Everything You Need to Get Started With CodeIgniter

by Ben Haines 26 Jan 2009

Difficulty: Beginner Length: Long Languages: English

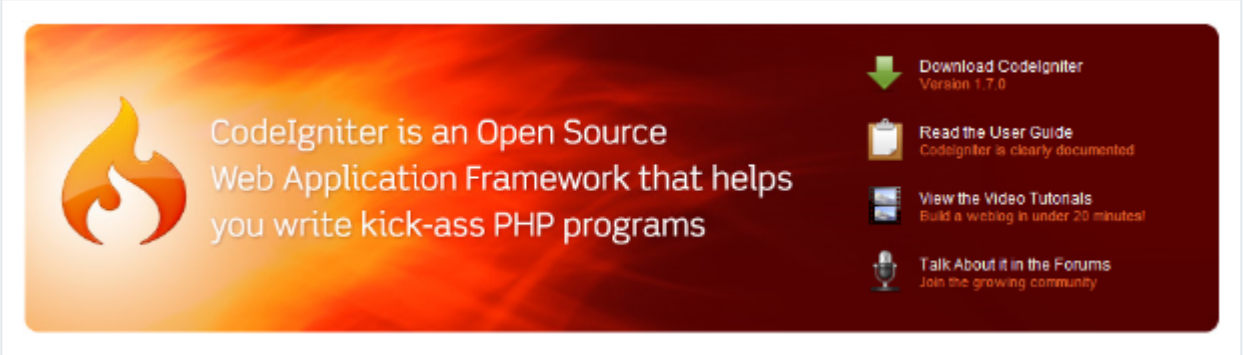
PHP Web Development CodeIgniter

CodeIgniter is a web application framework for PHP. It enables developers to build web applications faster, and it offers many helpful code libraries and helpers which speed up tedious tasks in PHP. CodeIgniter is based on a modular design; meaning that you can implement specific libraries at your discretion - which adds to the speed of the framework. This tutorial will attempt to show you the basics of setting up the framework, including how to build a basic hello world application that uses the MVC approach.

Why a Framework?

Frameworks allow for structure in developing applications by providing reusable classes and functions which can reduce development time significantly. Some downsides to frameworks are that they provide unwanted classes, adding code bloat which makes the app harder to navigate.

Why CodeIgniter?

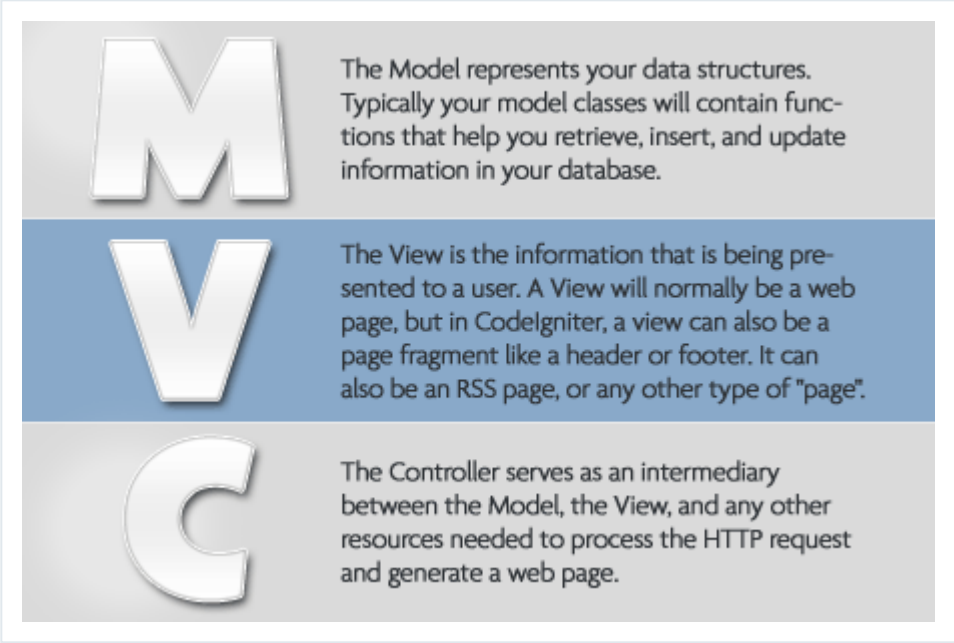


CodeIgniter is a very light, well performing framework. While, it is perfect for a beginner (because
of the small learning curve), it's also perfect for large and demanding web applications. CodeIgniter is developed by EllisLab and has thorough, easy to understand documentation. Below is a list of reasons of what makes CodeIgniter a smart framework to use?

- Small footprint with exceptional performance
- MVC approach to development (although it is very loosely based which allows for flexibility)
- Generates search engine friendly clean URLs
- Easily extensible

- Runs on both PHP 4 (4.3.2+) and 5
- Support for most major databases including MySQL (4.1+), MySQLi, MS SQL, Postgres, Oracle, SQLite,
  and ODBC.
- Application security is a focus
- Easy caching operations
- Many libraries and helpers to help you with complex operations such as email, image manipulation,
  form validation, file uploading, sessions, multilingual apps and creating apis for your app
- Most libraries are only loaded when needed which cuts back on resources needed

Why MVC?



For starters, MVC stands for Model, View, Controller. It is a programing pattern used in developing
web apps. This pattern isolates the user interface and backend (i.e. database interaction from each other. A successful
implementation of this lets developers modify their user interface or backend with out

affecting
the other. MVC also increases the flexibly of an app by being able to resuse models or views over
again). Below is a description of MVC.

- **Model:** The model deals with the raw data and database interaction and will contain functions
  like adding records to a database or selecting specific database records. In CI the model
  component is not required and can be included in the controller.
- **View:** The view deals with displaying the data and interface controls to the user with. In CI the view could be a web page, rss feed, ajax data or any other "page".
- **Controller:** The controller acts as the in between of view and model and, as the name suggests,
  it controls what is sent to the view from the model. In CI, the controller is also the place to load libraries and helpers.

An example of a MVC approach would be for a contact form.

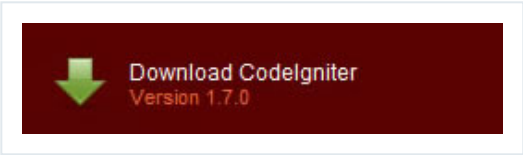
1. The user interacts with the view by filling in a form and submitting it.
2. The controller receives the POST data from the form, the controller sends this data to the model
   which updates in the database.
3. The model then sends the result of the database to the controller.
4. This result is updated in the view and displayed to the user.

This may sound like alot of work to do. But, trust me; when you're working with a large application, being able to reuse models or views saves a great deal of time.

Step 1: Downloading CodeIgniter

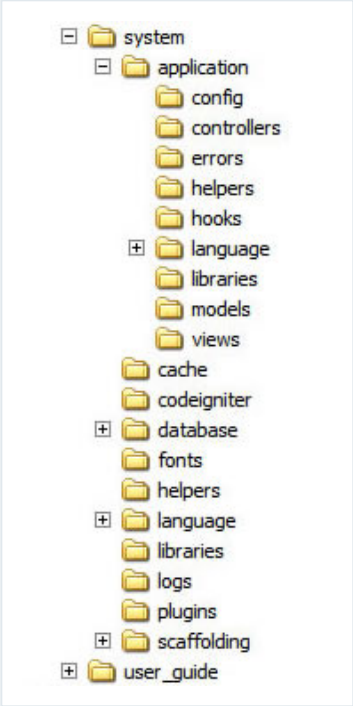
Too start off you will need to download CodeIgniter and upload it to your server. Point your browser

to http://www.codeigniter.com/ and
click the large download button. For this tutorial we are using version 1.70.



Step 2: Installing and Exploring CodeIgniter

Once you have downloaded CodeIgniter, all you need to do is unzip it, and rename the "CodeIgniter_1.7.0" folder to
either the application name or, in this case, "ci" and upload it to your PHP and MySQL enabled server.
Now that its on your server, I'll explain what all the folders and files are for:



- The **system** folder stores all the files which make CI work.
    - The **application** folder is almost identical to the contents of the **system** folder this is so the user can have files that are particular to that application, for example if a
      user only wanted to load a helper in one application he would place it in the

system/application/helpers folder
instead of the **system/helpers** folder.

- The **config** folder stores all the config files relevant to the application. Which
  includes information on what libaries the application should auto load and database details.
- The **controllers** folder stores all the controllers for the application.
- The **errors** folder stores all the template error pages for the application. When
  an error occurs the error page is generated from one of these templates.
- The **helpers** folder stores all the helpers which are specific to your application.
- The **hooks** folder is for hooks which modify the functioning of CI's core files,
  hooks should only be used by advanced users of CI
- The **language** folder stores lines of text which can be loaded through the language
  library to create multilingual sites.
- The **libraries** folder stores all the libraries which are specific to your application.
- The **models** folder stores all the models for the application.
- The **views** folder stores all the views for the application.

- The **cache** folder stores all the caches generated by the caching library.
- The **codeigniter** folder stores all the internals which make CI work.
- The **database** folder stores all the database drivers and class which enable you to
  connect to database.
- The **fonts** folder stores all the fonts which can be used by the image manipulation library.
- The **helpers** folder stores all of CI's core helpers but you can place your own helpers
  in here which can be accessed by all of your applications.
- The **language** folder stores all of CI's core language files which its libaries and helpers

use. You can also put your own language folders which can accessed by all of your applications.
    - The **libaries** folder stores all of CI's core libaries but you can place your own libraries
      in here which can be accessed by all of your applications
    - The **logs** folder stores all of the logs generated by CI.
    - The **plugin** folder stores all of the plugins which you can use. Plugins are almost identical
      to helpers, plugins are functions intended to be shared by the community.
    - The **scaffolding** folder stores all the files which make the scaffolding class work.
      Scaffolding provides a convenient CRUD like interface to access information in your database
      during development.
- The **user_guide** houses the user guide to CI.
- The **index.php** file is the bit that does all the CI magic it also lets the you change the name of the **system** and **application** folders.

Step 3: Configuring CodeIgniter

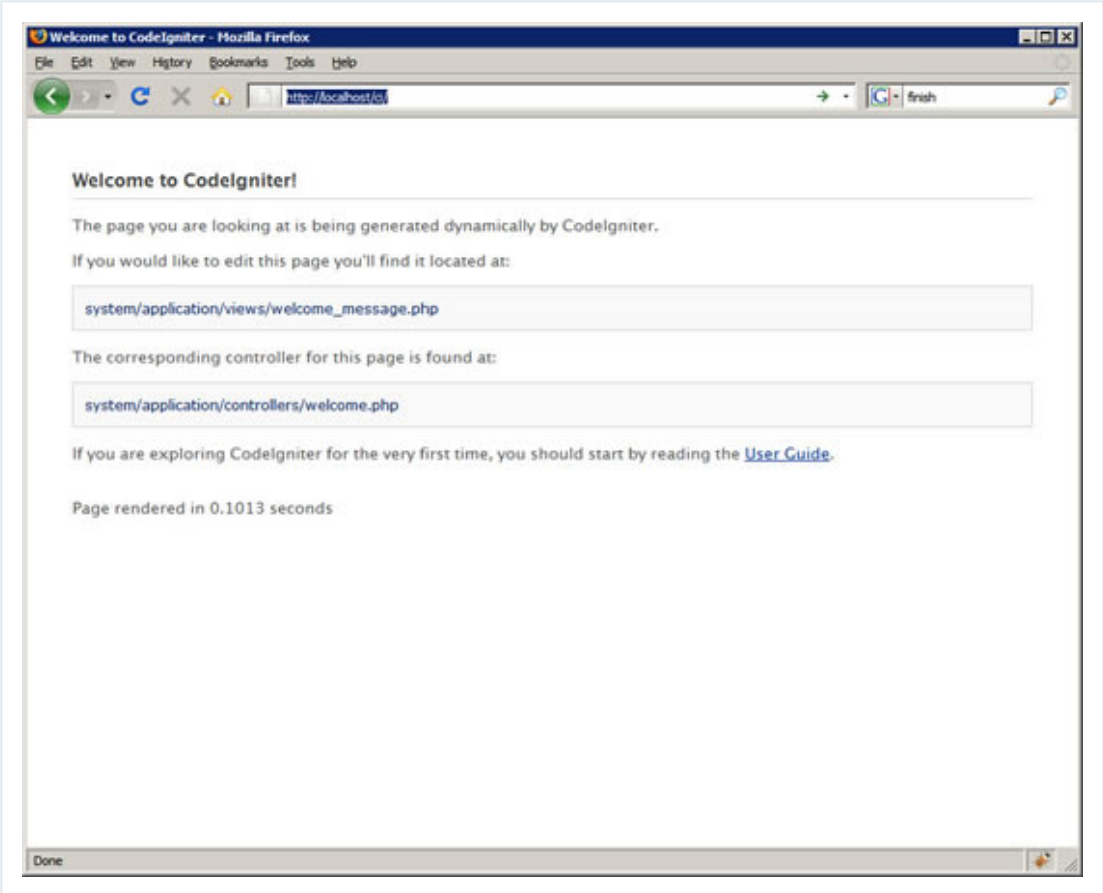
Getting CI up and running is rather simple. Most of it requires you to edit a few configuration
files.

You need to setup CI to point to the right base URL of the app. To do this, open up **system/application/config/config.php** and
edit the base_url array item to point to your server and CI folder.

```
1 $config['base_url'] = "http://localhost/ci/";
```

Step 4: Testing CodeIgniter

We'll do a quick test to see if CI is up and running properly. Go to http://localhost/ci/ and you
should see the following.



Step 5: Configuring CodeIgniter Cont.

If you're up and running, we should finish the configuration. We are starting to configure it specifically for our new helloworld app. If you want to use a database with your application, (which in this tutorial we do.) open up **system/application/config/database.php** and set the following array items to there corresponding values. This code connects to a MySQL database called "helloworld" on a localhost with the username "root", and the password, "root".

```
$db['default']['hostname'] = "localhost";
$db['default']['username'] = "root";
$db['default']['password'] = "root";
$db['default']['database'] = "helloworld";
$db['default']['dbdriver'] = "mysql";
```

Additionally, since we will be using the database quite a bit, we want it to auto load so that we don't have to specifically load it each time we connect. Open the **system/application/config/autoload.php** file and add 'database' to the autoload libaries array.

```
$autoload['libraries'] = array('database');
```

Currently, the CI setup will have a default controller called "welcome.php"; you can find this in the **system/application/controllers** folder. For this tutorial, delete it and open your **system/application/config/routes.php** file. Change the default array item to point to the "helloworld" controller.

```
$route['default_controller'] = "Helloworld"
```

CI also has a view file that we do not need. Open up the **system/application/view/** folder and delete the **welcome_message.php** file.

Step 6: Create the Helloworld Database

As this isn't really a tutorial on MySQL, I'll keep this section as short as possible. Create a database called "helloworld" and
run the following SQL through phpMyAdmin (or similar MySQL client).

```
CREATE TABLE `data` (
  `id` int(11) NOT NULL auto_increment,
  `title` varchar(255) NOT NULL,
  `text` text NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

INSERT INTO `data` (`id`, `title`, `text`) VALUES(1, 'Hello World!', 'Lorem ipsum dolor
sapien eros, lacinia eu, consectetur vel, dignissim et, massa. Praesent suscipit nunc v
nec libero. Phasellus lobortis, velit sed pharetra imperdiet, justo ipsum facilisis arc
Pellentesque molestie dui lacinia nulla. Sed vitae arcu at nisl sodales ultricies. Etia
vulputate in, augue. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices p
```

Step 7: Create the Helloworld Model

Models are optional in CI, but it's considered best practice to use them. They are just PHP classes that contain
functions which work with information from the database. Go
ahead and make a **helloworld_model.php** file
in the **system/application/models** folder.
In this file, create a Helloworld_model class, Helloworld_model construct and a function called getData.

In the getData function we are
going to use Active Record database functions which speed up database development times when working
with CI and databases. Essentially, they are simplified functions to create queries.

```
<?php
class Helloworld_model extends Model {

    function Helloworld_model()
    {
        // Call the Model constructor
        parent::Model();
    }
```

```
09
10      function getData()
11          {
12              //Query the data table for every record and row
13              $query = $this->db->get('data');
14
15              if ($query->num_rows() > 0)
16              {
17                  //show_error('Database is empty!');
18              }else{
19                  return $query->result();
20              }
21          }
22
23  }
24  ?>
```

Step 8: Create the Helloworld Controller

Let's create a controller that will display the view, and load the model. That way, when you go to the address http://localhost/ci/index.php/helloworld/, you will see the data from the database.
In the folder **system/application/controllers**, create a file called **helloworld.php**.
In this new file, we'll create a class which has the same name as the file.

Within this class, you need to create a function called "index". This is the function that will be
displayed when no other is provided - e.g. when http://localhost/ci/index.php/helloworld/ is
visited. If, for example, we created a function called foo, we could find this as http://localhost/ci/index.php/helloworld/foo/.

The key thing to remember is how CI structures its URLs; e.g http://host/codeignitordirectory/index.php/class/function.

In the controller index function, we need to load the model, query the database, and pass this queried
data to the view. To load any resources into CI e.g. libraries,
helpers, views, or, models, we use the load class. After we have loaded the model, we can access it through

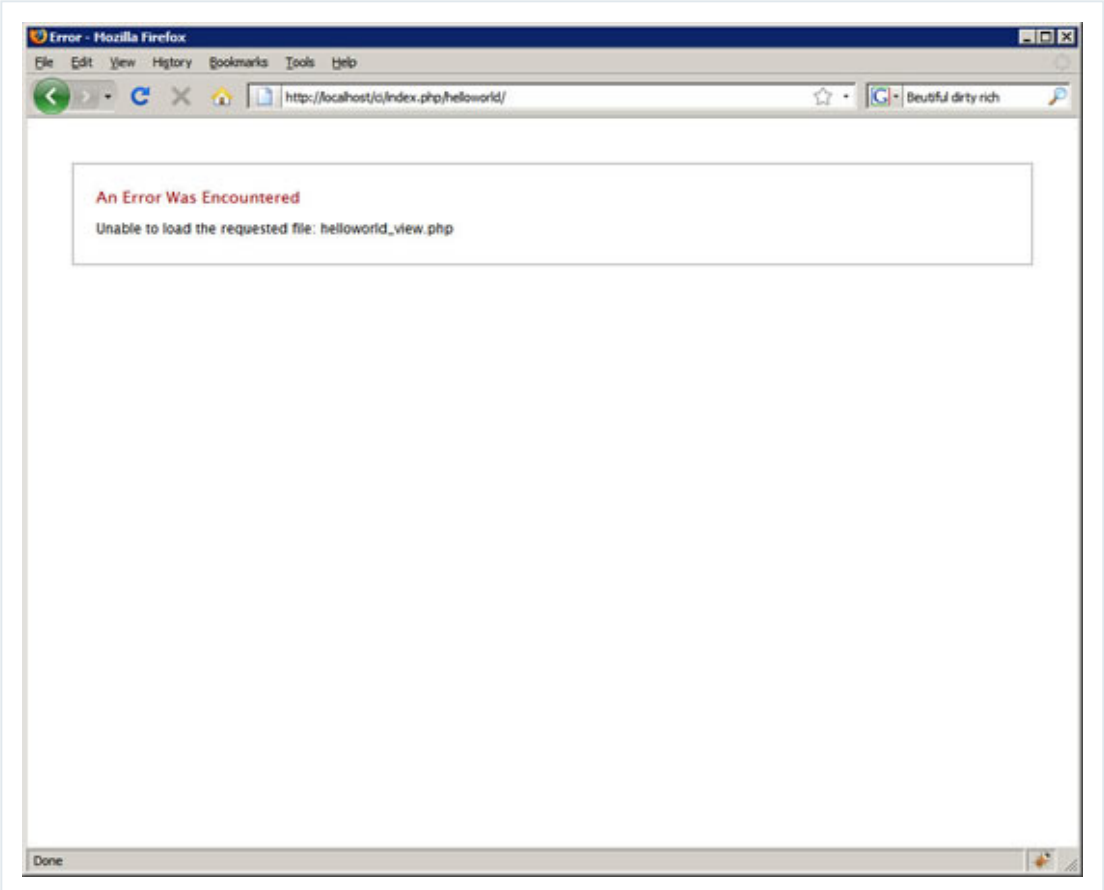
its model name and the particular function. To pass data to a view we need to assign it to an array
item and pass the array - which recreates the array items as a variable in the view file.

```
<?php
    class Helloworld extends Controller{
        function index()
        {
            $this->load->model('helloworld_model');

            $data['result'] = $this->helloworld_model-><span class="sql">getData</span>
            $data['page_title'] = "CI Hello World App!";

            $this->load->view('helloworld_view',$data);
        }
    }
?>
```

If we visited http://localhost/ci/index.php/helloworld/ now, it wouldn't work; this is because the view file does not exist.



Step 9: Create the Helloworld View

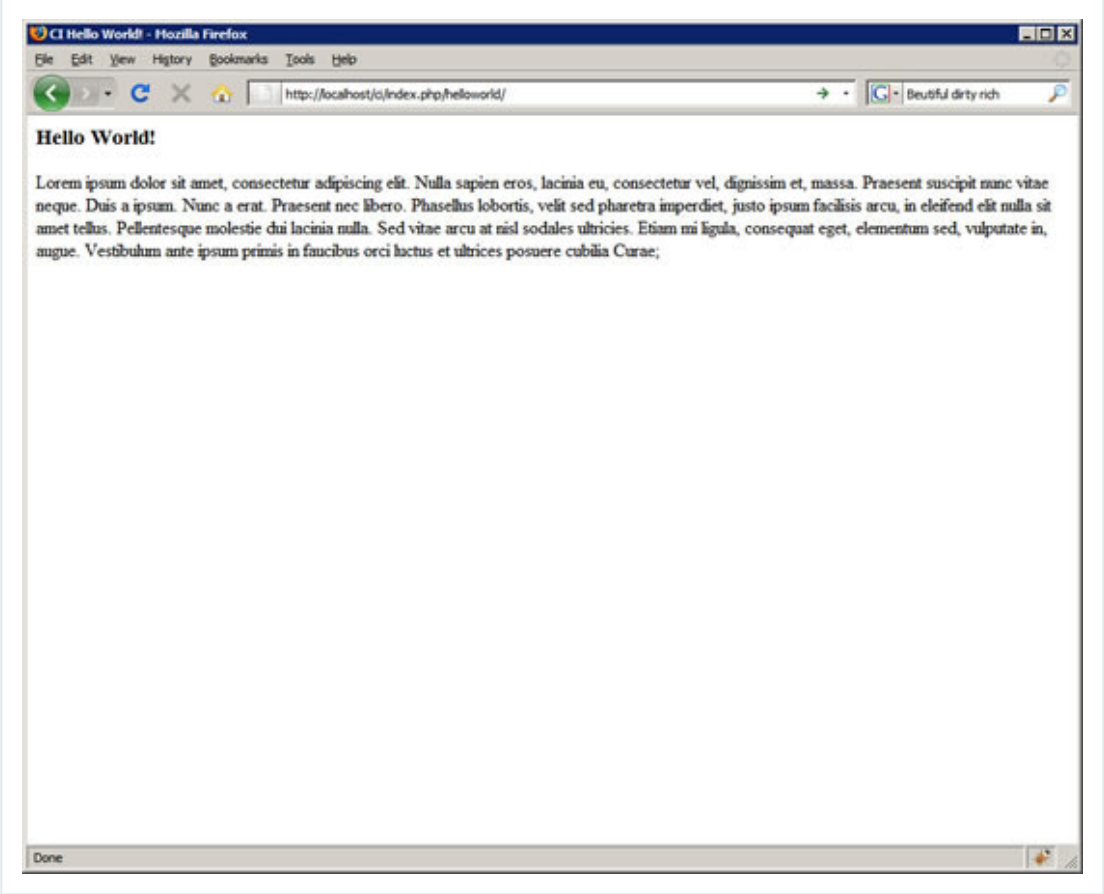
The view file is what the user sees and interacts with, it could be a segment of a page, or the whole page. You can pass an array of variables to the view through the second argument
of the load model function. To make the view for our tutorial, create a new file called **helloworld_view.php** in the **system/application/view** folder. Next, we just need to create our normal html, head
and body elements, and then a header and paragraph for the information from the database. To display all
the records received from the database, we put it in a "foreach" loop
which loops through all the elements.

```
<html>
    <head>
        <title><?=$page_title?></title>
    </head>
    <body>
        <?php foreach($result as $row):?>
        <h3><?=$row->title?></h3>
        <p><?=$row->text?></p>
        <br />
        <?php endforeach;?>
    </body>
</html>
```

You may have noticed that we are using php alternative syntax, this provides an convenient and time
saving way to write echo statements.

Step 10: Ta-da and Some Extras

When you visit "http://localhost/ci/index.php/helloworld/", you
should see something similar to this.



But we aren't done yet. There are a few things you can do to improve your CodeIgniter experience -
like removing that annoying index.php bit out of the URL. You can accomplish this task by creating a .htaccess file in the root folder, and adding the following code.

```
RewriteEngine on
RewriteCond $1 !^(index\.php|images|robots\.txt)
RewriteRule ^(.*)$ ci/index.php/$1 [L]
```

You'll also need to open up the **config.php** file in **system/application/config/** and edit the index_page array item to a blank string.

```
1 $config['index_page'] = "";
```

Another nifty trick is to turn on CI's ability to parse PHP alternative syntax if its not enabled by the server. To do this, open up the same file as before, **system/application/config/config.php**, and set the rewrite_short_tags array item to TRUE.

```
1 $config['rewrite_short_tags'] = TRUE;
```

I hope all goes well! Look forward to seeing more CodeIgniter tutorials from me in the future.

Subscribe to the NETTUTS RSS Feed for more daily web development tuts and articles.

Ben Haines

Hi, I'm Ben. I'm a developer/designer from a lil' country at the bottom of the world called New Zealand. My poisons of choice are PHP and CodeIgniter, JS and Mootools, CSS, and (X)HTML

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Email Address

Update me weekly

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by

Native