Modular Extensions HMVC

Modular Extensions makes the CodeIgniter PHP framework modular. Modules are groups of independent components, typically model, controller and view, arranged in an application modules sub-directory that can be dropped into other CodeIgniter applications.

HMVC stands for Hierarchical Model View Controller.

Module Controllers can be used as normal Controllers or HMVC Controllers and they can be used as widgets to help you build view partials.

Features:

All controllers can contain an $autoload class variable, which holds an array of items to load prior to running the constructor. This can be used together with module/config/autoload.php, however using the $autoload variable only works for that specific controller.

```
<?php

class Xyz extends MX_Controller

{

    $autoload = array(

        'helpers'   => array('url', 'form'),

        'libraries' => array('email'),

    );

}
```

The Modules::$locations array may be set in the application/config.php file. ie:

```
<?php

$config['modules_locations'] = array(

    APPPATH.'modules/' => '../modules/',

);
```

Modules::run() output is buffered, so any data returned or output directly from the controller is caught and returned to the caller. In particular, $this->load-

>view() can be used as you would in a normal controller, without the need for return.

Controllers can be loaded as class variables of other controllers using $this->load->module('module/controller'); or simply $this->load->module('module'); if the controller name matches the module name.

Any loaded module controller can then be used like a library, ie: $this->controller->method(), but it has access to its own models and libraries independently from the caller.

All module controllers are accessible from the URL via module/controller/method or simply module/method if the module and controller names match. If you add the _remap() method to your controllers you can prevent unwanted access to them from the URL and redirect or flag an error as you like.

Notes:

To use HMVC functionality, such as Modules::run(), controllers must extend the MX_Controller class.

To use Modular Separation only, without HMVC, controllers will extend the CodeIgniter Controller class.

You must use PHP5 style constructors in your controllers. ie:

```
<?php

class Xyz extends MX_Controller

{

    function __construct()

    {

        parent::__construct();

    }

}
```

Constructors are not required unless you need to load or process something when the controller is first created.

All MY_ extension libraries should include (require) their equivalent MX library file and extend their equivalent MX_ class

Each module may contain a config/routes.php file where routing and a default controller can be defined for that module using:

```
<?php

$route['module_name'] = 'controller_name';
```

Controllers may be loaded from application/controllers sub-directories.

Controllers may also be loaded from module/controllers sub-directories.

Resources may be cross loaded between modules. ie: $this->load->model('module/model');

Modules::run() is designed for returning view partials, and it will return buffered output (a view) from a controller. The syntax for using modules::run is a URI style segmented string and unlimited variables.

```
<?php

/** module and controller names are different, you must include the method name also, including 'index' **/

modules::run('module/controller/method', $params, $...);


/** module and controller names are the same but the method is not 'index' **/

modules::run('module/method', $params, $...);


/** module and controller names are the same and the method is 'index' **/

modules::run('module', $params, $...);


/** Parameters are optional, You may pass any number of parameters. **/
```

To call a module controller from within a controller you can use $this->load->module() or Modules::load() and PHP5 method chaining is available for any object loaded by MX. ie: $this->load->library('validation')->run().

To load languages for modules it is recommended to use the Loader method which will pass the active module name to the Lang instance; ie: $this->load->language('language_file');

The PHP5 spl_autoload feature allows you to freely extend your controllers, models and libraries from application/core or application/libraries base classes without the need to specifically include or require them.

The library loader has also been updated to accommodate some CI 1.7 features: ie Library aliases are accepted in the same fashion as model aliases, and loading config files from the module config directory as library parameters (re: form_validation.php) have beed added.

$config = $this->load->config('config_file'), Returns the loaded config array to your variable.

Models and libraries can also be loaded from sub-directories in their respective application directories.

When using form validation with MX you will need to extend the CI_Form_validation class as shown below,

```
<?php

/** application/libraries/MY_Form_validation **/

class MY_Form_validation extends CI_Form_validation

{

    public $CI;

}
```

before assigning the current controller as the $CI variable to the form_validation library. This will allow your callback methods to function properly. (This has been discussed on the CI forums also).

```
<?php

class Xyz extends MX_Controller

{

    function __construct()

    {

        parent::__construct();


        $this->load->library('form_validation');

        $this->form_validation->CI =& $this;
```

```
    }
}
```

View Partials

Using a Module as a view partial from within a view is as easy as writing:

```
<?php echo Modules::run('module/controller/method', $param, $...);
?>
```

Parameters are optional, You may pass any number of parameters.

Modular Extensions installation

1. Start with a clean CI install
2. Set $config[‘base_url’] correctly for your installation
3. Access the URL /index.php/welcome => shows Welcome to CodeIgniter
4. Drop Modular Extensions third_party files into the CI 2.0 application/third_party directory
5. Drop Modular Extensions core files into application/core, the MY_Controller.php file is not required unless you wish to create your own controller extension
6. Access the URL /index.php/welcome => shows Welcome to CodeIgniter
7. Create module directory structure application/modules/welcome/controllers
8. Move controller application/controllers/welcome.php to application/modules/welcome/controllers/welcome.php
9. Access the URL /index.php/welcome => shows Welcome to CodeIgniter
10. Create directory application/modules/welcome/views
11. Move view application/views/welcome_message.php to application/modules/welcome/views/welcome_message.php
12. Access the URL /index.php/welcome => shows Welcome to CodeIgniter

You should now have a running Modular Extensions installation.

Installation Guide Hints:

-Steps 1-3 tell you how to get a standard CI install working - if you have a clean/tested CI install, skip to step 4.

-Steps 4-5 show that normal CI still works after installing MX - it shouldn’t interfere with the normal CI setup.

-Steps 6-8 show MX working alongside CI - controller moved to the “welcome” module, the view file remains in the CI application/views directory - MX can find module resources in several places, including the application directory.

-Steps 9-11 show MX working with both controller and view in the “welcome” module - there should be no files in the application/controllers or application/views directories.

FAQ

Q. What are modules, why should I use them?

A. (http://en.wikipedia.org/wiki/Module)

(http://en.wikipedia.org/wiki/Modular_programming)

(http://blog.fedecarg.com/2008/06/28/a-modular-approach-to-web-development)

Q. What is Modular HMVC, why should I use it?

A. Modular HMVC = Multiple MVC triads

This is most useful when you need to load a view and its data within a view. Think about adding a shopping cart to a page. The shopping cart needs its own controller which may call a model to get cart data. Then the controller needs to load the data into a view. So instead of the main controller handling the page and the shopping cart, the shopping cart MVC can be loaded directly in the page. The main controller doesn’t need to know about it, and is totally isolated from it.

In CI we can’t call more than 1 controller per request. Therefore, to achieve HMVC, we have to simulate controllers. It can be done with libraries, or with this “Modular Extensions HMVC” contribution.

The differences between using a library and a “Modular HMVC” HMVC class is: 1. No need to get and use the CI instance within an HMVC class 2. HMVC classes are stored in a modules directory as opposed to the libraries directory.

Q. Is Modular Extensions HMVC the same as Modular Separation?

A. Yes and No. Like Modular Separation, Modular Extensions makes modules “portable” to other installations. For example, if you make a nice self-contained model-controller-view set of files you can bring that MVC into another project by copying just one folder - everything is in one place instead of spread around model, view and controller folders.

Modular HMVC means modular MVC triads. Modular Separation and Modular Extensions allows related controllers, models, libraries, views, etc. to be grouped together in module directories and used like a mini application. But, Modular Extensions goes one step further and allows those modules to “talk” to each other. You can get controller output without having to go out through the http interface again.