**CodeIgniter \_remap method – Power and Benefits**

If you are using [**Codeigniter (CI)**](http://www.codeigniter.com) then you will happy to know that, CI has a built in method named **\_remap** which allows you to overwrite the behavior of calling your controller methods over URI. Usually the **second segment of the URI** determines which function in the controller gets called.

Now think about the following factors:

* You want to **hide your original method name** from the URI
* You want to **named second segment of your URI** as you like
* You want to make a control over your controller methods so that you can **decide which method should be called using URI**

Now it’s time to talk in details.

First from the above factors is hiding original method name from URI. Yes, you can do it easily by using the \_remap method. Let one of your controller methods is **display\_successful\_message** which you should typically browse as:  
http://anmsaiful.net/blog/**display\_successful\_message**  
But you want to design your URI as follows  
http://anmsaiful.net/blog/**successful**  
So that it calls the method **display\_successful\_message**

Now the second factor is naming second segment of URI as you like. You know that **identifiers** (name of variable, function, method, constant etc.) does not allow using any character except the following rule:  
**^[a-zA-Z\_]+[a-zA-Z0-9\_]?$**  
So you could not use any other character rather than alpha, numeric or underscore. But you like to use the dash character ‘-‘ in URI as a replacement of the underscore character ‘\_’.  
More practically let one of your controller methods is **about\_me** which you should typically browse as:  
http://anmsaiful.net/blog/**about\_me**  
But you want to design your URI as follows  
http://anmsaiful.net/blog/**about-me**  
So that about\_me method is get called. Note that you can also do it by CI’s URI routing feature. But it’s really paining and using \_remap it’s become easier and organized.

Finally the third factor. You are going to make a control over your controller methods. You like to create a method’s **white list** which will be allowed to call using URI. Just create the list under the \_reamp method. Rest of the task will do by the \_remap. As for example you have a method named **secure\_method** and you don’t like to call the method using URI. To do so don’t include the method in your white list. You can also do it another way. Just place an underscore character as a prefix of your method name (**\_secure\_method**) so that controller ignores calling the method using URI.

Before using the \_remap method you should know that \_remap will always get called regardless of what your URI contains. It overrides the normal behavior in which the URI determines which function is called, allowing you to define your own function routing rules.

Many lectures have delivered already. Now it’s time to show the practical.  
Look at the following script:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | class Blog extends Controller  {      function \_remap( $method )      {          // $method contains the second segment of your URI          switch( $method )          {              case 'about-me':                  $this->about\_me();                  break;                case 'successful':                  $this->display\_successful\_message();                  break;                default:                  $this->page\_not\_found();                  break;          }      }        function index()      {          // ---      }        function about\_me()      {          // ---      }        function display\_successful\_message()      {          // ---      }        function page\_not\_found ()      {          // ---      }        function secure\_method()      {          // ---      }        function Blog()      {          parent::Controller();      }  } |

Now review the previous factors. The examples I discussed about each factor are applied on the above script. I think now it is easier to understand how to use \_remap and what’s it benefits and power.

[**ANM\_CI**](http://anmsaiful.net/blog/anm_class/anm_ci.html) is a PHP class which is developed to use as a CI’s user library. In the class there is a method named **remap** which will make your life easier to use the CI’s \_remap method.

Be Sociable, Share!

# Inhibitor

## An error handler for a CodeIgniter Application

Inhibitor catches any parse, fatal errors or exceptions of your application and handles them properly.

By "properly" means:

* formats the error message
* logs the error
* mails the error
* redirects to a "placatory" view

## Installation

Note that you can download Inhibitor with CodeIgniter 2.1.2 and with an example page.

Now, if you want to use Inhibitor to your application, copy the following files to your app:

controllers/

inhibitor\_handler.php <!-- controller that logs, mails and redirects-->

hooks/

inhibitor\_hook.php <!-- the hook class that catches the errors and formats the message-->

views/

inhibitor.php <!-- a custom view - here you can use yours-->

Edit your config file to activate hooks and enable logging:

$config['enable\_hooks'] = TRUE;

$config['log\_threshold'] = 1;

Edit lines 67, 68 and 69 of inhibitor\_handler.php to setup the email.

## Requirements

Inhibitor is tested with CodeIgniter 2.1.2

**What is the use of Core folder in Application folder** [CodeIgniter](http://ellislab.com/codeigniter) is an PHP framework built on the MVC design pattern. There is no doubt that it is popular, it's the most forked PHP project on Github ever, the second most watched PHP project on Github, and has a large group of followers and contributors. Why is it so well loved? The answer is that you can add your own functionality to the CodeIgniter core without even breaking a sweat. You can extend the core controllers, models, and just about everything else with relative simplicity. Let's have a look at how we can do that.

**What We Can Extend**

CodeIgniter allows you to extend a certain group of core classes, namely:

**LISTING 1**

**Why Do We Want to Extend a Class?**

Before we continue, we need to understand why we would want to extend the core classes of CodeIgniter in the first place.

The reason is simple--we have a controller. Let's call it "Welcome Controller" because that is the default CodeIgniter controller. Welcome Controller is situated in a file call welcome.php in the application/controllers directory. This file is a standard CodeIgniter controller, and it is there by default when you install CodeIgniter for the first time. Welcome Controller is invoked by the CodeIgniter router when you hit the CodeIgniter landing page for the first time, and it does nothing but load the welcome\_message.php view file when you hit the landing page. It does this from the index function (see Figure 3, below) which is an action method.

Now let's assume that my client has asked me to do something really silly: he wants the IP recorded every time a visitor lands on any page in the entire CodeIgniter application. There are various ways that I can do this, depending on the client's wishes, but because we know that clients don't like to waste money and we don't like to waste time, we need to do this fast, and cheap.

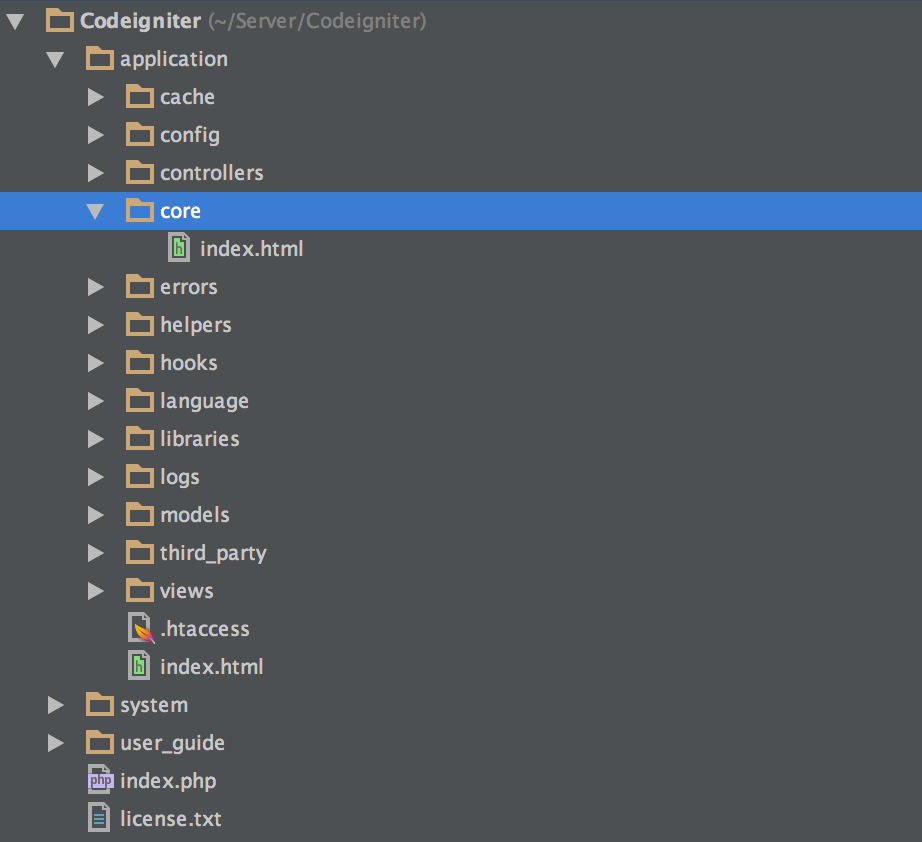
Human nature says all we do is create a model method that will record the IP of a user. It can record it in a database or in a file or it can email the client every time a new IP hits the site. The details of how it does this is out of the scope of this article and does not really matter. What matters, is that we want to do something every single time the page loads, no matter where we are in the application. The obvious thing to do is to chuck that model call into the constructor of every controller call, so that the IP of the visitor is recorded every time.

But what happens if our application has grown to ten, twenty or fifty controllers? Will you perhaps miss one controller? It's not hard to imagine that you could. Another point, much more important, is that you are using OOP here, for the love of Pete. Why on Earth would you want to go copy/pasting around everywhere like you learned to code last night?

Object Oriented Programming makes task this really easy for us. We have a controller (welcome.php), which extends a CodeIgniter core controller (CI\_controller.) Thus we have a SUPERCLASS, which really consists of these two classes. All we need to do here is pop a class in-between the two, making the superclass bigger. Sound simple? Well, it is. Let's check it out.

**An Example**

If you aren't aware of the CodeIgniter application structure, it looks like this:

  
**Figure 1**

As you can see in Figure 1, there is a directory called "core", which is where we can extend the previously mentioned core classes. My welcome controller exists in the application/controllers/welcome.php file (see Figure 2.) This is the default directory where all controllers are placed.

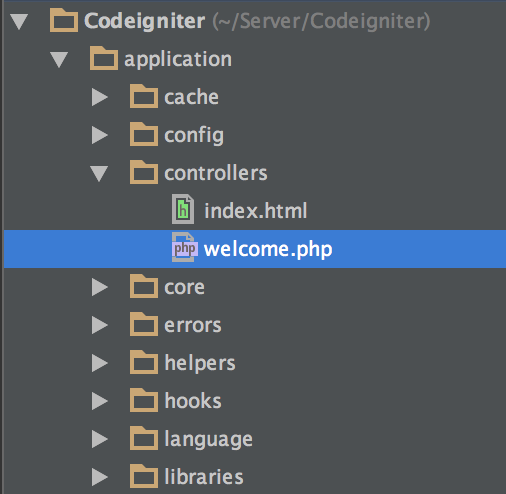
  
**Figure 2**

Figure 3 shows the contents of my welcome.php file. Pay particular attention to line 3. Here we can see that class Welcome (the Welcome controller) extends the CI\_Controller (the CodeIgniter Core Controller.)

  
**Figure 3**

This is where the fun begins. We need to create a new controller in the "application/core" directory. I create a file called my\_controller.php and add the following code to it.

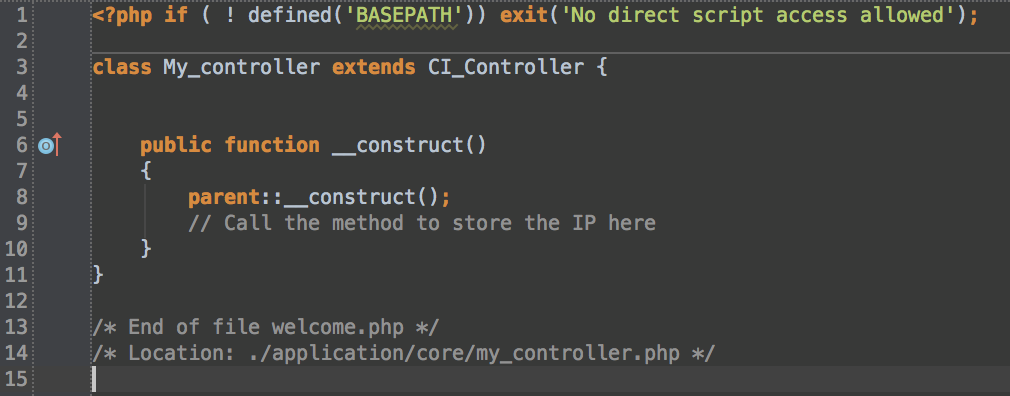
  
**Figure 4**

Figure 4 shows that I have created my controller which has a \_\_construct method. The first thing we need to remember is when extending a method of a class, we need to call the parent class. Thus, line 8 of my\_controller.php calls parent::\_\_construct(). After that, we can call our model method, which will save our IP, or do whatever we need to do. Now we need to adjust the Welcome controller to extend My\_controller (See Figure 5).

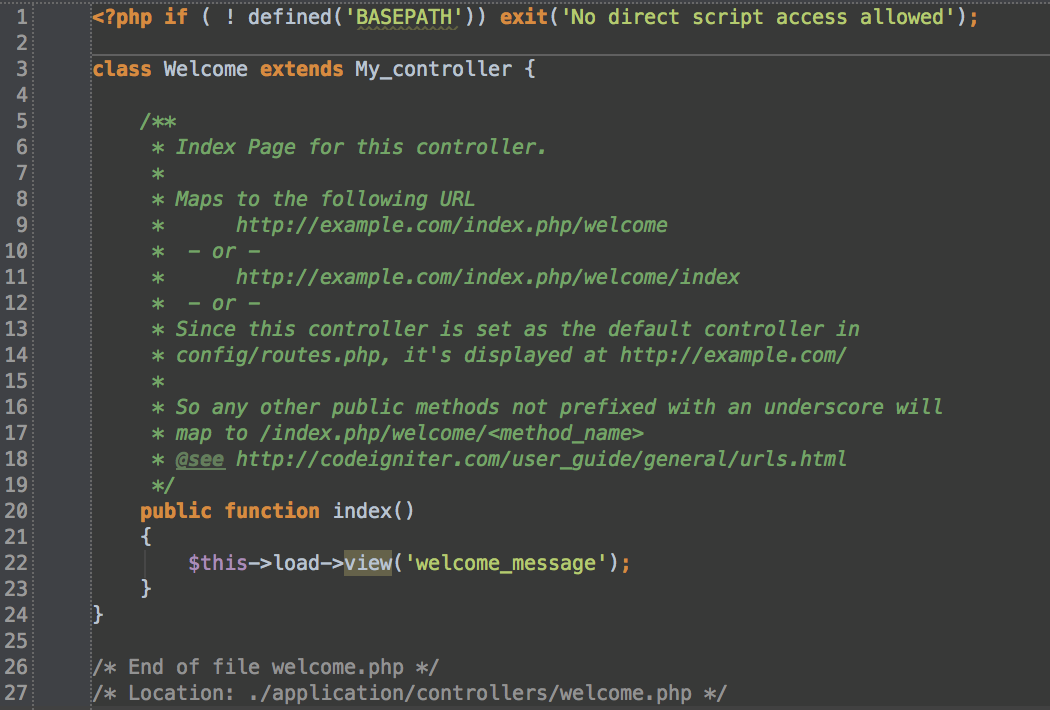
  
**Figure 5**

Figure 5 shows us that we have extended My\_controller, which in turn extends CI\_Controller. Everything flows through My\_Controller, and that's what we want. If you want to use this same method with the model, you can. You can create a model that is extended by every model you call, in exactly the same way. In that model you can do what? Anything you want to. Connect to the DB automatically every time you call a model. Build tables. Check if a user is logged in? It's your code and you can do anything you want, very simply. All you have to do now is make sure that every controller you create extends My\_controller, and not CI\_Controller.

**One Gotcha**

If you want to call your extended classes anything other than MY\_, you need to open the file application/config/config.php, and look around line 109. You'll see something similar to Figure 6.

  
**Figure 6**

All you need to do here is change the "MY\_" to anything you like. You'll then have to rename the file and the class accordingly, but there you have it. Some people chuck in a company name, or a library name. The choice is yours.

Remember, you can do the same with the complete list of library classes in LISTING 1, at the beginning of this article. Extending classes is a major part of Object Oriented Programming, and once you start doing it, you'll find that you can do anything you really need to do, quite simply. How to access config variable into the controller

$config['sess\_driver'] = 'files';

$config['sess\_cookie\_name'] = 'ci\_session';

$config['sess\_expiration'] = 7200;

$config['sess\_save\_path'] = NULL;

$config['sess\_match\_ip'] = FALSE;

$config['sess\_time\_to\_update'] = 300;

$config['sess\_regenerate\_destroy'] = FALSE; **echo "<br />Test - ".$this->config->item('sess\_expiration');**

**Use of hooks in codeigniter Set below line in the config.php $config['enable\_hooks'] = TRUE; code for in application->config->hooks.php <?php**

**defined('BASEPATH') OR exit('No direct script access allowed');**

**$hook['post\_controller\_constructor'][] = array(**

**'class' => ProfilerCustomHandler,**

**'function' => '** **EnableCustomProfiler ',**

**'filename' => 'anshu.php',**

**'filepath' => 'hooks',**

**'params' => array()**

**); ?> code for in application->hooks->anshu.php** <?php

defined('BASEPATH') OR exit('No direct script access allowed');

class ProfilerCustomHandler

{

function EnableCustomProfiler ()

{

$CI = & get\_instance();

/\* $CI->output->enable\_profiler( config\_item('enable\_hooks') ); \*/ $CI->output->enable\_profiler(TRUE);

}

# } ?> Web Page Caching

CodeIgniter lets you cache your pages in order to achieve maximum performance.

Although CodeIgniter is quite fast, the amount of dynamic information you display in your pages will correlate directly to the server resources, memory, and processing cycles utilized, which affect your page load speeds. By caching your pages, since they are saved in their fully rendered state, you can achieve performance that nears that of static web pages.

## How Does Caching Work?

Caching can be enabled on a per-page basis, and you can set the length of time that a page should remain cached before being refreshed. When a page is loaded for the first time, the cache file will be written to your application/cache folder. On subsequent page loads the cache file will be retrieved and sent to the requesting user’s browser. If it has expired, it will be deleted and refreshed before being sent to the browser.

## Enabling Caching

To enable caching, put the following tag in any of your controller methods:

$this->output->cache($n);

Where $n is the number of **minutes** you wish the page to remain cached between refreshes.

The above tag can go anywhere within a method. It is not affected by the order that it appears, so place it wherever it seems most logical to you. Once the tag is in place, your pages will begin being cached.

Important

Because of the way CodeIgniter stores content for output, caching will only work if you are generating display for your controller with a [view](https://www.codeigniter.com/user_guide/general/views.html).

Important

If you change configuration options that might affect your output, you have to manually delete your cache files.

Note

Before the cache files can be written you must set the file permissions on your application/cache/ directory such that it is writable.

## Deleting Caches

If you no longer wish to cache a file you can remove the caching tag and it will no longer be refreshed when it expires.

Note

Removing the tag will not delete the cache immediately. It will have to expire normally.

If you need to manually delete the cache, you can use the delete\_cache() method:

// Deletes cache for the currently requested URI

$this->output->delete\_cache();

// Deletes cache for /foo/bar

$this->output->delete\_cache('/foo/bar');

Download Files public function download\_client\_project\_document($fileName = NULL) {

$this->load->helper('download');

if ($fileName) {

//$file = realpath( "./assets/uploads/client-project-document/")."\\".$fileName;

$file = realpath("./assets/uploads/client-project-document/");

$filePath = $file . "/$fileName";

if (file\_exists($filePath)) {

$data = file\_get\_contents($filePath);

force\_download($fileName, $data);

} else {

redirect(base\_url());

}

}

} Upload Files $config['upload\_path'] = './assets/uploads/client-project-document/';

$config['allowed\_types'] = 'pdf|PDF|doc|DOC|docx|DOCX|xls|XLSX|gif|GIF|jpg|JPG|png|PNG|tiff|TIFF|zip|ZIP';

$this->load->library('upload', $config);

$fileName1 = NULL;

if($\_FILES['fileDocument1']['name']!="")

{

$this->upload->do\_upload('fileDocument1');

$data = array('upload\_data' => $this->upload->data());

$fileName1 = $data['uploame'];

} Send Emails $config = Array(

'protocol' => 'smtp',

'smtp\_host' => 'mail.utharaprint.co.uk',

'smtp\_port' => 26,

'smtp\_user' => 'anshu.porwal@utharaprint.co.uk',

'smtp\_pass' => 'anshu@123',

'mailtype' => 'html',

'charset' => 'iso-8859-1'

);

$obj = & get\_instance();

$obj->load->library('email', $config);

$obj->email->from($emailFrom, $fromName);

$obj->email->to($emailTo);

$obj->email->subject($subject);

$obj->email->message($message);

$obj->email->send();

**Jquery Ajax Example** <script type="text/javascript">

//jQuery(function()

$(document).ready(function()

{

$("#addQualification").click(function()

{

var teacherId = document.getElementById("teacherId").value;

var examPassed = document.getElementById("examPassed").value;

var examYear = document.getElementById("examYear").value;

var examCollage = document.getElementById("examCollage").value;

var examUniversity = document.getElementById("examUniversity").value;

var examMarksObtain = document.getElementById("examMarksObtain").value;

var examSubjects = document.getElementById("examSubjects").value;

var examAchivements = document.getElementById("examAchivements").value;

var dataString = 'teacherId1=' + teacherId + '&examPassed1=' + examPassed + '&examYear1=' + examYear + '&examCollage1=' + examCollage + '&examUniversity1=' + examUniversity + '&examMarksObtain1=' + examMarksObtain + '&examSubjects1=' + examSubjects+ '&examAchivements1=' + examAchivements;

//alert(dataString);

$.ajax({

type: "POST",

url: "<?php echo base\_url('admin/teacher/add\_qualification'); ?>",

data: dataString,

cache: false,

success: function(html)

{

alert(html);

window.location.href = "<?php echo base\_url('admin/teacher/update'); ?>/"+teacherId;

}

});

});

$("#addTeachSubjects").click(function()

{

var teacherId = document.getElementById("teacher\_Id").value;

var formdata = new FormData(document.getElementById("teacherTeachClassSubjects"));

$.ajax({

type: "POST",

url: "<?php echo base\_url('admin/teacher/add\_teach\_subjects'); ?>",

data: formdata,

processData: false,

contentType: false,

cache: false,

success: function(res)

{

alert(res);

window.location.href = "<?php echo base\_url('admin/teacher/update'); ?>/"+teacherId;

}

});

});

});

</script> **Calendar (Date Picker)** <script type="text/javascript">

$.noConflict();

jQuery(function()

{

jQuery( "#dob" ).datepicker({

changeMonth: true,

changeYear: true,

dateFormat: 'dd/mm/yy',

yearRange: "-100:-18",

});

jQuery( "#forSubmitDate" ).datepicker({

changeMonth: true,

changeYear: true,

dateFormat: 'dd/mm/yy',

});

jQuery( ".dateCalender" ).datepicker({

changeMonth: true,

changeYear: true,

dateFormat: 'dd/mm/yy',

});

}); **How to get total segments** $this->uri->total\_segments(); $this->uri->segment\_array();

**Error Levels**

**case E\_ERROR: // 1 return 'Fatal error';**

**case E\_WARNING: // 2 return 'Warning';**

**case E\_PARSE: // 4 return 'Parse error';**

**case E\_NOTICE: // 8 return 'Notice';**

**case E\_CORE\_ERROR: // 16 return 'Core fatal error';**

**case E\_CORE\_WARNING: // 32 return 'Core warning';**

**case E\_COMPILE\_ERROR: // 64 return 'Compile-time fatal error';**

**case E\_COMPILE\_WARNING: // 128 return 'Compile-time warning';**

**case E\_USER\_ERROR: // 256 return 'Fatal user-generated error';**

**case E\_USER\_WARNING: // 512 return 'User-generated warning';**

**case E\_USER\_NOTICE: // 1024 return 'User-generated notice';**

**case E\_STRICT: // 2048 return 'E\_STRICT';**

**case E\_RECOVERABLE\_ERROR: //4096 return 'Catchable fatal error';**

**case E\_DEPRECATED: // 8192 return 'E\_DEPRECATED';**

**case E\_USER\_DEPRECATED: // 16384 return 'E\_USER\_DEPRECATED';**