

PHP FREEDOM FRAMEWORK 2.0 (SKELETAL) DOCUMENTATION

Contents at a Glance

- 1.0 Introduction
- 2.0 What is New?
- 3.0 Requirements
- 4.0 Installation
- 5.0 Framework Structure
- 6.0 Creating the First Page
- 7.0 Default Administration
- 8.0 Creating and Rendering Grouped Contents
- 9.0 Using Display Blocks
- 10.0 Site Design and Themes
- 11.0 Creating and Using Widgets
- 12.0 Building Controllers

Introduction

PHP Freedom Framework is developed in Nigeria, by the project group, founded by ADEFOKUN Tomiwa Michael; the first public version (1.0) was released on the 11th of May 2009. It is a Free and Open Source project. Details of the project information is available at the project website, <http://www.phpfreedom.org>

1.0.1 Why another Fork?

A comment was posted on the project website on the 2nd day of release, titled “*Why another fork?*” and the poster asked thus: “*Why don’t you contribute directly to the Zend Framework?*” and this is the reply for him by another commentator: “*Zend is a cool framework about the best for PHP, but it is kind of killing a fly with a machine gun, with respect to developing simple websites.*”

Truly, there are quite a number of great frameworks that experienced developer could choose from; P2F would not be one of them until it is clear why the framework is developed in the first place. I have worked with Seagull and Zend Framework, and I could say that ZF is quite powerful and it becomes stronger, everyday, but it is not in any way a ready-to-use framework and it can only be used effectively by very experienced PHP developers. Seagull is also a nice one, quite nice, with lots of ready to use features, like so many other content management systems, flexibility was a trade off.

My very first concern when I was introduced to frameworks was the fact that I would end up learning these frameworks and not the programming language behind them, PHP in this case, but this holds true for almost all cases.

P2F is a unique thinking, and the architecture’s main focus is to step up web builders with not much of experience to the level of advanced developers over time. Hence features that would hide real development concepts from developers will be ignored, this means that you will be encouraged to do a few work, but you will be able to set up your website in 15 minutes as promised! So, really, you would not be learning the framework but rather you would realize the real idea behind object oriented programming, Model-view-controller architectural pattern and the core reasoning behind great application systems.

Inspired by personal experiences, coming from develop-for-passion background in a develop-for-income environment, I realized the reasons for rapid application development frameworks, particularly to achieve the following:

- Need of a standard way of reusing codes
- Be sure of maintainability
- Simultaneous development by many persons

Really, it would be a good idea to have a working system that could help out. And to the rescue come the frameworks, and there are many of them, so many that developers need to make very critical decisions in choosing their favorite development framework.

I needed a framework that would allow me get jobs done on time and teach me what it does in the background, but there was none. So I decided to create one.

1.0.2 Online Resource

Thanks to Google Inc. for providing the Google Code, a project hosting service for open source codes. P2F code is hosted at <http://code.google.com/p/phpfreedom>, downloads and bugs tracking for the project are shared on there. You are invited to be a part of the project; Subversion Repository for projects is also available at the site, to enable version control for the project and also enhance collaborative development, when people like you decide to help out.

Downloads: <http://code.google.com/p/phpfreedom/downloads/list>

Wiki: <http://code.google.com/p/phpfreedom/wiki/list>

Issues Track: <http://code.google.com/p/phpfreedom/issues/list>

Project Blog: <http://phpfreedom.blogspot.com>

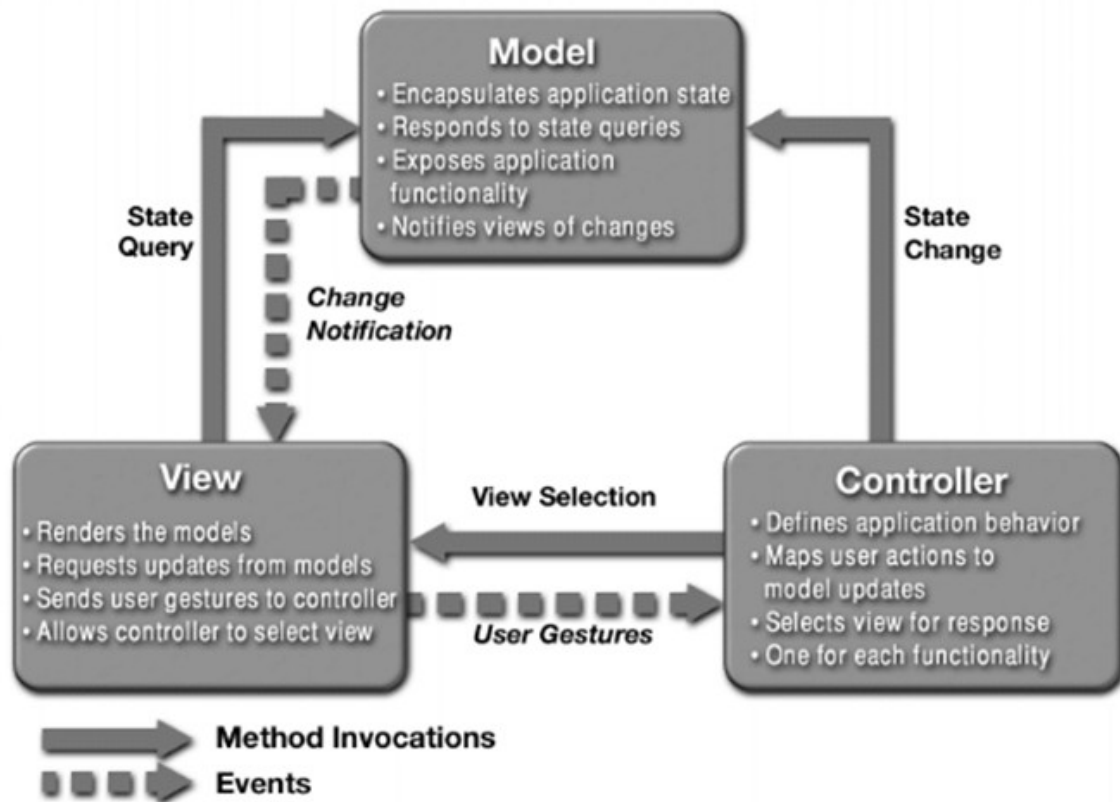
PHP Website: <http://www.php.net>

1.0.3 Model-view-controller Development Pattern

Several problems can arise when applications contain a mixture of data access code, business logic code, and presentation code. Such applications are difficult to maintain, because interdependencies between all of the components cause strong ripple effects whenever a change is made anywhere. High coupling makes classes difficult or impossible to reuse because they depend on so many other classes. Adding new data views often requires re-implementing or cutting and pasting business logic code, which then requires maintenance in multiple places. Data access code suffers from the same problem, being cut and pasted among business logic methods.

The Model-View-Controller design pattern solves these problems by decoupling data access, business logic, and data presentation and user interaction.

The process is described diagrammatically below:



- **Model** - The model represents enterprise data and the business rules that govern access to and updates of this data. Often the model serves as a software approximation to a real-world process, so simple real-world modeling techniques apply when defining the model.
- **View** -The view renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes. This can be achieved by using a push model, where the view registers itself with the model for change notifications, or a pull model, where the view is responsible for calling the model when it needs to retrieve the most current data.
- **Controller** - The controller translates interactions with the view into actions to be performed by the model. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

Source: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

As at the time of writing, the project uses a generic model class, that developer can extend. But there would not be a need for this case in a not extremely complex project.

What is New?

Since the 2.0 version, about everything is new – this is a wonderful development, and I am proud to say that I am very happy about it. Let us briefly describe these changes and the new features.

- **Modular Structure:** This structure allows you to create your application in some portable bits. So a module is a kind of package for related controllers or managers and there view templates.
- **Enhanced Security:** The system now separates the core application components from the web component. Therefore the application modules, configuration files and other non-static-html components are now in the /application directory that should not be accessible via a web URL. This is generally considered a major security feat.
- **Multiple Themes/Design One Application:** Your application can have various themes or skins. These skins can be shared or obtained from a different application. This is achievable by completing separating the application from the design.
- **Distributed Databases:** In the world of distributed computing, having a framework that talks to various databases at the same time is inevitable. PHP Freedom Framework does. And it very useful when you consider building applications that would read data from a replicated data systems or clusters.
- **Data Modeling:** The framework now comes with a handy data modeling solution that allows you construct how you data are received from the data store programmatically. The system's data model class can be extended for further functionalities.
- **Basic AJAX Features:** For developers that wish to build no-reload apps, P2F is for you. Though there are a number of tools and JavaScript frameworks that can be used to achieve this concept. P2F does not use or bundle such libraries. And this is to allow developers go with their various choices at will.
- **Widgets and Plug-in System:** You can now have portable view components that can be used anywhere on the templates, or layout. You can write your own plug-in using the widgets API and may be distribute them.

Requirements

Running a web application on your PC requires that you have a working web development environment on your machine. The basic components are:

- A web server such as Apache 2.0: www.apache.org
- PHP5: www.php.net
- MySQL database version 5+: www.mysql.com
- Web Browser. Mozilla Firefox would be suitable because of the developer toolbar. But you could as well get the developer toolbar for IE
- Text Editor

E.g. Notepad++ <http://notepad-plus.sourceforge.net>

There are some software that bundles the trio of Apache, PHP and MySQL, pre-configured. You may download one and just install it on your machine and you are ready to go, the following are recommended:

- Xampp Server: www.apachefriends.org/en/xampp.html

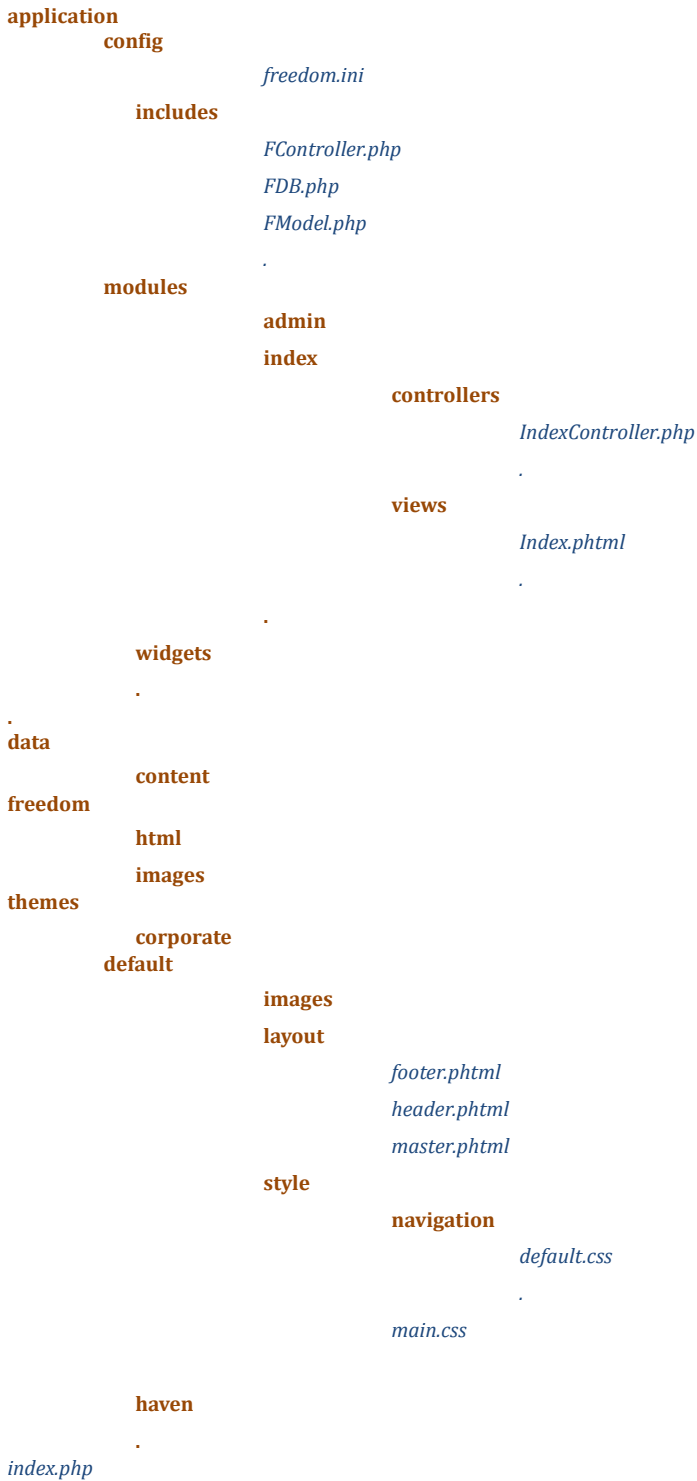
Installation

- Download the latest release of the framework from Google Code at:
<http://code.google.com/p/phpfreedom/downloads/list>
- Unpack the archive onto your server's web root directory
- Create a database to use with your project and run the /phpfreedom.sql against the database.
- Open /application/config/freedom.ini and edit it to configure the framework to your local environment. Save the changes.
- Open <http://localhost/phpfreedom> from your web browser, and you already have a website.
- First, we need to start by securing your application... All the backend functionalities of your application shall reside in the /application folder. This folder contains and will contain your application modules. In order to ensure some security, this folder should not be accessible from a url! Therefore,
 - Move this folder, and put it anywhere outside of your web server
 - Open /index.php and set \$path_to_application_directory = /new/path/to/application and save
- To login as administrator, click login and sign in with the following credentials:
 - Username = administrator
 - Password = administrator

To change the admin credentials, expand the administrator menu and click --Manage Users-- and click the edit icon.

Framework Structure

The Freedom Tree 2.0



The structure of PHP Freedom Framework focuses on extensibility. As from 2.0 it has been based on a modular structure, and the design has been completely separated from the application itself. The main application resides in the /application directory which should be moved outside the web server to enhance security.

If the URL <http://localhost/phpfreedom/admin/content/list> is requested from a browser; the processes starts from index.php in The Freedom Tree above. A controller instance from FController.php is created as \$master, The URL is digested and the module, controller and action are determined, in this case admin, content and list respectively.

The ContentController.php of admin module is called and an instance of it is created, then the listAction() method of the instance is called.

And finally, it is time to display the result of the merry-go-round. The first is to determine the theme and layout to use. There are various ways to set a theme and this can be done at various points during the lifecycle of the instance using the setTheme(\$theme) method. The default theme is set in the freedom.ini. Meanwhile, you could inform the application not to use a layout for this instance, in case you want to send the response as XML, JSON or any other format of web data exchange. To do this you must call the noLayout() method of the controller, this will set the \$layout variable of the instance to **false**. You can also have more than one layout per theme, the default is master.phtml. To choose a different layout you can use the setLayout(\$layout) method.

Once the appropriate theme and layout is determined, the view template for the result of listAction() is defined in the method using \$this->assign('view',\$template). In this case, say \$template = 'content_list'.

Meanwhile, the view template file is:

```
/application/module/admin/views/content/list.phtml
```

If \$template = 'list', the view template file would be:

```
/application/module/admin/views/list.phtml
```

The underscores in the template name are replaced by "/".

The assign(\$name,\$value) method is used to set properties for the view template. Therefore, \$this->assign('title','First Page'); will allow you get the title on the view page using \$this->title.

In the layout, \$master->displayContent() is called to render the requested content in the layout, say master.phtml.

A developer can choose to create more modules and themes at will. Details of this will be discussed latter or in a revision of this documentation, as this is just a guide to using PHP Freedom Framework, hoping that we could get help with documentation of the project.

Creating your first page

If you have been able to install it on your machine, it is running properly and you have secured your application, great! The next thing is creating pages using the content manager.

To create pages, you must be the logged on as the administrator.

Expand the Administrator link and click --Content manager--

Click --Add New-- to create a new content, there are a lot of fields on the content management form but you can ignore most of them for now and just fill the following category as follows:

Title = My First Page

Category (Page Name) = firstpage

The value for the category should be unique.

Subsections Title = Subsections of My First Page

Note: This is only necessary if the page has sections, there will be more about this shortly.

FULL CONTENT DETAILS = anything that you may like.

You may attach some pictures by click --add attachments-- for as many pictures that you may wish to add.

Click the **Save** button to save the content.

Attach the page to a menu link using Menu Manager

On the administrator links click --Manage Menu--, the Menu Manager is used to create and organize the main menu of the site. It is a multi level system.

Click --Add New-- and fill the following fields

Title = First Page

URL = index/pages/display/?page=firstpage

This means, the framework should load use the PagesController and render the page with category (Page Name) firstpage. There will be more explanation on this.

Leave **internal URL** Selected, this means that the referenced resource is within the site or application.

For now, set the **Parent Link** to "Project Demo."

Click the **Save** button to create the new link.

The -- First Page—will now be available for use on the navigation under Project Demo, click it and see the page.

Setting display template for the page

You can change the template that the page is using from the content management.

Follow this link --administrator-- -> --Content Manager-- and Click the **Edit Link** attached to the item whose Category (Page Name) = FIRSTPAGE.

Edit the page as follows:

Page Template = [Select a template and save, repeat this step for other templates]

Let me briefly explain the templates:

If you browsed the project directory and go to /freedom/modules/index/views, there resides all the view pages used and to be used for index module of your project, you can create a folder for every controller. Since the content manager will use so many templates, there is a pages folder in the views, and it has 3 templates for version 1.02, it also has a folder named sections, which contains the templates for the subsections of a page generated from the content manager. For version 1.02, there are 6 templates.

You can write your own templates, too, and put them in the appropriate places as described above and it will be available to be selected in the content manager, either at [Page Template] or [Subsections Template] fields. We shall get to that part latter.

Creating subsections for a page

Follow the link --administrator-- -> --Content Manager-- -> --Add new-- and fill the form as follows:

Title = First Section of My First Page

Pane Label = First Section

Category (Page Name) = firstpage

Sub Category (Subsection Name) = firstsection

The sub category should be unique for every category or subsection i.e. for a category, "firstpage"; you can only have one sub category, "firstsection".

FULL CONTENT DETAILS = The first section of my first page.

Add other contents and attach pictures

Click the **Save** button to create the subsection.

Create about three sections for the firstpage by repeating the steps above.

Then go to --First Page-- on the main menu to view the result.

Now you may then want to change the way the subsections are displayed. Go to --administrator-- -> --Content Manager-- and Click the **Edit Link** icon attached to the item with Category (Page Name) = FIRSTPAGE and set the following to your preference:

Page Template = [Select: VerticalBoxedImgPage]

Subsections Template = [Select: Page_Sections_Panes]

Save the changes and click –First Page—in the main menu.

Also note that:

- <http://localhost/phpfreedom/index/pages/display/?page=firstpage&subsection=firstsection> will display the content of the “firstsection” subcategory of the page “firstpage”.
- <http://localhost/phpfreedom/index/subcontent/?category=firstpage> will display all the subsections of the page

Description of Fields in Content Management Wizard

The following describes the fields available in the content management wizard, their usage and when to use them.

1. **Title:** This field defines the title of the content. It can be ignored but this would imply that content manager will not automatically render a title for the content.
2. **Pane Label:** This is only required if the content being created will be used in a tabbed panes layout or accordion panes layout as a tab label. It can be ignored if the content has a title but is recommended.
3. **Content Type:** This is useful if you want to categorize the content into a special class which could be used as a group such as news. The content types are defined by the administrator. It can also be ignored.
4. **Category (Page Name):** This is a compulsory field that must be unique; it is required so that a developer can access the content directly by using the category name provided.
5. **Sub Category (Subsection Name):** This field is only required if the content is a sub category of a category and it must also be unique for each category. If the content is the main category the subcategory name must be the same as the category name.
6. **Sub Category Order:** This field is used to specify the preferred order for the subcategories. The value must be an integer, and the content with the least value takes precedence over the one with a higher value. It can also be ignored.
7. **Sub Sections Title:** This is the title to be given to the subsection area of the page if the content has subcategories. It can also be ignored.
8. **Page Template:** This defines the display template or design to be used for the content. It defaults to using "Parent page template", this means that if the content has a parent content, the page template of the parent should be used. Another primary option is to use default content type template; i.e. the Page Template defined for the chosen content type should be used. This can be ignored.
9. **Subsections Template:** Defines the template to be used to display the subcategories of the content on the page. It also has 2 primary options, i.e. the parent subsections template and the content type subsections template. This can also be ignored.
10. **Permitted Roles:** This defines the access roles for the content; only persons with the required role(s) will have access to the content. It is also not required, and can be ignored.
11. **Summary:** This is a field for brief summary of the content. It is useful for highlights but it is not required.
12. **The Content Details:** The rich text editor is used to format the body of the content. It is recommended that you get detailed information about using rich text editors. Basically, the main content of the page goes here.
13. **Create Date:** This field is not editable; it tells the user the date that the content was created.

14. **Expiry Date:** If set, the content will not be available for viewer after the stipulated date.
15. **Attachments:** This will allow users attach files, basically images to the content. Each attachment has a optional label. The attachments can be removed or updated.
16. **Show Last Update Info:** This will allow the system to include the last update date on the page when it is set to yes.

Default Administration

When logged on as the administrator you will be able to perform basic system management such as navigation, user and access roles management, security, content management, feedbacks and e-support setup. The related links are by default under the --Administrator-- link. Let us look at each of these in details.

- **Manage Menu:** Add, edit or delete main navigation items. The edit or add form has the following fields:
 - **Title:** This is the text to be displayed for the item.
 - **URL:** This is the URL that the link will point to. There are 2 options
 - **Internal URL:** For links that are internal to the project. It has the following format [module/controller/action/?param1=hello]
 - **External URL:** For links outside the project.
 - **Parent:** If set, the menu will be a child to the parent.
 - **Target:** To specify the target of the link, for example, to open the link in a new browser tab, set the field as “_blank”.
 - **Permitted Roles:** Select the users groups that will have access to the link.
 - **Order:** This is useful to order the menu items. It takes as integer value; the link with a lower value takes precedence over that with a higher value.
 - **Available in top bar:** If set, the menu item is available in the top navigation bar.
 - **Load in AJAX Mode:** If this option is set to yes, when the link is click, the result of the action is dynamically loaded into the page without refreshing the whole page.
- **Manage Users:** Add, edit, delete, set permissions and manage credentials for registered users.
- **Manage Roles:** Use this to manage set up user roles for the application. It has 2 fields, Role Key and Role Name – There are 2 primary roles the Admin and Ordinary User, the role key for admin must always be 1 and the role key ordinary user must always 0.
- **Security Setup:** This is used to define user access security for various resources. It has the following parameters:
 - **Module:** The module for the resource
 - **Manager:** The controller or manager name for the resource.
 - **Action:** The resource action in the Manager/Controller class.

- **Permitted Roles:** Select the user roles that are to have access to this resource.
- **Content Types:** This allows you to create some predefined content groups that can be used from the content manager. E.g. News. This enables a developer to customize the display of such group content. It has the following fields:
 - **Type Name:** The name to be given to the group of content.
 - **Type Key:** This is the unique key that could be used to access or identify this group of content.
 - **Default Listing Template:** This is the default template to use when you display a particular content group as a list. These templates are available in the /freedom/modules/index/views/page/sections directory.
 - **Default Template:** This is the default display template to be used for the contents created under this group. These templates are available in the /freedom/modules/index/views/page directory. You can also create you custom templates in this directory.
 - **Default Sections Template:** This is the default template to be used for the subsections of the contents created under this group. These templates are available in the /freedom/modules/index/views/page/sections directory. You can also create you custom templates in this directory.
- **Content Manager:** This has been explained above.
- **User Feedbacks:** This is a built in functionality that allows you get feedbacks from visitors to the website.
- **E-Support Setup:** E-Support setup is a link to the E-Support Portal provided by Soft-and-smart Technologies. This will enable you create the live chat support for the project, E-Support is a hosted applications service that you can integrate into you application for free.

Creating and Rendering Grouped Contents

For example, news can be categorized as grouped content. And it is so easy to create and display such information.

First, go to Administrator -> Content Types and create a new content type or group. Assign appropriate templates if need be. The group that you just created will be available for use by the content manager.

Go to Administrator -> Content Manager and create contents for this group. This is done by setting the content type for the item to the desired group. Set the templates to use the default type templates for the page and its subsections.

You can create as many as possible content for a particular group or type.

Displaying this content is also pretty easy. It is accessed directly using, assuming the group is: news <http://localhost/phpfreedom/index/groups/?group=news>

This will display the list of content in this group, using the default listing template defined in the content type setup. It could be news, events, job vacancies etc.

Using Display Blocks

Since the beginning of this documentation we have not written a piece of code, we have only been working with the inbuilt wizards. Now we may start doing some little coding, using some content management APIs of the framework.

Every controller/action or content or even content groups can be rendered as blocks in various positions on the page. For example, you can have your news highlights and events as different blocks on the same page and you can have as many as possible blocks rendering different types of contents.

Let us examine the code for some scenarios.

Display Block for Content Group

News is a content group, so we can display the top 3 news as a block. The code for creating blocks should be placed inside a view template. To achieve our goal, the following PHP code is required:

```
<?php
//create the block object
$news = $this->buildBlock('groups');
//set required properties
$news->setProperty('group','news');
$news->setModelProperty('limit',3);
$news->setProperty('title','Recent News Highlights');
$news->setProperty('template','page_sections_highlights');
//build the block
$news->build();
//display the content or records
$news->displayContent();
?>
```

This is very simple.

Display Block for any Content Item

For any content item created there is a content category or name and apart from accessing it from the URL, it can as well be rendered as a display block. The following PHP code illustrates two different content pages being rendered as 2 display blocks on the same page:

```
<?php
//Create the first block object $page1
```

```

$page1 = $this->buildBlock('pages');
//set the page category or name. This is not the same as group.
$page1->setProperty('page','home');
$page1->build();

//Create the second block object $page2
$page2 = $this->buildBlock('pages');
//set the page category or name. This is not the same as group.
$page2->setProperty('page','contact');
$page2->build();

//Create the spaces for the blocks on the page using HTML
?>
<table>
    <tr>
        <td><?php $page1->displayContent(); ?></td>
        <td><?php $page2->displayContent(); ?></td>
    </tr>
</table>

```

The buildBlock(\$param) method requires the first parameter which must be a controller name. Meanwhile, you can pass an array of parameters to it, such as:

```

<?php
//render from index module, page controller and list
$params = array('module'=>'index', 'controller'=>'group', 'action'=>'list');
//module = index controller = group action = list
?>

```

Note that the word controller and manager are used alternately, but in the case above it must be controller.

The setProperty(\$name,\$value) method of the block object takes two parameters, the \$name as the property name and \$value as the value of the property. These are instance properties of these controllers or managers. Examples are action, name, limit and template etc. Refer to the controller that you intend to use for the block for guidance about using the properties. To check if you can set a property, you can use the getProperty(\$name) method of the block to check, it will return false if the property is not allowed.

E.g. `$page = $this->buildBlock('page'); $page->getProperty('template')`

Site Design and Themes

Assuming you are done with the content and you have created all required blocks and you have placed them where you want them... everything seems nice and right. Sure you don't want to keep any of the default themes – you, perhaps want your own.

Copy the default theme or any other theme that looks closest to what you desire, and give it your choice name. You are about to create a PHP Freedom framework theme that can be reused by anyone and in any project by just dropping it in the theme folder. It will be available in the theme selector. To make it your default theme, go to the `/application/config/freedom.ini`, the themes section set default to your new theme.

You will need good CSS and HTML skills to deliver an appealing theme, but it is worth the effort since, no one cares about the power of your framework, but how nice your website looks. Refer to the structure of the themes in The Freedom Tree.

Make sure that all your skins images and animations, CSS and JavaScript and what have you that are used to develop your theme are within its directory for portability.

Creating and Using Widgets

The idea of widgets is introduced in version 2.0 to enable reusability of view components across themes and layouts programmatically. Example of widgets are TopNavBar and ThemeSelector, there will be many more in the future. You can create your widgets, by following the simple steps described in this section.

Creating a Widget

A widget contains at least one Class that extends Widget.php. This means that it can be more than a file. The base directory for widgets is /application/widgets and it can be accessed via the constant P2F_WIDGETS_DIR.

You could even use images, CSS files, JavaScript files and whatever kind of static files that you wish within your widgets. Since widgets are part of the application and cannot be accessed via the web server, these html components should be located in a folder with the same name as the widget, under /freedom/html/widgets, this path can be accessed via the constant P2F_WIDGETS_PATH.

For a single file widget its name should be the same as the class name and also the file name, i.e. the widget ThemeSelector resides in

/application/widgets/ThemeSelector.php which contains a class ThemeSelector that extends Widget.php. It must have a class variable \$widget, and public method build() that builds the result as a HTML string that is contained in an element with attribute class set to the widget name and assigns the string to \$widget.

But a widget that contains many files must be packaged in a folder. This could even be extended to package multiple widgets into one. That has not been implemented, yet, but it is obviously very possible. Suppose that you package your widget in MyFlash folder, which has its base class as TheFlash in /application/widgets/MyFlash/TheFlash.php that satisfies the conditions of a widget above will be named MyFlash_TheFlash.

The idea of multiple widgets in one package means that you could have another widget base class SecondFlash, in /application/widgets/MyFlash/SecondFlash.php and named MyFlash_SecondFlash. It should work. I will test it right away...

It worked!

Note The idea is to enhance portability, so you can drag your widgets from one app to another or even mail it to a friend!

Using a Widget

Basically, a view object such as your view templates has the \$widget = FController::createWidget(\$widgetName) which returns an instance of the widget requested. Call \$widget->build(); just before \$widget->display(). You could also get your widget as a string using \$widget->toString().

Building Controllers

Building custom controllers require some knowledge of PHP; controllers are basically subclasses of FController that reside in the controller folder of a module. The name of the controller is the same as the class name and must be suffixed by Controller, e.g. IndexController.

A simple controller requires at least one method, the indexAction() method, which is the default action called when the controller is requested.

The following is an example of a controller:

```
<?php
class IndexController extends FController{
function __construct(){
    $this->setSiteTitle('Demo');
}
    function indexAction(){
        //...
    }
    function testAction(){
        //...
    }
}
?>
```

Assuming the code above is saved in:

/application/modules/demo/controllers/IndexController.php;

It would be accessed via the following url:

<http://localhost/phpfreedom/demo/index>;

and the indexAction() method is called. To access the testAction() method use:

<http://localhost/phpfreedom/demo/index/test>

Interfacing Database Tables

To build a controller that interfaces a database table, you may set the model property of the controller.

Considering a table `demo_employees`, you may create a model over the table as thus:

```
require_once 'FModel.php';
```

```
$this->model = new FModel('employees',demo_employees');
```

```
/*
```

The first parameter of the `FModel` constructor specifies the model name, and the second parameter specifies the table, which is being modeled.

```
*/
```

```
//Set the value of the primary key
```

```
$this->model->setPrimaryKey('employee_id');
```

```
/*
```

This part is ignored because we are facing one table, which is defined in the constructor.

```
*/
```

```
$this->model->setMainSelectFrom('select * from demo_employees');
```

```
$this->model->setCountSelectFrom('select count(*) from demo_employees');
```

```
/*
```

If the search criteria are not provided, the system generates it if the table is specified. But it is advised that you set the search criteria.

```
*/
```

```
$this->model->setSearchCriteria(array('employee_firstname','employee_lastname'));
```

```
$this->model->setLimit(20);
```

```
$this->model->addWhere("employee_id < 100");
```

```
$this->model->setOrderBy('employee_lastname,employee_firstname,employee_id');
```

Using View Templates for Controllers

The view templates used for a particular controller, say, *index* controller of the *demo* module are in the folder:

```
/application/modules/demo/views/index
```

A view template is a phtml file; you may have view template for each action in the controller. To assign a template to be used, you may use the following method:

```
$this->assign('view','index_index');
```

Which simply means, use the template file:

```
/application/modules/demo/views/index/index.phtml
```

Alternatively, you can assign the templates once for various actions, by setting the *'templates'* property of the controller as an associative array. The employees controller of the demo module has its template views set this way:

```
$this->templates = array(  
    'list'=>'employees_list',  
    'edit'=>'employees_edit',  
    'add'=>'employees_edit'  
);
```

Which simply means, that the system should use:

```
/application/modules/demo/views/employees/list.phtml
```

Using the Inbuilt CRUD Functionality

FController class has the; create, read, update and delete functionalities, inbuilt. Therefore, if your controller does not require these functionalities you may override the following action methods:

```
function listAction(){}  
function addAction(){}  
function editAction(){}  
function saveAction(){}  
function deleteAction(){}
```

You could as well override the `prepare()` method, this method is called each time you create or update a record.

The `prepare()` method, take a single parameter, which is the in-bound data. So you can perform validations and filters on the data. The method also returns the fine-tuned data, which is eventually committed into the data store.