an associative array. Each time you call it a new row is fetched

```
$Blog = new Blog();
$Blog->findById(1);
$row = $Blog->row();
```

Fetching more than one row:

```
$Blog = new Blog();
$Blog->execute(); // Find all rows
while($row = $Blog->next()){

}
```

Getting Values Through Properties

In the previous example we can get the “title” field of the row by

```
echo $row[‘title’]; // Will output “First blog”
```

But we can also access it directly from the object once the next() function has been called:

```
echo $Blog->title; // Will output “First blog”
```

recordset()

This function return the current row fetched by next() function

```
$Blog = new Blog();
$Blog->findById(1);
$row = $Blog->row();
$copy = $Blog->recordset(); // $row is the same as $copy
```

first()

Same as recordset but this one instead of returning the array return the actual object.

```
$Blog = Blog::find_by_id(10)->first();
```

resultArray()

Return all the rows returned by the query in a multidimensional associative array.

Updating Data

To update data there is no need to call the execute function after calling the save function

save()

Once you have fetched the data you can change the values by changing the object’s properties and calling the save() function.

```
$Blog = new Blog();
$Blog->execute()
$Blog->next();
```

```
echo $Blog->title; // “First blog”;
$Blog->title = “New first blog”;
$Blog->save();
// Generates: UPDATE blog SET title = ‘New first blog’ WHERE id_blog = 1
echo $Blog->title; // “New first blog”
```

Or you can create a new row:

```
$Blog = new Blog();
$Blog->name = “New blog”;
$Blog->date = “now()”;
$Blog->id_author = 1;
$Blog->save();
// Generates: INSERT INTO blog(name,date,id_autor) VALUES(‘New blog’,now(),’1’)
$Blog->id_blog; // This will have the new auto generated id
```

populate($array)

This function populates all the fields of a row generally to create a new one. For the previous example the equivalent would be:

```
$Blog = new Blog();
$Blog->populate(array(
       ‘title’ => ‘New blog’,
       ‘date’ => ‘now()’,
       ‘id_author’ => 1
))->save();
```

delete()

Deletes the current row selected

```
$Blog = new Blog();
$Blog->execute();
$Blog->delete(); // Deletes the first row obtained
```

Miscelaneous Functions

clear()

Clears the SQL and the recordset of the current object

affectedRows()

Returns the affected rows of the executed query. In case of a SELECT it will return the number of rows returned by the query.
In case of a UPDATE or INSERT it will return the number of added rows or updated rows.

Check For Errors

You can check is the last query failed with the following code:

```
$Blog = new Blog();
$Blog->populate(array(‘title’ => ‘Blog title’))->save();
if($Blog->lastQueryFailed())
       die(“Blog could not be saved”);
```

Foreign Key Relations

As we described earlier in the Models chapter we can setup the table relations inside the model using the $fk variable.
Using the previous example of the tables “blog” and “author” and assuming their relations have been coded in their models, this is what we can do with those relations. Let’s say we need to get the blog and the author’s name.

Normally we would make a query with an inner join from blog to author like this:

```
$Blog = new Blog();
$Blog->join('author a USING(id_author)')->where('id_blog = 1')->select('b.*,a.name')-
>execute();
$Blog->next();
echo $Blog->name; // outputs “Rick”
```

This is the alternate way:

```
$Blog = Blog::find_by_id(1)->first();
echo $Blog->Author->name; // outputs “Rick”
```

And vice-versa, remember that one author may have several blogs:

```
$Author = Author::find_by_id(1)->first();
echo $Author->Blog->first()->title; // outputs “First blog”
```

Or you can iterate over the author’s blogs:

```
while($Author->Blog->next())
       echo $Author->Blog->title;
```

Disabling Foreign Key Object Relations

It’s true that foreign key object relations can affect the performance as several extra queries are executed every time next() function is called.

If you dont want to use use object relations, just do the following:

```
$Blog->next(true,false);
// The second parameter tells the function not to fetch object relations
// The following line would generate an error:
// $Blog->Author->name;
```

Auth

This class handles all the user authentication. It requires the table usuario, usuario_rol and usuario_permiso in order to work correctly as established in the security and authentication chapters.

Auth::attemptLogin(array $post)

Return true if everything is ok, returns false if the user is already logged or if post is empty.
If authentication failed, an exception is raised.

Parameter	Type	Description
$post	array	user and password fields to validate

Auth::logout()

Logs the user out and deletes any active session cookies he has

Cipher

Public properties

Name	Type	Description
$cipher	constant	The name of the Mcrypt cypher to use. Defaults to MCRYPT_RIJNDAEL_256
$key	string	The key use for encoding. Defaults to the constant CRYPT_KEY
$mode	constant	The encription mode. Defaults to MCRYPT_MODE_CBC
$iv	string	Initialization vector. Defaults to CRYPT_IV

encrypt($text*[,$key[,$iv]]*)

Function used to encript the string. Returns the encrypted string.

Parameter	Type	Description
$text	string	text to encrypt
$key	string	The encryption key, defaults to the $key property
$iv	string	The iniciation vector, defaults to the $iv property

decrypt($encoded_text*[,$key[,$iv]]*)

Decodes the encoded string and returns the decoded text

Parameter	Type	Description
$encoded_text	string	text to decrypt
$key	string	The encryption key, defaults to the $key property
$iv	string	The iniciation vector, defaults

		to the $iv property

Config

Class used to get the user configuration variables. This class is a singleton as such can't be instanciated directly.

This class is available through the module controller: **$this->config**

Config::getInstance()

Returns the single instance of the class

__get()

Get magic method. Gets the variable key:

echo $this->config->text_variable;

load($file)

Loads the configuration file. The file path should be relative to engine/config/

Controller

This is an abstract class and is the class that is extended in all ModuleControllers, therefore almost all of it's methods are available from the ModuleController's scope through the $this object.

setDefaultAction($action)

Sets the default action in case no action is defined

Parameter	Type	Description
$action	string	name of the public function inside the ModuleController

title($name)

Sets the <title></title> tag inside the <head> of the layout. It is the title of the page. If $config['tpl_append_title'] == true then the application name with be appended at the end of the string.

Parameter	Type	Description
$name	string	The title of the page

blank($boolean)

Sets or unsets the _blank.html layout as the current layout and therefore the layout is completely blank.

Parameter	Type	Description
$bool	boolean	boolean to tell if the layout is blank or not

render()

This function renders the default view. This function is not necessary to execute as it will be executed automatically.

breadcrumb($array)

This function sets the page breadcrumb which will be displayed later by a snippet.

Parameter	Type	Description
$array	array	This should be an associative array

Example:

```
$this->breadcrumb(array(
       array('Home',url('home')),
       arrat('Current page',url('current_module'))
));
```

memcached()

Returns the system memcached object referece

throwAccessDenied($error_message)

This is a shortcut to:

throw new ControllerException($msg,ControllerException::ACCESS_DENIED);

Parameter	Type	Description
$error_message	string	Error message to pass to the error controller action

throwUnderMaintenance($error_message)

This is a shortcut to:

```
throw new ControllerException($msg,ControllerException::UNDER_MAINTENANCE);
```

Parameter	Type	Description
$error_message	string	Error message to pass to the error controller action

throwCustomError($error_message)

This is a shortcut to:

```
throw new ControllerException($msg,ControllerException::CUSTOM_ERROR);
```

Parameter	Type	Description

$error_message	string	Error message to pass to the error controller action

throwValidationError($error_message)

This is a shortcut to:

```
throw new ControllerException($msg,ControllerException::VALIDATION_ERROR);
```

Parameter	Type	Description
$error_fields	array	An associative array containing the error fields. Example: array(array('field' => 'username', 'error' => 'Bad user'));

Db

This is the main database connector and abstraction layer. All Modules use an upper level query generated through this class. This class is a singleton and can be accessed through the ModuleController using **$this->db**

Connect To Database

Database connection details are especified in the config.php file in the following vars:

```
$config['db_host'] = "localhost";
$config['db_user'] = "username";
$config['db_pass'] = "password";
$config['db_name'] = "dbname";
$config['db_engine'] = "MySQL"; // Driver name, should coincide with a file name inside engine/lib/core/db_drivers
```

Db::connect()

Connects to the database using the database credentials defined in config.php

Db::getInstance()

Gets an object instance of the singleton class. If the DB is not connected then first attempts connection and then returns the object.

query($sql)

Executes the SQL query and returns the query handler resource

Parameter	Type	Description
$sql	string	Plain SQL to execute

Example from inside a module controller action function:

```
$query = $this->db->query("SELECT * FROM blog");
```

getRow($resource)

Fetch the next row of the SQL resource and returns an associative array containing key/value pairs of the row.

Parameter	Type	Description
$resource	object	SQL resource returned by the query() function

Example from inside a module controller action function:

```
$query = $this->db->query("SELECT * FROM blog");
$row = $this->db->getRow($query);
```

getRows($resource)

Gets all the rows of the SQL query.

Parameter	Type	Description
$resource	object	SQL resource returned by the query() function

Example from inside a module controller action function:

```
$query = $this->db->query("SELECT * FROM blog");
$rows = $this->db->getRows($query);
```

selectDb($db_name)

Selects the current database

Parameter	Type	Description
$db_name	string	Name of the database

lastId()

Returns the last insert id. For example in MySQL is the equivalent to mysql_insert_id()

startTransaction()

Starts a SQL transaction

commitTransaction()

Commits a SQL transaction

rollbackTransaction()

Rollbacks a SQL transaction

escape($string)

Escapes the string user the driver function and return the escaped string. In MySQL is the equivalent to mysql_real_escape_string()

Parameter	Type	Description
$strin	string	String to be escaped

affectedRows()

Returns the number of affected rows of the query. Equivalent in MySQL to mysql_affected_rows()

numRows()

Gets the number of rows of the current SELECT query. MySQL equivalent to mysql_num_rows()

listTables()

Gets an array of all the database tables

getError()

Returns the last error message in case the last query failed.

fieldExists($table,$field)

Returns true if the field exists in the given table

Parameter	Type	Description
$table	string	Table name
$field	string	The column or field name

maxVal($table,$field*[,$extra_sql]*)

Returns the greatest numeric value in the column in the given table.

Parameter	Type	Description
$table	string	Table name
$field	string	Column name of the table
$extra_sql	string	Set extra SQL in case you need to set some extra conditions in your query. Example “AND category = ‘test’ ” DONT start with a WHERE statement, instead use AND

fetch($sql)

Executes and gets the result array of the given SQL query.
Returns an object with 3 properties:

- num_rows: Has the number of rows returned by the query
- row: Associative array of the first row of the query
- rows: 2 dimensional associative containing all the rows returned by the query

Parameter	Type	Description
$sql	string	Plain SQL query to be executed

Example:

```
$result = $this->db->fetch("SELECT * FROM blog");
echo $result->num_rows; // Outputs 2
echo $result->row['title']; // Outputs "First blog";
// Or get all the rows of the query
foreach($result->rows as $r){
	echo $r['title'];
}
```

getVal($table,$field,$reference,$get_field*[,$extra_sql[,$join[,$bt]]]*)

Gets a specific row table field value. In case no field exists it will return false.

Parameter	Type	Description
$table	string	Table name
$field	string	Column name
$reference	string	Will find the column with this value as a reference. Should be a table key or an index
$get_field	string	The name of the column whose value will be returned
$extra_sql	string	extra SQL to narrow down the query. Do not start with a WHERE command, instead start with AND
$join	string	If the query needs a join, put it here. Example: INNER JOIN table t USING(id)
$bt	boolean	Wheather or not to surround the $reference with backticks. Defaults to true. Set to false if $reference is a MySQL function or variable

Example

```
$blog_title = $this->db->getVal('blog','id_blog',1,'title'); // Outputs "First blog"
// Equivalent SQL: SELECT title FROM blog WHERE id_blog = 1
```

insert($table,$fields*[,$extra_sql]*)

Executes a SQL insert command. This approach has many advantages over executing the insert directly using $this->query(). It escapes automatically every field value, converts all the "now()" functions to GMT timezone and is more readable and easy to use.

Parameter	Type	Description
$table	string	Table name to insert in
$fields	array	Associative array containing the field/value information to insert
$extra_sql	string	Extra sql to add

Example

```
$this->db->insert('blog',array(
        'title' => 'New blog',
        'id_author' => 1,
        'date' => 'NOW()'
));
// Would generate something like:
// INSERT INTO blog( title,id_author,date ) VALUES( 'New blog',1,NOW() )
// But with escaped values and other security issues resolved
```

update($table,$fields,$extra_sql)

Executes an SQL update command, similar syntax and behaviour of the insert() function.

Parameter	Type	Description
$table	string	Table name
$fields	array	Associative array containing field/value pairs or data to insert
$extra_sql	string	Specified the row that will be updated

Example:

```
$this->db->update('blog',array(
       'title' => 'First blog updated',
       'id_author' => 2
),'WHERE id_blog = 1');
// Would generate:
// UPDATE blog SET title = 'First blog updated', id_author = 2 WHERE id_blog = 1
```

updateTimezone()

Updates the SQL server timezone to match the internet server timezone

now()

Returns the GTM date in format Y-m-d H:i:s

Database Drivers

The Db class is only a standard database function repository that works directly with the given database driver specified in the config.php file.

Database drivers must have a series of functions. For more information on how to build a database driver please check out the MySQL.php driver class inside engine/lib/core/db_drivers.

Lang

Lang class gives the ability to access the current language dictionary items.
This class can be accessed directly from the ModuleController using: **$this->lang**

Lang::get($item*[,$placeholder_replace[,$placeholder_replace[,…]]]*)

Gets the language dictionary item and replace the placeholders if necessary.

Parameter	Type	Description
$item	string	Array index of the lang dictionary
$placeholders	string	Strings to replace the placeholders, can be several and each one will replaced by the corresponding position placeholder. Example: the first one will be replaced for the placeholder "%1" and so on

Lang::getDictionary()

Gets the whole dictionary items as an associative array.

__get($item)

This is a magic function alias for Lang::get(). Can be used from the ModuleController:

```
echo $this->lang->test_item;
```

item()

Alias for Lang::get() when accessing item with placeholders. Can use as:

```
echo $this->lang->item('test_with_placeholders','First','Second');
```

Loader

This class is used to load views,templates,layouts,helpers,extensions,js and css.
It is available at any time from the ModuleController using: **$this->load**
It is a singleton class and as such can't be instanciated.

It can be called from the module controller or directly. Example:

```
$this->load->extension('test');
// Is the same as :
Loader::getInstance()->extension('test');
```

Loader::getInstance()

Gets an instance of the loader singleton object.

helper($file)

Loads the given helper.

Parameter	Type	Description
$file	string	The file name relative to engine/lib/helpers/ File extension is not neccesary if it is .php

Example:

$this->load->helper("array");

view($file*[,$variables[,$view_alias]]*)

Loads a view as explained in the View chapter.

Parameter	Type	Description
$file	string	The file name relative to content/templates/[active]/html/[module]/ File extension is not neccesary if it is $config['tpl_default_extension']
$vars	array OR string	If it's an array, all array elements will be passed as variables to the view. If it's a string the file will be included relative to controller/default/. The file sould populate the $data array whose elements will be passed as variables to the view
$view_alias	string	The loaded views will remain available at any time in the Loaded object and are available through this alias. It defaults to "template".

Example:

```
$this->load->view("view.html",array('test' => 'var'));
// You can access the object like this:
$this->template->var = $var;
Or with a custom alias:
$this->load->view('view.html',arrary(),'my_view');
$this->my_view->var = $var;
```

extension($file)

Loads the extension as explained in the Extensions chapter.

Parameter	Type	Description
$file	string	The file name or folder name relative to engine/lib/extensions/

js($file)

Includes the javascript file to the javascript autoload list.

Parameter	Type	Description
$file	string	The file name relative to content/templates/[active]/js/

css($file)

Includes the css file to the css autoload list.

Parameter	Type	Description
$file	string	The file name relative to content/templates/[active]/css/

model($name)

Loads the model into the loaded objects. This is an alternative to instanciating a model directly.

Parameter	Type	Description
$name	string	The name of the model or the table

Example:

```
$this->load->model('Blog');
$this->Blog->execute();
```

Is the same as (this one is recommended):

```
$Blog = new Blog();
$Blog->execute();
```

layout($file)

Sets the active layout file. This is an alias to Tpl::set(MAIN_TEMPLATE,$file)

Parameter	Type	Description
$file	string	The file name relative to content/templates/[active]/

template($folder)

Sets the current active template to $folder. It is an alias to Tpl::set(ACTIVE,$folder)

Parameter	Type	Description
$folder	string	The template folder name inside content/templates/

Mail

The mail class uses the PHP Mailer open source mailing class (http://code.google.com/a/apache-extras.org/p/phpmailer/) which is excellent. Mail class provides only helper functions that help with mail sending.

The default SMTP credentials are as specified in config.php variables (must change them first):

```
$config['mail_host'] = "smtp.sever.net";
$config['mail_port'] = 587;
$config['mail_user'] = "user";
$config['mail_pass'] = "pass";
$config['mail_from'] = 'noreply@yourdomain.com';
```

__construct([$params])

If params is set, this function provides a shortcut to send the email directly from the constructor function. If you're not familiar with the class I advice you to skip this function and read the other functions first as this one assumes you have basic knowledge of the class.

$params is an associative array that can have the following indexes:

Index	Required	Description
to	yes	The destinatary email address. Can have this format: John Doe <john@doe.com> Or just the plain email address
from	yes	The email sender. Can have this format: John Doe <john@doe.com> Or just the plain email address
html	no	Boolean. If set to true the email body will be sent as HTML. If set to false, it will be sent as plain text
subject	yes	The email title or subject
body	yes	The mail body. Can be HTML if html is set to true
alt_body	no	Alternate body to the HTML content.
reply_to	no	Defaults to the "from" parameter
tpl	no	The html view to render. Explained later on this chapter
context	no	The vars to send to the view "tpl"

Example:

```
Mail(array(
       'to' => 'test@test.com',
       'from' => 'noreply@test.com'
       'subject' => 'Test email',
       'body' => 'This email is a test'
```

```
));
// With just this code a very basic email will be sent
```

to($email*[,$name]*)

to($complete_info_recipients)

Sets the recipient email address. Can be called multiple times and the mail will be sent to all of them. This function is overloaded and can be called with two different parameter sets.

First overload:

Parameter	Type	Description
$email	string	The email address of the recipient
$name	string	The name of the recipient

Second overload:

Parameter	Type	Description
$complete_info_recipients	string	You can enter the email addresses of the recipients separated by commas. You can type the recipient's information as such: John Doe <john@doe.com> Or just the plain email address

Example:

```
$Mail = new Mail();
$Mail->to("john@doe.com","John Doe");
// Is exactly the same as;
$Mail->to("John Doe<john@doe.com>");
```

from($email*[,$name]*)

from($complete_info_sender)

Sets the sender email address. This function is overloaded and can be called with two different parameter sets.

First overload:

Parameter	Type	Description
$email	string	The email address of the sender
$name	string	The name of the sender

Second overload:

Parameter	Type	Description
$complete_info_sender	string	You can type the recipient's information as such: John Doe <john@doe.com> Or just the plain email address

Example:

```
$Mail = new Mail();
$Mail->from(“john@doe.com”,”John Doe”);
// Is exactly the same as;
$Mail->from(“John Doe<john@doe.com>");
```

cc($email*[,$name]*)

cc($complete_info_recipients)

Sets the copied recipient email address. Can be called multiple times and the mail will be sent to all of them. This function is overloaded and can be called with two different parameter sets.

First overload:

Parameter	Type	Description
$email	string	The email address of the recipient
$name	string	The name of the recipient

Second overload:

Parameter	Type	Description
$complete_info_recipients	string	You can enter the email addresses of the recipients separated by commas. You can type the recipient’s information as such: John Doe <john@doe.com> Or just the plain email address

Example:

```
$Mail = new Mail();
$Mail->cc(“john@doe.com”,”John Doe”);
// Is exactly the same as;
$Mail->cc(“John Doe<john@doe.com>");
```

bcc($email*[,$name]*)

bcc($complete_info_recipients)

Works exactly the same as css() but it send a hidden copy

subject($text)

Sets the mail subject

Parameter	Type	Description
$text	string	Text of the email title

body($content)

Sets the body of the message. Can be HTML or plain text.

Parameter	Type	Description
$content	string	Body content

renderBody($view*[,$context]*)

If your email is heavily HTML loaded it's best to save it as a separate view and treat it just like any other view. The view will be rendered and it's output will be set as the body of the mail automatically.

As emails are commonly not changed from template to template, all email views are saved inside the common template.

Parameter	Type	Description
$view	string	The file name of the view relative to content/templates/common/email/
$context	array	Array containing the variables to pass to the view

Example create this file inside content/templates/common/email/test.html

```
<h1>Test email</h1>
<p>Hi, {$name}. Welcome to my site!</p>
<p>Regards, My App Staff</p>
```

Now to send it:

```
$Mail = new Mail();
$Mail->renderBody('test.html',array('name' => 'Rick'));
```

attach($file)

This function attaches one file to the message. It can be called severtal times, one for each file.

Parameter	Type	Description
$file	string	This needs to be the absolute path of the file. Example: DIR_CONTENT_FILES.'uploads/test.pdf'

html($bool)

Tells php mailer wheather your message body will be treated as HTML or plain text.

Parameter	Type	Description
$bool	boolean	True or false

send($enqueue = false)

This function needs to be called after all other functions to actually send the mail.

Sending Asyncronous Mails

When sending a large number of mails, the server would delay a little bit and therefore it would seem very slow. If you would like to enqueue the email instead on sending it right away you must set $enqueue to true. Note that $enqueue defaults to false.

The solution to this problem is to enqueue this massive email messages in a table and then execute a periodical script in server side who's in charge of sending those emails.

To enable this solution we need to create a "mailer" table:

```
CREATE TABLE `mailer` (
  `id_mailer` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `to` varchar(100) DEFAULT NULL,
  `from` varchar(100) DEFAULT NULL,
  `title` varchar(255) DEFAULT NULL,
  `body` text,
  `alt_body` text,
  `html` int(1) DEFAULT '1',
  `fecha_creado` datetime DEFAULT NULL,
  `fecha_enviado` datetime DEFAULT NULL,
  `enviado` int(1) DEFAULT '0',
  PRIMARY KEY (`id_mailer`)
)
```

Then the periodical script needs to be set up as explained in the Automatic Tasks chapter.

Paginator

The pagintor class is very usefull for performing page pagination functions. As we may have already noticed, every pagination is the same code in almost every page thoughout the same site, that's why made a snippet-based approach in which the paginator class performs the paginator calculations based on the SQL query, loads the view, compiles it and returns it as plain HTML that can be then passed to the main layout.

Public Properties

Name	Default Value	Description
$sql		This is the plain SQL from which the paginator will get the total number of rows returned and make calculations
$navSql		SQL limit query generated based on the calculations
$navPgVar	pg	The $_GET var used to store the page numer in
$navOffset	1	Number of pages to show before and after the current page. Example (this will be 1 offset): << < 4 **5** 6 > >>
$navActPg	0	Actual page
$navTotalPg	0	Total pages
$navRowsPerPg	10	Results shown per page

$infLimit		Lower range limit
$supLimit		Upper range limit
$tplDir	nav/	Where is the view located relative to content/tempaltes/[active]/html/
$tplFile	paginator.html	View file inside $tplDir
$tplHtml		Rendered view output HTML
$numRows		Number of rows returned by the query

paginate()

Calculates and renders the view.

limitQuery($include_command = true)

Returns a LIMIT SQL query containing the limits of the pagination. If $include_commend is set to true, it will include the LIMIT sql query inside the string.

Example

```
$Blog = new Blog();
$Blog->execute(false); // Dont execute the query, just compile the SQL query string
// Paginate
$Pag = new Paginator();
$Pag->sql = $Blog->Sql;
$Pag->navRowsPerPg = 20;
$Pag->paginate();
// Rebuild and execute the query
$Blog->limit($Pag->limitQuery(false)); // We don't need the LIMIT sql command, just
limits
$Blog->execute(); // This is the paginated SQL query…
// Now get the paginator HTML
$paginator = $Pag->tplHtml; // This has the compiled HTML
```

Request

Request class gets information about everything sent to the server, the http request. It's a singleton class so it cant be instanciated directly.
This class is available anytime from the ModuleController using **$this->request**

Request::getInstance()

Get an instance of the Request object

get($var*[,$cast]*)

Gets the $_GET[$var] super global. In case magic_quotes is enabled, this function removes the slashes added automatically by PHP.

Parameter	Type	Description
$var	string	$_GET array index

$cast	int OR float OR string or boolean	In case this is set, it returns a casted version of the variable

post($var*[,$return_object[,$cast]]*)

Gets the $_POST[$var] super global. In case magic_quotes in enabled, this function removes the slashes added automatically by PHP.

Parameter	Type	Description
$var	string	$_POST variable index. In case the variable is itself an array you can add: "vars[subindex]"
$return_object	boolean	In case it is set to true, it will return the RequestVar object instead of the variable value
$cast	int OR float OR string or boolean	If set will cast the variable value

__get($var)

The magic get function. Checks first if a $_POST[$var] exists and if not returns $_GET[$key].

Example:

echo $this->request->name;

isAjax()

Returns true if the request made was made through AJAX or through a common AJAX library. Else will return false.

module()

Get the current module. An alias for Router::$Control['module'] or DSP_MODULE.

action()

Get the current action. An alias for Router::$Control['control'] or DSP_CONTROL.

validate()

Executes the validation of the input variables. If an error was found then a validation exception will be raised as explained in the Validation chapter.

path($index)

Return the path in the $index position. Is an alias to Router::$path[$index]

Parameter	Type	Description
$index	number	The path position

RequestVar

This class is available only for input validation and is passed by the Request's post() function when setting to true the second parameter as defined in the Input Validation chapter.

errorMsg($msg)

Sets the default error message if no particupar error message is found.

Parameter	Type	Description
$msg	string	Error message text

required()

The input must be set and be different from "". Example:

```
$this->request->post('name',true)->required();
```

unique($table,$column*[,$except[,$error_msg]]*)

Validates that the input is unique in the table and column specified.

Parameter	Type	Description
$table	string	table name to check in
$column	string	column name inside the table where the value must be unique
$except	string	if the value found equals this, it will be ignored
$error_msg	string	the error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

minLength($length*[,$error_msg]*)

The input must have at least the specified length in characters.

Parameter	Type	Description
$length	number	String minimum length
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

maxLength($length*[,$error_msg]*)

The input must have at the most the specified length in characters.

Parameter	Type	Description
$length	number	String maximum length
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

lessThan($length*[,$error_msg]*)

The input must be smaller than $length. In numeric representation.

Parameter	Type	Description
$length	number	Length limit
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

greaterThan($length*[,$error_msg]*)

The input must be greater than $length. In numeric representation.

Parameter	Type	Description
$length	number	Minimum limit
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

notEmpty()

The input must not be empty. Empty as the PHP function empty()

numeric(*[$error_msg]*)

The input must be a numeric value.

Parameter	Type	Description
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

email(*[$error_msg]*)

The input must bea valid email address.

Parameter	Type	Description
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

equals($value*[,$error_msg]*)

The input must be equal to $value.

Parameter	Type	Description
$value	string	The value to campare the input to
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

regex($regex*[,$error_msg]*)

The input must validate the specific regular expression.

Parameter	Type	Description
$regex	string	A valid regular expression as used in preg_match()
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

func($function_name*[,$error_msg]*)

The input must pass a specific validation performed by an external function.

Parameter	Type	Description
$function_name	string	The name of the function to call. This function MUST return true for the validation to be passed
$error_msg	string	The error to show when this specific validation is failed. If is not set then will use the one set on errorMsg()

val()

Returns the value of the input.

validate()

Validates this specific request variable. Generally this in not used, it is used by Request::validate().

Response

This class handles all the output that's returned to the user's browser. It's a singleton and as such can't be instanciated directly.
This class is available anytime from the ModuleController as **$this->response**

Response::getInstance()

Gets an instance of the Response object

setHeader($header)

Set a response header. This is the same as setting it via the native PHP function header() with the exception that in FALCODE headers should never be sent directly because it could cause a headers already sent error.

Parameter	Type	Description
$header	string	Header like: "Content-type: text/html".

		Note that it's not necessary to send a content-type header because it defaults to "text/html"

isJSON()

This tells FALCODE that you want to send a JSON formatted response and as such you don't want to use the default layout, propper headers are send and blank template is used.

isBlank()

Set the layout to _blank.html

getHeaders()

Get all the response headers in an associative array that are going to be sent.

setOutput($data)

Appends the $data to the output response text

getOutput()

Gets the output data

sendHeaders()

Send the headers to the browser. This function is called automatically by FALCODE, there is no need to call it.

sendOutput()

Echoes the output. This function is called automatically by FALCODE, there is no need to call it.

Router

Router class handles most of the routing of the request. Most of it's functions are not supposed to be called manually and therefore are not going to be described. If you need more information about a specific function you can go ahead and read it directly from the documentation inside the PHP file.

This is a singleton class and all of it's properties and functions are static.

Public Properties

Property	Description
$Control	Array containing request details: array('module' => '', 'control' => '', 'dir' => '', 'file' => '' ,'path' => 'hash')
$module	The requested module
$action	The requested action or control'

Constants

Constant	Description
CONTROL_VAL	This is the variable used to accept module/action details through GET or POST. It defaults to “i”
CONTROL_SEP	It is the symbol that separates the module from the action inside the “i” variable. Defaults to “/”
ROUTE_VAR	It is the _GET var populated by apache where the whole request is set. Defaults to “_route_”

Session

Session class is a singleton. It handles the start of a session and specific session configuration.

In FALCODE sessions can be stored in the database instead of separate files.

Save Sessions On Db

To enable this feature $config[‘session_save_on_db’] needs to be set to true inside config.php.

A table needs to be created with the following structure:

```
CREATE TABLE `session` (
  `session_id` char(40) NOT NULL,
  `ip` char(45) NOT NULL,
  `user_agent` varchar(120) NOT NULL,
  `last_activity` int(11) unsigned NOT NULL,
  `data` text NOT NULL,
  PRIMARY KEY (`session_id`),
  KEY `last_activity` (`last_activity`)
)
```

Change Session Settings

Session settings are established in config.php in the following variables:

```
$config['session_expire'] = 86000; // Time in seconds
// This ones prevent your session persist throughout subdomains
$config['session_name'] = "yourdomain";
$config['session_domain'] = ".yourdomain.com"; // MUST START WITH A COLON
```

All functions are called directly from the FALCODE core system and therefore the functions of the class are not going to be explained here.

Sys

The sys class handles system errors, information, loaded modules and flash messages. It is available at any time in the ModuleController using **$this->system**
System class is also used as a variable registry.

Sys::set($var,$val)

Sets a registry var

Property	Description
$var	Variable name or key
$val	Variable value

Sys::get($var)

Gets the value of a registry var

Property	Description
$var	The variable name es set in set()

Sys::setInfoMsg($msg)

Sets a flash information message to be handled in the view. This is reseted every new execution in the user's session.

Property	Description
$msg	Information message

Sys::setErrorMsg($msg)

Sets a flash error message to be handled in the view. This is reseted every new execution in the user's session.

Property	Description
$msg	Error message

Sys::getInfoMsg()

If set, get the information message

Sys::getErrorMsg()

If set, get the error message

Sys::clearInfoMsg()

Clears the information message

Sys::clearErrorMsg()

Clears the error message

Sys::path($to)

Helper function to get the absolute http URL of images, javscript and css, useful when you don't want to deal with changing templates.

Property	Tipo	Description
$to	javascript OR css OR images	The type of path to get

setError($msg)

Alias of Sys::setErrorMsg()

setInfo($msg)

Alias of Sys::setInfoMsg()

getError()

Alias of Sys::getErrorMsg()

getInfo()

Alias of Sys::getInfoMsg()

setFlash($name,$val)

Sets a flash variable to handle from the template. Available only on the next user's request.

setFlashNow($name,$val)

Sets a flash variable same as setFlash() but it is available immediatly and cleared in the user's next request.

getFlash($name)

Gets the flash variable

__get($key)

Magic function to get a system registry variable. Alias of Sys::get()

__set($key,$var)

Magic function to set a system registry variable. Alias of Sys::set()

TaskPipeline

This class is used for queuing tasks that need to be run server-side in the background.
This class is used by the sys built-in module but you can use it if you need to perform custom tasks.

Task Pipeline Table

This class relies on the following table:

```
CREATE TABLE `task_pipeline` (
  `id_task_pipeline` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `command` text,
  `executed` int(1) DEFAULT '0',
  `output` text,
  PRIMARY KEY (`id_task_pipeline`)
)
```

create($command)

This function creates a row in the table of a command to be executed

Parameter	Type	Description
$command	string	The command to execute

executeNext()

Executes the next task that hasn't been executed yet in the task pipeline

TemplateEngine

This is the class that is used for rendering all views that use the TemplateEngine Syntax

Public Properties

Name	Default	Description
$root	./	Default include path
$controller_root	./	Default controller root path
$default_extension	.html	Default extension for views

Constants

Name	Default	Description
ESCAPE_TAGS_IN_VARS	true	Escape all tags in all variables?

setGlobals ($globals)

Sets the global vars available at any time in all the views

Parameter	Type	Description
$globals	array	Associative array containing globals

load($file)

Loads the file view

Parameter	Type	Description
$file	string	Absolute path to the view

loadFromString($string)

Load the view from a string instead from a file

Parameter	Type	Description
$string	string	String containing HTML or other view format

assign($var,$val)

Assigns a variable to the view

Parameter	Type	Description
$var	string	Variable name to assign
$val	string	Value of the variable

getContext()

Gets al the context vars assigned from setContext() or assign() or __set() functions.

__set($val,$val)

Magic function alias of assign().

setContext($array)

Assign several variables at once.

Parameter	Type	Description
$array	array	Array of variables to assign

render($replace_cache)

Renders the view and returns the output generated.

Parameter	Type	Description
$replace_cache	boolean	If set to true, will replace the cache if exists

Example Usage

This is how you can render your own template manually:

```
$Tpl = new TemplateEngine();
$Tpl->load('content/template/default/html/module/view.html');
$Tpl->assign('title','test');
$Tpl->title = 'test'; // Exactly the same as above
$html = $Tpl->render();
```

Tpl

Tpl class is a registry of template vars and helper functions to get template paths.

Tpl::set($var,$val)

Add a new variable to the registry or update an existing value.

Parameter	Type	Description
$var	string	Variable name
$val	string	Variable value

Tpl::get($var)

Gets a variable value

Parameter	Type	Description
$var	string	The variable name

Tpl::templatePath()

Gets the active template directory path. For example if the active template is “default” it would return: content/template/default/

Tpl::htmlPath()

Gets the html directory path of the active template. For template “default”, it would return: content/template/default/html/

Tpl::moduleHtmlPath()

Gets the module directory path inside the html folder of the active template. For “default” it would be: content/template/default/html/module/