



CodeIgniter

tutorialspoint point
SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

CodeIgniter is a powerful PHP framework with a very small footprint, built for developers who need a simple and elegant toolkit to create full-featured web applications. CodeIgniter was created by EllisLab, and is now a project of the British Columbia Institute of Technology.

Audience

This tutorial has been prepared for developers who would like to learn the art of developing websites using CodeIgniter. It provides a complete understanding of this framework.

Prerequisites

Before you start proceeding with this tutorial, we assume that you are already exposed to HTML, Core PHP, and Advance PHP. We have used CodeIgniter version 3.0.1 in all the examples.

Copyright & Disclaimer

©Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial.....	i
Audience	i
Prerequisites	i
Copyright & Disclaimer.....	i
Table of Contents	ii
 1. CODEIGNITER – OVERVIEW	1
2. INSTALLING CODEIGNITER.....	3
3. APPLICATION ARCHITECTURE	4
Directory Structure.....	5
4. CODEIGNITER – MVC FRAMEWORK.....	8
5. CODEIGNITER – BASIC CONCEPTS.....	9
Controllers	9
Views	11
Models	13
Helpers.....	15
Routing	16
6. CODEIGNITER – CONFIGURATION	19
Configuring Base URL	19
Database Configuration.....	19
Autoload Configuration.....	21
7. WORKING WITH DATABASE.....	23
Connecting to a Database	23
Inserting a Record	23

Updating a Record	24
Deleting a Record	25
Selecting a Record	26
Closing a Connection	26
Example	26
8. CODEIGNITER – LIBRARIES.....	33
Library Classes	33
Creating Libraries	34
9. ERROR HANDLING	37
10. FILE UPLOADING.....	39
11. SENDING EMAIL.....	43
12. FORM VALIDATION.....	49
13. SESSION MANAGEMENT.....	55
14. FLASHDATA	58
15. TEMPDATA	61
16. COOKIE MANAGEMENT.....	65
17. COMMON FUNCTIONS.....	68
18. PAGE CACHING.....	71
19. PAGE REDIRECTION	73
20. APPLICATION PROFILING	75
21. BENCHMARKING	77
22. ADDING JS AND CSS.....	80

23. INTERNATIONALIZATION	83
24. CODEIGNITER – SECURITY	88
XSS Prevention.....	88
SQL Injection Prevention	88
Hiding PHP Errors	89
CSRF Prevention.....	90
Password Handling.....	90

1. CodeIgniter – Overview

CodeIgniter is an application development framework, which can be used to develop websites, using PHP. It is an Open Source framework. It has a very rich set of functionality, which will increase the speed of website development work.

If you know PHP well, then CodeIgniter will make your task easier. It has a very rich set of libraries and helpers. By using CodeIgniter, you will save a lot of time, if you are developing a website from scratch. Not only that, a website built in CodeIgniter is secure too, as it has the ability to prevent various attacks that take place through websites.

CodeIgniter Features

Some of the important features of CodeIgniter are listed below:

- Model-View-Controller Based System
- Extremely Light Weight
- Full Featured database classes with support for several platforms.
- Query Builder Database Support
- Form and Data Validation
- Security and XSS Filtering
- Session Management
- Email Sending Class. Supports Attachments, HTML/Text email, multiple protocols (sendmail, SMTP, and Mail) and more.
- Image Manipulation Library (cropping, resizing, rotating, etc.). Supports GD, ImageMagick, and NetPBM
- File Uploading Class
- FTP Class
- Localization
- Pagination
- Data Encryption
- Benchmarking
- Full Page Caching
- Error Logging
- Application Profiling
- Calendaring Class
- User Agent Class
- Zip Encoding Class

- Template Engine Class
- Trackback Class
- XML-RPC Library
- Unit Testing Class
- Search-engine Friendly URLs
- Flexible URI Routing
- Support for Hooks and Class Extensions
- Large library of “helper” functions

2. Installing CodeIgniter

It is very easy to install CodeIgniter. Just follow the steps given below:

- **Step-1:** Download the CodeIgniter from the link
<http://www.codeigniter.com/download>
- **Step-2:** Unzip the folder.
- **Step-3:** Upload all files and folders to your server.
- **Step-4:** After uploading all the files to your server, visit the URL of your server, e.g., www.domain-name.com.

On visiting the URL, you will see the following screen:

Welcome to CodeIgniter!

The page you are looking at is being generated dynamically by CodeIgniter.

If you would like to edit this page you'll find it located at:

`application/views/welcome_message.php`

The corresponding controller for this page is found at:

`application/controllers/Welcome.php`

If you are exploring CodeIgniter for the very first time, you should start by reading the [User Guide](#).

Page rendered in 0.1761 seconds. CodeIgniter Version 3.0.1

3. Application Architecture

The architecture of CodeIgniter application is shown below.

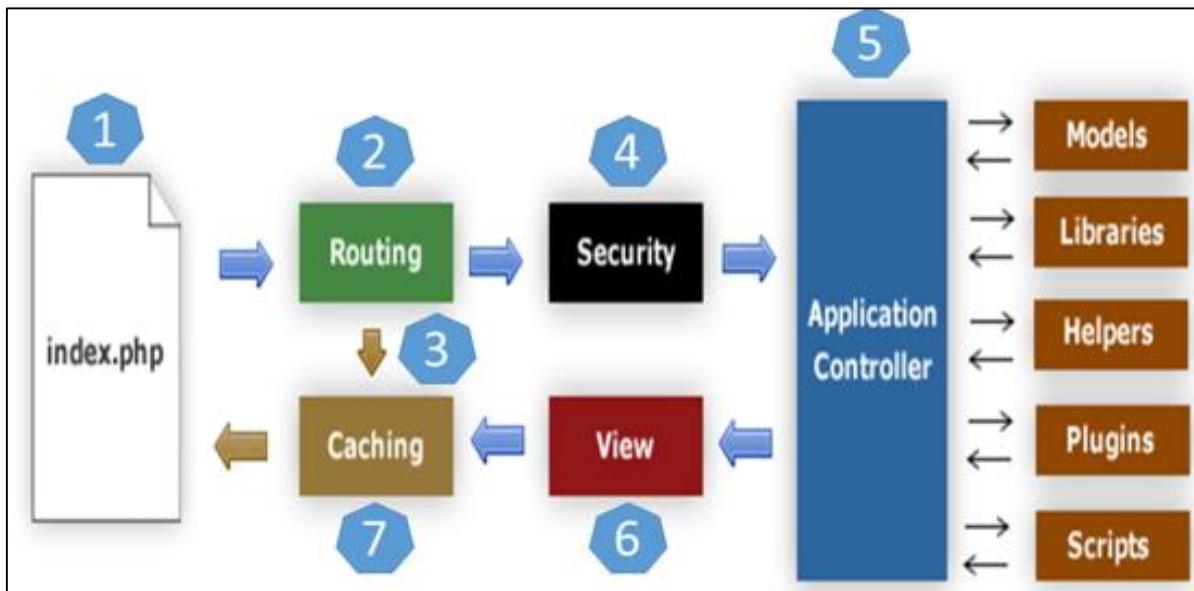


Figure: CodeIgniter Application Flowchart

- As shown in the figure, whenever a request comes to CodeIgniter, it will first go to **index.php** page.
- In the second step, **Routing** will decide whether to pass this request to step-3 for caching or to pass this request to step-4 for security check.
- If the requested page is already in **Caching**, then **Routing** will pass the request to step-3 and the response will go back to the user.
- If the requested page does not exist in **Caching**, then **Routing** will pass the requested page to step-4 for **Security** checks.
- Before passing the request to **Application Controller**, the **Security** of the submitted data is checked. After the **Security** check, the **Application Controller** loads necessary **Models**, **Libraries**, **Helpers**, **Plugins** and **Scripts** and pass it on to **View**.
- The **View** will render the page with available data and pass it on for **Caching**. As the requested page was not cached before so this time it will be cached in **Caching**, to process this page quickly for future requests.

Directory Structure

The image given below shows the directory structure of the CodeIgniter.

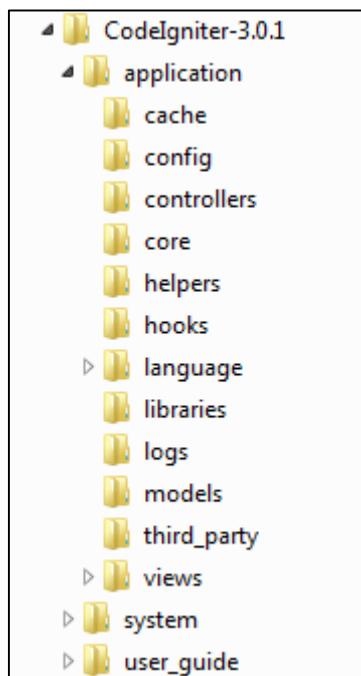


Figure: Directory Structure

CodeIgniter directory structure is divided into 3 folders:

- Application
- System
- User_guide

Application

As the name indicates the Application folder contains all the code of your application that you are building. This is the folder where you will develop your project. The Application folder contains several other folders, which are explained below:

- **Cache:** This folder contains all the cached pages of your application. These cached pages will increase the overall speed of accessing the pages.
- **Config:** This folder contains various files to configure the application. With the help of **config.php** file, user can configure the application. Using **database.php** file, user can configure the database of the application.
- **Controllers:** This folder holds the controllers of your application. It is the basic part of your application.
- **Core:** This folder will contain base class of your application.
- **Helpers:** In this folder, you can put helper class of your application.

- **Hooks:** The files in this folder provide a means to tap into and modify the inner workings of the framework without hacking the core files.
- **Language:** This folder contains language related files.
- **Libraries:** This folder contains files of the libraries developed for your application.
- **Logs:** This folder contains files related to the log of the system.
- **Models:** The database login will be placed in this folder.
- **Third_party:** In this folder, you can place any plugins, which will be used for your application.
- **Views:** Application's HTML files will be placed in this folder.

System

This folder contains CodeIgniter core codes, libraries, helpers and other files, which help make the coding easy. These libraries and helpers are loaded and used in web app development.

This folder contains all the CodeIgniter code of consequence, organized into various folders:

- **Core:** This folder contains CodeIgniter's core class. Do not modify anything here. All of your work will take place in the application folder. Even if your intent is to extend the CodeIgniter core, you have to do it with hooks, and hooks live in the application folder.
- **Database:** The database folder contains core database drivers and other database utilities.
- **Fonts:** The fonts folder contains font related information and utilities.
- **Helpers:** The helpers folder contains standard CodeIgniter helpers (such as date, cookie, and URL helpers).
- **Language:** The language folder contains language files. You can ignore it for now.
- **Libraries:** The libraries folder contains standard CodeIgniter libraries (to help you with e-mail, calendars, file uploads, and more). You can create your own libraries or extend (and even replace) standard ones, but those will be saved in the **application/libraries** directory to keep them separate from the standard CodeIgniter libraries saved in this particular folder.

User_guide

This is your user guide to CodeIgniter. It is basically, the offline version of user guide on CodeIgniter website. Using this, one can learn the functions of various libraries, helpers

and classes. It is recommended to go through this user guide before building your first web app in CodeIgniter.

Beside these three folders, there is one more important file named “**index.php**”. In this file, we can set the application environment and error level and we can define system and application folder name. It is recommended, not to edit these settings if you do not have enough knowledge about what you are going to do.

4. CodeIgniter – MVC Framework

CodeIgniter is based on the **Model-View-Controller (MVC) development pattern**. MVC is a software approach that separates application logic from presentation. In practice, it permits your web pages to contain minimal scripting since the presentation is separate from the PHP scripting.

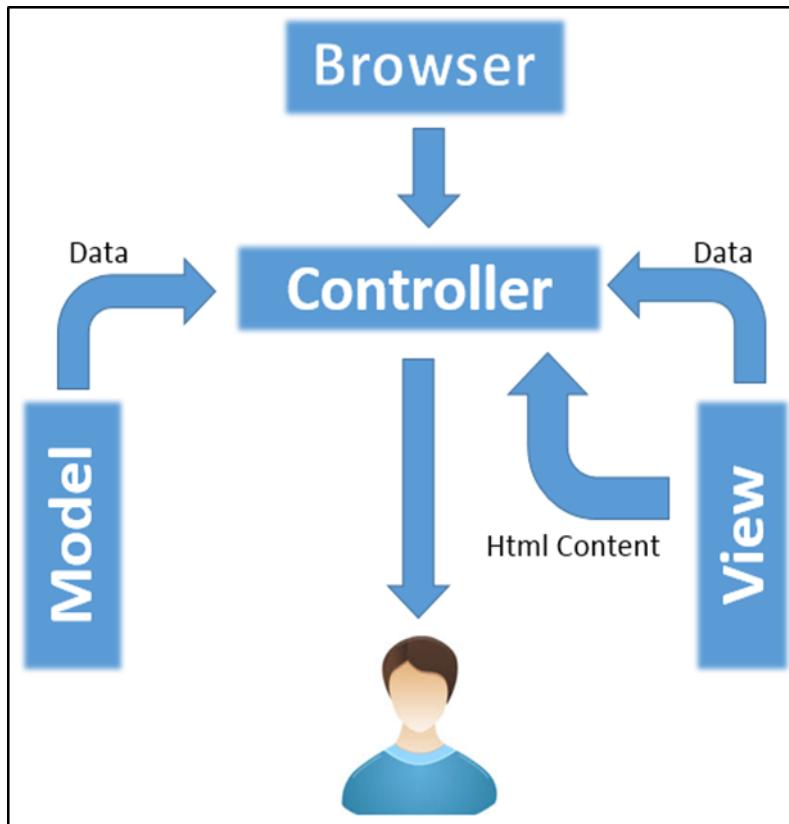


Figure: CodeIgniter – MVC Framework

- The **Model** represents your data structures. Typically, your model classes will contain functions that help you retrieve, insert and update information in your database.
- The **View** is information that is being presented to a user. A View will normally be a web page, but in CodeIgniter, a view can also be a page fragment like a header or footer. It can also be an RSS page, or any other type of "page".
- The **Controller** serves as an intermediary between the Model, the View, and any other resources needed to process the HTTP request and generate a web page.

5. CodeIgniter – Basic Concepts

Controllers

A controller is a simple class file. As the name suggests, it controls the whole application by URI.

Creating a Controller

First, go to **application/controllers** folder. You will find two files there, **index.html** and **Welcome.php**. These files come with the CodeIgniter.

Keep these files as they are. Create a new file under the same path named "**Test.php**". Write the following code in that file:

```
<?php  
class Test extends CI_Controller {  
  
    public function index()  
    {  
        echo "Hello World!";  
    }  
}  
?>
```

The **Test** class extends an in-built class called **CI_Controller**. This class must be extended whenever you want to make your own Controller class.

Calling a Controller

The above controller can be called by URI as follows:

```
http://www.your-domain.com/index.php/test
```

Notice the word "**test**" in the above URI after index.php. This indicates the class name of controller. As we have given the name of the controller "**Test**", we are writing "**test**" after the index.php. The class name must start with **uppercase letter** but we need to write **lowercase letter** when we call that controller by URI. The general syntax for calling the controller is as follows:

```
http://www.your-domain.com/index.php/controller/method-name
```

Creating & Calling Constructor Method

Let us modify the above class and create another method named "hello".

```
<?php
class Test extends CI_Controller {

    public function index()
    {
        echo "This is default function.";
    }

    public function hello()
    {
        echo "This is hello function.";
    }
}
?>
```

We can execute the above controller in the following three ways:

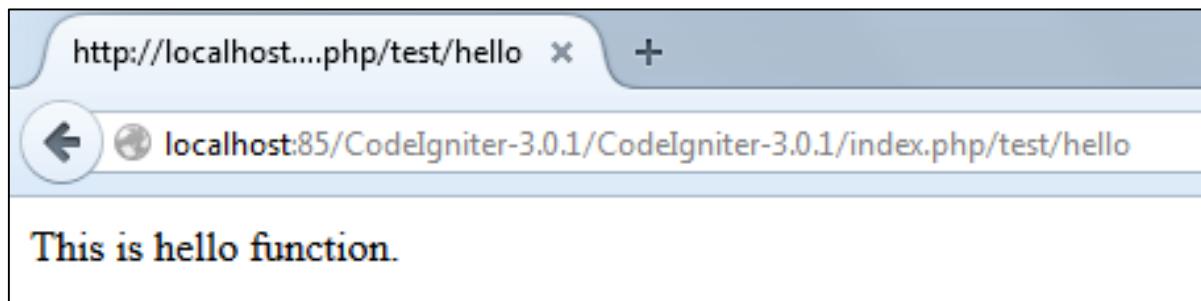
1. <http://www.your-domain.com/index.php/test>
2. <http://www.your-domain.com/index.php/test/index>
3. <http://www.your-domain.com/index.php/test/hello>

After visiting the first URI in the browser, we get the output as shown in the picture given below. As you can see, we got the output of the method "**index**", even though we did not pass the name of the method the URI. We have used only controller name in the URI. In such situations, the CodeIgniter calls the default method "**index**".



Visiting the second URI in the browser, we get the same output as shown in the above picture. Here, we have passed method's name after controller's name in the URI. As the name of the method is "**index**", we are getting the same output.

Visiting the third URI in the browser, we get the output as shown in picture given below. As you can see, we are getting the output of the method “**hello**” because we have passed “**hello**” as the method name, after the name of the controller “**test**” in the URI.



Points to Remember:

- The name of the controller class must start with an uppercase letter.
- The controller must be called with lowercase letter.
- Do not use the same name of the method as your parent class, as it will override parent class's functionality.

Views

This can be a simple or complex webpage, which can be called by the controller. The webpage may contain header, footer, sidebar etc. View cannot be called directly. Let us create a simple view. Create a new file under **application/views** with name “**test.php**” and copy the below given code in that file.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter View Example</title>
</head>
<body>
    CodeIgniter View Example
</body>
</html>
```

Change the code of **application/controllers/test.php** file as shown in the below.

Loading the View

The view can be loaded by the following syntax:

```
$this->load->view('name');
```

Where name is the view file, which is being rendered. If you have planned to store the view file in some directory then you can use the following syntax:

```
$this->load->view('directory-name/name');
```

It is not necessary to specify the extension as php, unless something other than .php is used.

The index() method is calling the view method and passing the "test" as argument to view() method because we have stored the html coding in "**test.php**" file under **application/views/test.php**.

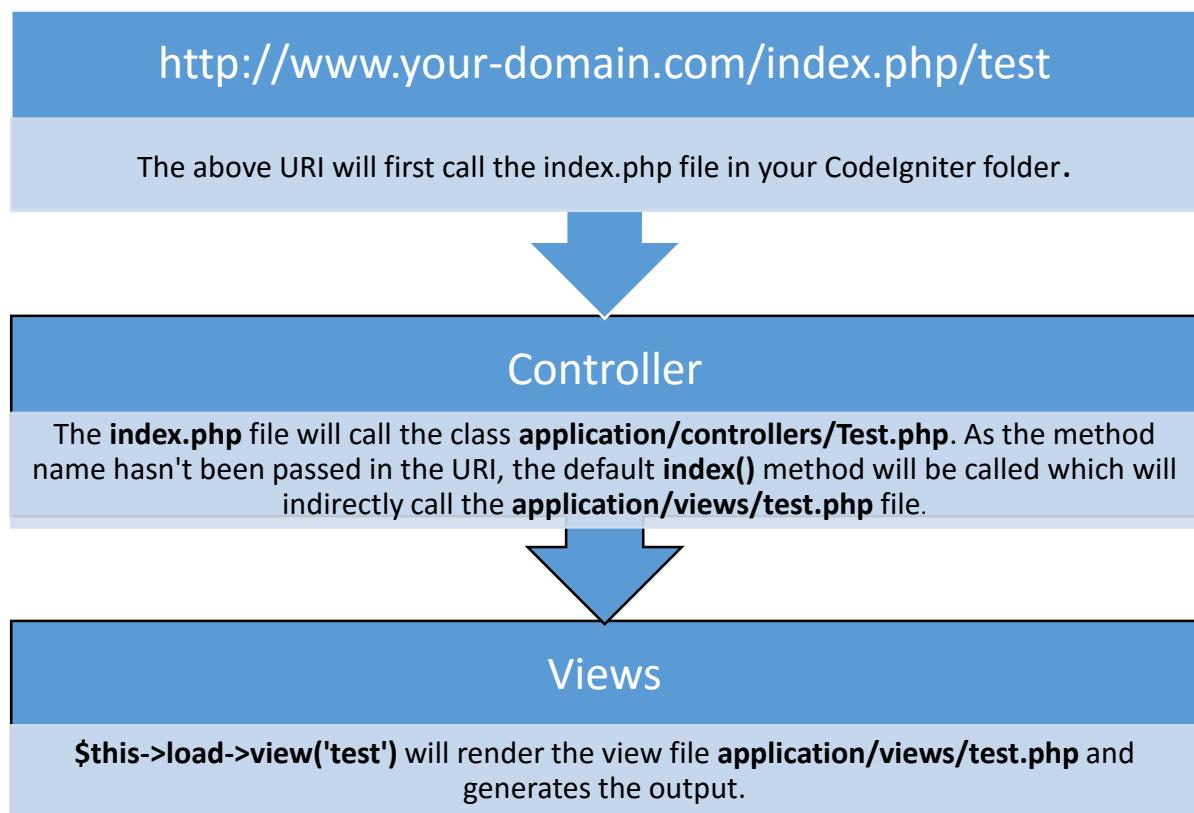
```
<?php
class Test extends CI_Controller {

    public function index()
    {
        $this->load->view('test');
    }
}
?>
```

Here is the output of the above code:



The following flowchart illustrates of how everything works:



Models

Models classes are designed to work with information in the database. As an example, if you are using CodeIgniter to manage users in your application then you must have model class, which contains functions to insert, delete, update and retrieve your users' data.

Creating Model Class

Model classes are stored in **application/models** directory. Following code shows how to create model class in CodeIgniter.

```

<?php
Class Model_name extends CI_Model{

    Public function __construct()
    {
        parent::__construct();
    }
?>
    
```

Where Model_name is the name of the model class that you want to give. Each model class must inherit the CodeIgniter's CI_Model class. The first letter of the model class must be in capital letter. Following is the code for users' model class.

```
<?php
Class zzzextends CI_Model{

Public function __construct()
{
    parent::__construct();
}

}

?>
```

The above model class must be saved as User_model.php. The class name and file name must be same.

Loading Model

Model can be called in controller. Following code can be used to load any model.

```
$this->load->model('model_name');
```

Where model_name is the name of the model to be loaded. After loading the model you can simply call its method as shown below.

```
$this->model_name->method();
```

Auto-loading Models

There may be situations where you want some model class throughout your application. In such situations, it is better if we autoload it.

```
/*
| -----
| Auto-load Models
| -----
| Prototype:
|
|     $autoload['model'] = array('first_model', 'second_model');
|
| You can also supply an alternative model name to be assigned
| in the controller:
|
|     $autoload['model'] = array('first_model' => 'first');
*/
$autoload['model'] = array();
```

As shown in the above figure, pass the name of the model in the array that you want to autoload and it will be autoloaded, while system is in initialization state and is accessible throughout the application.

Helpers

As the name suggests, it will help you build your system. It is divided into small functions to serve different functionality. A number of helpers are available in CodeIgniter, which are listed in the table below. We can build our own helpers too.

Helpers are typically stored in your **system/helpers**, or **application/helpers directory**. Custom helpers are stored in **application/helpers** directory and systems' helpers are stored in **system/helpers** directory. CodeIgniter will look first in your **application/helpers directory**. If the directory does not exist or the specified helper is not located, CodeIgniter will instead, look in your global **system/helpers/ directory**. Each helper, whether it is custom or system helper, must be loaded before using it.

Helper Name	Description
Array Helper	The Array Helper file contains functions that assist in working with arrays.
CAPTCHA Helper	The CAPTCHA Helper file contains functions that assist in creating CAPTCHA images.
Cookie Helper	The Cookie Helper file contains functions that assist in working with cookies.
Date Helper	The Date Helper file contains functions that help you work with dates.
Directory Helper	The Directory Helper file contains functions that assist in working with directories.
Download Helper	The Download Helper lets you download data to your desktop.
Email Helper	The Email Helper provides some assistive functions for working with Email. For a more robust email solution, see CodeIgniter's Email Class.
File Helper	The File Helper file contains functions that assist in working with files.
Form Helper	The Form Helper file contains functions that assist in working with forms.
HTML Helper	The HTML Helper file contains functions that assist in working with HTML.
Inflector Helper	The Inflector Helper file contains functions that permits you to change words to plural, singular, camel case, etc.
Language Helper	The Language Helper file contains functions that assist in working with language files.
Number Helper	The Number Helper file contains functions that help you work with numeric data.
Path Helper	The Path Helper file contains functions that permits you to work with file paths on the server.
Security Helper	The Security Helper file contains security related functions.

Smiley Helper	The Smiley Helper file contains functions that let you manage smileys (emoticons).
String Helper	The String Helper file contains functions that assist in working with strings.
Text Helper	The Text Helper file contains functions that assist in working with text.
Typography Helper	The Typography Helper file contains functions that help your format text in semantically relevant ways.
URL Helper	The URL Helper file contains functions that assist in working with URLs.
XML Helper	The XML Helper file contains functions that assist in working with XML data.

Loading a Helper

A helper can be loaded as shown below:

```
$this->load->helper('name');
```

Where name is the name of the helper. For example, if you want to load the URL Helper, then it can be loaded as:

```
$this->load->helper('url');
```

Routing

CodeIgniter has user-friendly URI routing system, so that you can easily re-route URL. Typically, there is a one-to-one relationship between a URL string and its corresponding controller class/method. The segments in a URI normally follow this pattern:

```
your-domain.com/class/method/id/
```

- The **first segment** represents the controller class that should be invoked.
- The **second segment** represents the class function, or method, that should be called.
- The **third**, and any additional segments, represent the ID and any variables that will be passed to the controller.

In some situations, you may want to change this default routing mechanism. CodeIgniter provides facility through which you can set your own routing rules.

Customize Routing Rules

There is a particular file where you can handle all these. The file is located at application/config/routes.php. You will find an array called \$route in which you can customize your routing rules. The key in the \$route array will decide what to route and the value will decide where to route. There are three reserved routes in CodeIgniter.

Reserved Routes	
<code>\$route['default_controller']</code>	This route indicates which controller class should be loaded, if the URI contains no data, which will be the case when people load your root URL. You are encouraged to have a default route otherwise a 404 page will appear, by default. We can set home page of website here so it will be loaded by default.
<code>\$route['404_override']</code>	This route indicates which controller class should be loaded if the requested controller is not found. It will override the default 404 error page. It won't affect to the <code>show_404()</code> function, which will continue loading the default <code>error_404.php</code> file in <code>application/views/errors/error_404.php</code> .
<code>\$route['translate_uri_dashes']</code>	As evident by the Boolean value, this is not exactly a route. This option enables you to automatically replace dashes ('-') with underscores in the controller and method URI segments, thus saving you additional route entries if you need to do that. This is required because the dash is not a valid class or method-name character and will cause a fatal error, if you try to use it.

Routes can be customized by **wildcards** or by using **regular expressions** but keep in mind that these customized rules for routing must come after the reserved rules.

Wildcards

We can use two wildcard characters as explained below:

- **(:num)** – It will match a segment containing only numbers.
- **(:any)** – It will match a segment containing any character.

Example

```
$route['product/:num']='catalog/product_lookup';
```

In the above example, if the literal word "product" is found in the first segment of the URL, and a number is found in the second segment, the "catalog" class and the "product_lookup" method are used instead.

Regular Expressions

Like wildcards, we can also use regular expressions in **\$route array key** part. If any URI matches with regular expression, then it will be routed to the value part set into \$route array.

Example

```
$route['products/([a-z]+)/(\d+)'] = '$1/id_$2';
```

In the above example, a URI similar to products/shoes/123 would instead call the “**shoes**” controller class and the “**id_123**” method.

6. CodeIgniter – Configuration

After setting up the site, the next thing that we should do is to configure the site. The application/config folder contains a group of files that set basic configuration of your site.

Configuring Base URL

The base URL of the site can be configured in application/config/config.php file. It is URL to your CodeIgniter root. Typically, this will be your base URL, with a trailing slash e.g.

```
http://example.com/
```

If this is not set, then CodeIgniter will try to guess the protocol, domain and path to your installation. However, you should always configure this explicitly and never rely on auto-guessing, especially in production environments. You can configure the base URL in the \$config array with key "base_url" as shown below:

```
$config['base_url'] = 'http://your-domain.com';
```

Database Configuration

The database of the site can be configured in application/database.php file. Often we need to set up database for different environment like development and production. With the multidimensional array provided in the CodeIgniter, we can setup database for different environment. The configuration settings are stored in the array as shown below:

```
$db['default'] = array(
    'dsn'      => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'database_name',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => TRUE,
    'db_debug' => TRUE,
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
```

```

'swap_pre' => '',
'encrypt' => FALSE,
'compress' => FALSE,
'stricton' => FALSE,
'failover' => array()
);

```

You can leave few options to their default values except hostname, username, password, database and dbdriver.

- **hostname:** Specify location of your database here e.g. localhost or IP address
- **username:** Set username of your database here.
- **password:** Set password of your database here.
- **database:** Set name of the database here.
- **dbdriver:** Set type of database that you are using e.g. MySQL, MySQLi, PostgreSQL, ODBC, and MS SQL.

By changing the key of the array **\$db**, you can set other configuration of database as shown below. Here, we have set the key to '**test**' to set the database for testing environment, by keeping the other database environment as it is.

```

$db['test'] = array(
    'dsn'      => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'database_name',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => TRUE,
    'db_debug' => TRUE,
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array()
);

```

```
);
```

You can simply switch to different environment by changing the value of a variable as shown below:

```
$active_group = 'default'; //This will set the default environment
$active_group = 'test'; //This will set the test environment
```

Autoload Configuration

This file specifies, by default, which systems should be loaded. In order to keep the framework as light-weight as possible, only the absolute minimal resources are loaded by default. One should autoload the frequently used system, rather than loading it at local level, repeatedly. Following are the things you can load automatically:

- **Libraries:** It is a list of libraries, which should be auto loaded. Provide a list of libraries in an array as shown below to be autoloaded by CodeIgniter. In this example, we are auto loading database, email and session libraries.

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

- **Drivers:** These classes are located in system/libraries/ or in your application/libraries/ directory, but are also placed inside their own subdirectory and they extend the CI_Driver_Library class. They offer multiple interchangeable driver options. Following is an example to autoload cache drivers.

```
$autoload['drivers'] = array('cache');
```

- **Helper files:** It is a list of helper files, to be autoloaded. Provide a list of libraries in the array, as shown below, to be autoloaded by CodeIgniter. In the given example, we are autoloading URL and file helpers.

```
$autoload['helper'] = array('url', 'file');
```

- **Custom config files:** These files are intended for use, only if you have created custom config files. Otherwise, leave it blank. Following is an example of how to autoload more than one config files.

```
$autoload['config'] = array('config1', 'config2');
```

- **Language files:** It is a list of language files, which should be auto loaded. Look at the example given below. Provide a list of languages in an array as shown below to be auto loaded by CodeIgniter. Keep in mind that do not include the "_lang" part of your file. For example, "codeigniter_lang.php" would be referenced as array('codeigniter');

```
$autoload['language'] = array('lang1', 'lang2');
```

- **Models:** It is a list of models file, which should be autoloaded. Provide a list of models in an array as shown below to be autoloaded by CodeIgniter. Following is the example of how to auto load more than one models files.

```
$autoload['model'] = array('first_model', 'second_model');
```

7. Working with Database

Like any other framework, we need to interact with the database very often and CodeIgniter makes this job easy for us. It provides rich set of functionalities to interact with database.

In this section, we will understand how the CRUD (Create, Read, Update, Delete) functions work with CodeIgniter. We will use **stud** table to select, update, delete, and insert the data in **stud** table.

Table Name: stud	
roll_no	int(11)
Name	varchar(30)

Connecting to a Database

We can connect to database in the following two way::

- **Automatic Connecting:** Automatic connection can be done by using the file application/config/autoload.php. Automatic connection will load the database for each and every page. We just need to add the database library as shown below:

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

- **Manual Connecting:** If you want database connectivity for only some of the pages, then we can go for manual connecting. We can connect to database manually by adding the following line in any class.

```
$this->load->database();
```

Here, we are not passing any argument because everything is set in the database config file application/config/database.php

Inserting a Record

To insert a record in the database, the insert() function is used as shown in the following table:

Syntax	insert([\$table = ''[, \$set = NULL[, \$escape = NULL]]])
Parameters	<ul style="list-style-type: none">• \$table (string) – Table name• \$set (array) – An associative array of field/value pairs• \$escape (bool) – Whether to escape values and identifiers

Returns	TRUE on success, FALSE on failure
Return Type	bool

The following example shows how to insert a record in **stud** table. The \$data is an array in which we have set the data and to insert this data to the table **stud**, we just need to pass this array to the insert function in the 2nd argument.

```
$data = array(
    'roll_no' => '1',
    'name' => 'Virat'
);
$this->db->insert("stud", $data);
```

Updating a Record

To update a record in the database, the **update()** function is used along with **set()** and **where()** functions as shown in the tables below. The **set()** function will set the data to be updated.

Syntax	set(\$key[, \$value = "[, \$escape = NULL]]])
Parameters	<ul style="list-style-type: none"> \$key (<i>mixed</i>) – Field name, or an array of field/value pairs \$value (<i>string</i>) – Field value, if \$key is a single field \$escape (<i>bool</i>) – Whether to escape values and identifiers
Returns	CI_DB_query_builder instance (method chaining)
Return Type	CI_DB_query_builder

The **where()** function will decide which record to update.

Syntax	where(\$key[, \$value = NULL[, \$escape = NULL]])
Parameters	<ul style="list-style-type: none"> \$key (<i>mixed</i>) – Name of field to compare, or associative array \$value (<i>mixed</i>) – If a single key, compared to this value \$escape (<i>bool</i>) – Whether to escape values and identifiers
Returns	DB_query_builder instance
Return Type	object

Finally, the **update()** function will update data in the database.

Syntax	update([<i>\$table</i> = "[, <i>\$set</i> = NULL[, <i>\$where</i> = NULL[, <i>\$limit</i> = NULL]]]]])
Parameters	<ul style="list-style-type: none"> • \$table (<i>string</i>) – Table name • \$set (<i>array</i>) – An associative array of field/value pairs • \$where (<i>string</i>) – The WHERE clause • \$limit (<i>int</i>) – The LIMIT clause
Returns	TRUE on success, FALSE on failure
Return Type	bool

```
$data = array(
    'roll_no' => '1',
    'name' => 'Virat'
);
$this->db->set($data);
$this->db->where("roll_no", '1');
$this->db->update("stud", $data);
```

Deleting a Record

To delete a record in the database, the **delete()** function is used as shown in the following table:

Syntax	delete([<i>\$table</i> = "[, <i>\$where</i> = "[, <i>\$limit</i> = NULL[, <i>\$reset_data</i> = TRUE]]]]])
Parameters	<ul style="list-style-type: none"> • \$table (<i>mixed</i>) – The table(s) to delete from; string or array • \$where (<i>string</i>) – The WHERE clause • \$limit (<i>int</i>) – The LIMIT clause • \$reset_data (<i>bool</i>) – TRUE to reset the query "write" clause
Returns	CI_DB_query_builder instance (method chaining) or FALSE on failure
Return Type	mixed

Use the following code to delete a record in the **stud** table. The first argument indicates the name of the table to delete record and the second argument decides which record to delete.

```
$this->db->delete("stud", "roll_no = 1");
```

Selecting a Record

To select a record in the database, the **get** function is used, as shown in the following table:

Syntax	<code>get([<i>\$table</i> = "[, <i>\$limit</i> = NULL[, <i>\$offset</i> = NULL]]])</code>
Parameters	<ul style="list-style-type: none"> \$table (string) – The table to query \$limit (int) – The LIMIT clause \$offset (int) – The OFFSET clause
Returns	<code>CI_DB_result</code> instance (method chaining)
Return Type	<code>CI_DB_result</code>

Use the following code to get all the records from the database. The first statement fetches all the records from "stud" table and returns the object, which will be stored in \$query object. The second statement calls the **result()** function with \$query object to get all the records as array.

```
$query = $this->db->get("stud");
$data['records'] = $query->result();
```

Closing a Connection

Database connection can be closed manually, by executing the following code:

```
$this->db->close();
```

Example

Create a controller class called **Stud_controller.php** and save it at **application/controller/Stud_controller.php**

Here is a complete example, wherein all of the above-mentioned operations are performed. Before executing the following example, create a database and table as instructed at the starting of this chapter and make necessary changes in the database config file stored at **application/config/database.php**

```
<?php
class Stud_controller extends CI_Controller {
```

```

function __construct(){
    parent::__construct();

    $this->load->helper('url');
    $this->load->database();
}

public function index()
{
    $query = $this->db->get("stud");
    $data['records'] = $query->result();
    $this->load->helper('url');
    $this->load->view('Stud_view',$data);
}

public function add_student_view(){
    $this->load->helper('form');
    $this->load->view('Stud_add');
}

public function add_student(){
    $this->load->model('Stud_Model');
    $data = array(
        'roll_no' => $this->input->post('roll_no'),
        'name' => $this->input->post('name')
    );
    $this->Stud_Model->insert($data);

    $query = $this->db->get("stud");
    $data['records'] = $query->result();
    $this->load->view('Stud_view',$data);
}

public function update_student_view(){
    $this->load->helper('form');
    $roll_no = $this->uri->segment('3');
    $query = $this->db->get_where("stud",array("roll_no"=>$roll_no));
}

```

```

        $data['records'] = $query->result();
        $data['old_roll_no'] = $roll_no;
        $this->load->view('Stud_edit',$data);
    }

public function update_student(){
    $this->load->model('Stud_Model');
    $data = array(
        'roll_no' => $this->input->post('roll_no'),
        'name' => $this->input->post('name')
    );
    $old_roll_no = $this->input->post('old_roll_no');
    $this->Stud_Model->update($data,$old_roll_no);
    $query = $this->db->get("stud");
    $data['records'] = $query->result();
    $this->load->view('Stud_view',$data);
}

public function delete_student(){
    $this->load->model('Stud_Model');
    $roll_no = $this->uri->segment('3');
    $this->Stud_Model->delete($roll_no);

    $query = $this->db->get("stud");
    $data['records'] = $query->result();
    $this->load->view('Stud_view',$data);
}

?>

```

Create a model class called **Stud_Model.php** and save it in **application/models/Stud_Model.php**

```

<?php
class Stud_Model extends CI_Model{

    function __construct(){

```

```

        parent::__construct();
    }

    public function insert($data){
        if ($this->db->insert("stud", $data)) {
            return true;
        }
    }

    public function delete($roll_no){
        if ($this->db->delete("stud", "roll_no = ".$roll_no)) {
            return true;
        }
    }

    public function update($data,$old_roll_no){
        $this->db->set($data);
        $this->db->where("roll_no", $old_roll_no);
        $this->db->update("stud", $data);
    }
}

?>
```

Create a view file called **Stud_add.php** and save it in **application/views/Stud_add.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Students Example</title>
</head>
<body>
<form method = "" action = "">

<?php
    echo form_open('Stud_controller/add_student');
```

```

echo form_label('Roll No.');
echo form_input(array('id'=>'roll_no', 'name'=>'roll_no'));
echo "<br/>";
echo form_label('Name');
echo form_input(array('id'=>'name', 'name'=>'name'));
echo "<br/>";
echo form_submit(array('id'=>'submit', 'value'=>'Add'));
echo form_close();
?>
</form>
</body>
</html>

```

Create a view file called **Stud_edit.php** and save it in **application/views/Stud_edit.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Students Example</title>
</head>
<body>
<form method = "" action = "">

<?php
    echo form_open('Stud_controller/update_student');
    echo form_hidden('old_roll_no',$old_roll_no);
    echo form_label('Roll No.');
    echo
form_input(array('id'=>'roll_no', 'name'=>'roll_no', 'value'=>$records[0]->roll_no));
    echo "<br/>";
    echo form_label('Name');
    echo
form_input(array('id'=>'name', 'name'=>'name', 'value'=>$records[0]->name));
    echo "<br/>";
    echo form_submit(array('id'=>'submit', 'value'=>'Edit'));

```

```

        echo form_close();

?>

</form>
</body>
</html>

```

Create a view file called **Stud_view.php** and save it in **application/views/Stud_view.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Students Example</title>
</head>
<body>
<a href="=base_url('index.php/stud/add_view')?&gt;"Add</a>      <table border="1">
<?php
    $i=1;
    echo "<tr>";
        echo "<td>Sr#</td>";
        echo "<td>Roll No.</td>";
        echo "<td>Name</td>";
        echo "<td>Edit</td>";
        echo "<td>Delete</td>";
    echo "<tr>";
    foreach($records as $r){
        echo "<tr>";
            echo "<td>".$i++. "</td>";
            echo "<td>".$r->roll_no."</td>";
            echo "<td>".$r->name."</td>";
            echo "<td><a href='".base_url()."index.php/stud/edit/".$r->roll_no."'>Edit</a></td>";
            echo "<td><a href='".base_url()."index.php/stud/delete/".$r->roll_no."'>Delete</a></td>";
        echo "<tr>";
    }
?>

```

```
</table>
</body>
</html>
```

Make the following change in the route file at **application/config/routes.php** and add the following line at the end of file.

```
$route['stud'] = "Stud_controller";
$route['stud/add'] = 'Stud_controller/add_student';
$route['stud/add_view'] = 'Stud_controller/add_student_view';
$route['stud/edit/(\d+)'] = 'Stud_controller/update_student_view/$1';
$route['stud/delete/(\d+)'] = 'Stud_controller/delete_student/$1';
```

Now, let us execute this example by visiting the following URL in the browser. Replace the yoursitename.com with your URL.

```
http://yoursite.com/index.php/stud
```

8. CodeIgniter – Libraries

The essential part of a CodeIgniter framework is its libraries. It provides a rich set of libraries, which indirectly increase the speed of developing an application. The system library is located at system/libraries. All we need to do is to load the library that we want to use. The library can be loaded as shown below:

```
$this->load->library('class name');
```

Where **class name** is the name of the library that we want to load. If we want to load multiple libraries, then we can simply pass an array as argument to **library()** function as shown below:

```
$this->load->library(array('email', 'table'));
```

Library Classes

The library classes are located in **system/libraries**. Each class has various functions to simplify the developing work. Following table shows the names of the library class and its description.

Library Class	Description
Benchmarking Class	Benchmarking class is always active, enabling the time difference between any two marked points to be calculated.
Caching Class	This class will cache the pages, to quickly access the page speed.
Calendaring Class	Using this class, you can dynamically create calendars.
Shopping Cart Class	Using this class, you can add or remove item from Shopping Cart. The items are saved in session and will remain active until the user is browsing the site.
Config Class	Configuration preferences can be retrieved, using this class. This class is initialized automatically.
Email Class	This class provides email related functionality, like send or reply to email.
Encryption Class	This class provides two-way data encryption functionality.
File Uploading Class	This class provides functionalities related to file uploading. You can set various preferences like type of file to be uploaded, size of the files etc.
Form Validation Class	This class provides various functions to validate form.
FTP Class	This class provides various FTP related functions like transferring files to remote server, moving, renaming or deleting files on server.
Image Manipulation Class	Manipulation of image like resize, thumbnail creation, cropping, rotating, watermarking can be done with the help of this class.
Input Class	This class pre-processes the input data for security reason.

Language Class	This class is used for internationalization.
Loader Class	This class loads elements like View files, Drivers, Helpers, Models etc.
Migrations Class	This class provides functionalities related to database migrations.
Output Class	This class sends the output to browser and also, caches that webpage.
Pagination Class	This class adds pagination functionalities to web page.
Template Parser Class	The Template Parser Class can perform simple text substitution for pseudo-variables contained within your view files. It can parse simple variables or variable tag pairs.
Security Class	This class contains security related functions like XSS Filtering, CSRF etc.
Session Library	This class provides functionalities to maintain session of your application.
HTML Table	This class is used to auto-generate HTML tables from array or database results.
Trackback Class	The Trackback Class provides functions that enable you to send and receive Trackback data.
Typography Class	The Typography Class provides methods that help to format text.
Unit Testing Class	This class provides functionalities to unit test your application and generate the result.
URI Class	The URI Class provides methods that help you retrieve information from your URI strings. If you use URI routing, you can also retrieve information about the re-routed segments.
User Agent Class	The User Agent Class provides functions that help identify information about the browser, mobile device, or robot visiting your site. In addition, you can get referrer information as well as language and supported character-set information.
XML-RPC and XML-RPC Server Classes	CodeIgniter's XML-RPC classes permit you to send requests to another server, or set up your own XML-RPC server to receive requests.
Zip Encoding Class	This class is used to create zip archives of your data.

Creating Libraries

CodeIgniter has rich set of libraries, which you can find in **system/libraries** folder but CodeIgniter is not just limited to system libraries, you can create your own libraries too, which can be stored in **application/libraries** folder. You can create libraries in three ways.

- Create new library
- Extend the native library
- Replace the native library

Create New Library

While creating new library one should keep in mind, the following things:

- The name of the file must start with a capital letter e.g. Mylibrary.php
- The class name must start with a capital letter e.g. class Mylibrary
- The name of the class and name of the file must match.

Mylibrary.php

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class Mylibrary {

    public function some_function()
    {
    }

}

/* End of file Mylibrary.php */
```

Loading the Custom Library

The above library can be loaded by simply executing the following line in your controller.

```
$this->load->library('mylibrary');
```

mylibrary is the name of your library and you can write it in lowercase as well as uppercase letters. Use the name of the library without ".php" extension. After loading the library, you can also call the function of that class as shown below.

```
$this->mylibrary->some_function();
```

Extend the Native Library

Sometimes, you may need to add your own functionality to the library provided by CodeIgniter. CodeIgniter provides facility by which you can extend the native library and add your own functions. To achieve this, you must extend the class of native library class. For example if you want to extend the Email library then it can be done as shown below:

```
Class MY_Email extends CI_Email {
}
```

Here, in the above example, MY_Email class is extending the native library's email class CI_Email. This library can be loaded by the standard way of loading email library. Save the above code in file My_Email.php

Replace the Native Library

In some situations, you do not want to use the native library the way it works and want to replace it with your own way. This can be done by replacing the native library. To achieve this, you just need to give the same class name as it is named in native library. For example, if you want to replace the **Email class**, then use the code as shown below. Save your file name with **Email.php** and give a class name to **CI_Email**.

Email.php

```
Class CI_Email {  
}
```

9. Error Handling

Many times, while using application, we come across errors. It is very annoying for the users if the errors are not handled properly. CodeIgniter provides an easy error handling mechanism.

You would like the messages to be displayed, when the application is in developing mode rather than in production mode as the error messages can be solved easily at the developing stage.

The environment of your application can be changed, by changing the line given below from **index.php** file. This can be set to anything but normally there are three values (development, test, production) used for this purpose.

```
define('ENVIRONMENT', isset($_SERVER['CI_ENV']) ? $_SERVER['CI_ENV'] :  
'development');
```

Different environment will require different levels of error reporting. By default, development mode will display errors and testing and live mode will hide them. CodeIgniter provides three functions as shown below to handle errors.

1. show_error() function displays errors in HTML format at the top of the screen.

Syntax	show_error(\$message, \$status_code, \$heading = 'An Error Was Encountered')
Parameters	<ul style="list-style-type: none">• \$message (mixed) – Error message• \$status_code (int) – HTTP Response status code• \$heading (string) – Error page heading
Return Type	mixed

2. show_404() function displays error if you are trying to access a page which does not exist.

Syntax	show_404(\$page = "", \$log_error = TRUE)
Parameters	<ul style="list-style-type: none">• \$page (string) – URI string• \$log_error (bool) – Whether to log the error
Return Type	void

3. `log_message()` function is used to write log messages. This is useful when you want to write custom messages.

Syntax	<code>log_message(\$level, \$message, \$php_error = FALSE)</code>
Parameters	<ul style="list-style-type: none"> • \$level (string) – Log level: 'error', 'debug' or 'info' • \$message (string) – Message to log • \$php_error (bool) – Whether we're logging a native PHP error message
Return Type	<code>void</code>

Logging can be enabled in **application/config/config.php** file. Given below is the screenshot of config.php file, where you can set threshold value.

```
/*
|-----|
| Error Logging Threshold
|-----|
|
| You can enable error logging by setting a threshold over zero. The
| threshold determines what gets logged. Threshold options are:
|
| 0 = Disables logging, Error logging TURNED OFF
| 1 = Error Messages (including PHP errors)
| 2 = Debug Messages
| 3 = Informational Messages
| 4 = All Messages
|
| You can also pass an array with threshold levels to show individual error types
|
| array(2) = Debug Messages, without Error Messages
|
| For a live site you'll usually only enable Errors (1) to be logged otherwise
| your log files will fill up very fast.
|
*/
$config['log_threshold'] = 0;
```

You can find the log messages in **application/log/**. Make sure that this directory is writable before you enable log files.

Various templates for error messages can be found in **application/views/errors/cli** or **application/views/errors/html**.

10. File Uploading

Using File Uploading class, we can upload files and we can also, restrict the type and size of the file to be uploaded. Follow the steps shown in the given example to understand the file uploading process in CodeIgniter.

Example

Copy the following code and store it at **application/view/Upload_form.php**.

```
<html>
<head>
<title>Upload Form</title>
</head>
<body>
<form method = "" action = "">

<?php echo $error;?>
<?php echo form_open_multipart('upload/do_upload');?>
<input type="file" name="userfile" size="20" />
<br /><br />
<input type="submit" value="upload" />
</form>
</body>
</html>
```

Copy the code given below and store it at **application/view/Upload_success.php**

```
<html>
<head>
<title>Upload Form</title>
</head>
<body>

<h3>Your file was successfully uploaded!</h3>

<ul>
<?phpforeach ($upload_data as $item => $value):?>
```

```

<li><?php echo $item;?>: <?php echo $value;?></li>
<?php endforeach; ?>
</ul>

<p><?php echo anchor('upload', 'Upload Another File!'); ?></p>

</body>
</html>

```

Copy the code given below and store it at **application/controllers/Upload.php**. Create “**uploads**” folder at the root of CodeIgniter i.e. at the parent directory of application folder.

```

<?php

class Upload extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        $this->load->helper(array('form', 'url'));
    }

    public function index()
    {
        $this->load->view('upload_form', array('error' => ' '));
    }

    public function do_upload()
    {
        $config['upload_path']          = './uploads/';
        $config['allowed_types']        = 'gif|jpg|png';
        $config['max_size']             = 100;
        $config['max_width']            = 1024;
        $config['max_height']           = 768;

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

```

```

if ( ! $this->upload->do_upload('userfile'))
{
    $error = array('error' => $this->upload->display_errors());
    $this->load->view('upload_form', $error);
}

else
{
    $data = array('upload_data' => $this->upload->data());
    $this->load->view('upload_success', $data);
}

}

?>

```

Make the following change in the route file in **application/config/routes.php** and add the following line at the end of file.

```
$route['upload'] = 'Upload';
```

Now let us execute this example by visiting the following URL in the browser. Replace the yoursitename.com with your URL.

```
http://yoursite.com/index.php/upload
```

It will produce the following screen:

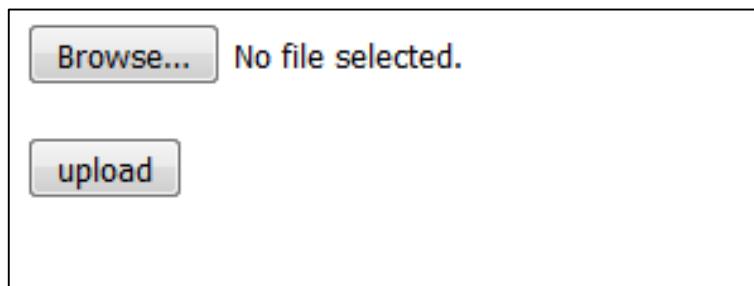


Figure: Upload Form

After successfully uploading a file, you will see the following screen:

Your file was successfully uploaded!

- file_name: vgsoft_logo.png
- file_type: image/png
- file_path: C:/wamp/www/CodeIgniter-3.0.1/CodeIgniter-3.0.1/uploads/
- full_path: C:/wamp/www/CodeIgniter-3.0.1/CodeIgniter-3.0.1/uploads/vgsoft_logo.png
- raw_name: vgsoft_logo
- orig_name: vgsoft_logo.png
- client_name: vgsoft_logo.png
- file_ext: .png
- file_size: 7.95
- is_image: 1
- image_width: 140
- image_height: 110
- image_type: png
- image_size_str: width="140" height="110"

[Upload Another File!](#)

Figure: File successfully uploaded

11. Sending Email

Sending email in CodeIgniter is much easier. You also configure the preferences regarding email in CodeIgniter. CodeIgniter provides following features for sending emails:

- Multiple Protocols: Mail, Sendmail, and SMTP
- TLS and SSL Encryption for SMTP
- Multiple recipients
- CC and BCCs
- HTML or Plaintext email
- Attachments
- Word wrapping
- Priorities
- BCC Batch Mode, enabling large email lists to be broken into small BCC batches.
- Email Debugging tools

Email class has the following functions to simplify the job of sending emails.

Syntax	from(\$from[, \$name = "[", \$return_path = NULL]])
Parameters	<ul style="list-style-type: none">• \$from (string) – “From” e-mail address• \$name (string) – “From” display name• \$return_path (string) – Optional email address to redirect undelivered e-mail to
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	reply_to(\$replyto[, \$name = ""])
Parameters	<ul style="list-style-type: none">• \$replyto (string) – E-mail address for replies• \$name (string) – Display name for the reply-to e-mail address
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	to(\$to)
Parameters	<ul style="list-style-type: none">• \$to (mixed) – Comma-delimited string or an array of e-mail addresses

Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	cc(\$cc)
Parameters	<ul style="list-style-type: none"> \$cc (mixed) – Comma-delimited string or an array of e-mail addresses
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	bcc(\$bcc[, \$limit = ""])
Parameters	<ul style="list-style-type: none"> \$bcc (mixed) – Comma-delimited string or an array of e-mail addresses \$limit (int) – Maximum number of e-mails to send per batch
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	subject(\$subject)
Parameters	<ul style="list-style-type: none"> \$subject (string) – E-mail subject line
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	message(\$body)
Parameters	<ul style="list-style-type: none"> \$body (string) – E-mail message body
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	set_alt_message(\$str)
Parameters	<ul style="list-style-type: none"> \$str (string) – Alternative e-mail message body
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	set_header(\$header, \$value)
Parameters	<ul style="list-style-type: none"> \$header (string) – Header name \$value (string) – Header value
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	clear([\$clear_attachments = FALSE])
Parameters	<ul style="list-style-type: none"> • \$clear_attachments (bool) – Whether or not to clear attachments
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	send([\$auto_clear = TRUE])
Parameters	<ul style="list-style-type: none"> • \$auto_clear (bool) – Whether to clear message data automatically
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	attach(\$filename[, \$disposition = "[, \$newname = NULL[, \$mime = "]]])
Parameters	<ul style="list-style-type: none"> • \$filename (string) – File name • \$disposition (string) – ‘disposition’ of the attachment. Most email clients make their own decision regardless of the MIME specification used here. https://www.iana.org/assignments/cont-disp/cont-disp.xhtml • \$newname (string) – Custom file name to use in the e-mail • \$mime (string) – MIME type to use (useful for buffered data)
Return	CI_Email instance (method chaining)
Return type	CI_Email

Syntax	attachment_cid(\$filename)
Parameters	<ul style="list-style-type: none"> • \$filename (string) – Existing attachment filename
Return	Attachment Content-ID or FALSE if not found
Return type	string

Sending an Email

To send an email using CodeIgniter, first you have to load email library using the following:

```
$this->load->library('email');
```

After loading the library, simply execute the following functions to set necessary elements to send an email. The **from()** function is used to set- from where the email is being sent and **to()** function is used - to whom the email is being sent. The **subject()** and **message()** function is used to set the subject and message of the email.

```
$this->email->from('your@example.com', 'Your Name');
$this->email->to('someone@example.com');

$this->email->subject('Email Test');
$this->email->message('Testing the email class.');
```

After that, execute the **send()** function as shown below to send an email.

```
$this->email->send();
```

Example

Create a controller file **Email_controller.php** and save it in **application/controller/Email_controller.php**.

```
<?php

class Email_controller extends CI_Controller {

    function __construct()
    {
        parent::__construct();
        $this->load->library('session');
        $this->load->helper('form');
    }

    public function index()
    {
        $this->load->helper('form');
        $this->load->view('email_form');
    }

    public function send_mail()
    {
        $from_email = "your@example.com";
        $to_email = $this->input->post('email');

        //Load email library
        $this->load->library('email');

        $this->email->from($from_email, 'Your Name');
        $this->email->to($to_email);
    }
}
```

```

        $this->email->subject('Email Test');
        $this->email->message('Testing the email class.');

        //Send mail
        if($this->email->send())
            $this->session->set_flashdata("email_sent","Email sent
successfully.");
        else
            $this->session->set_flashdata("email_sent","Error in sending
Email.");
        $this->load->view('email_form');
    }
}
?>

```

Create a view file called **email_form.php** and save it at **application/views/email_form.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter Email Example</title>
</head>
<body>
<?php
    echo $this->session->flashdata('email_sent');
    echo form_open('/Email_controller/send_mail');
?>
    <input type="email" name="email" required />
    <input type="submit" value="SEND MAIL">
<?php
    echo form_close();
?>
</body>
</html>

```

Make the changes in the **routes.php** file in **application/config/routes.php** and add the following line at the end of the file.

```
$route['email'] = 'Email_Controller';
```

Execute the above example by visiting the following link. Replace the `yoursite.com` with the URL of your site.

```
http://yoursite.com/index.php/email
```

12. Form Validation

Validation is an important process while building web application. It ensures that the data that we are getting is proper and valid to store or process. CodeIgniter has made this task very easy. Let us understand this process with a simple example.

Example

Create a view file **myform.php** and save the below code it in **application/views/myform.php**. This page will display form where user can submit his name and we will validate this page to ensure that it should not be empty while submitting.

```
<html>
<head>
<title>My Form</title>
</head>
<body>
<form method = "" action = "">

<?php echo validation_errors(); ?>

<?php echo form_open('form'); ?>

<h5>Name</h5>
<input type="text" name="name" value="" size="50" />

<div><input type="submit" value="Submit" /></div>

</form>

</body>
</html>
```

Create a view file **formsuccess.php** and save it in **application/views/formsuccess.php**. This page will be displayed if the form is validated successfully.

```
<html>
<head>
```

```

<title>My Form</title>
</head>
<body>

<h3>Your form was successfully submitted!</h3>

<p><?php echo anchor('form', 'Try it again!'); ?></p>

</body>
</html>

```

Create a controller file **Form.php** and save it in **application/controller/Form.php**. This form will either, show errors if it is not validated properly or redirected to **formsuccess.php** page.

```

<?php

class Form extends CI_Controller {

    public function index()
    {
        /* Load form helper */
        $this->load->helper(array('form'));

        /* Load form validation library */
        $this->load->library('form_validation');

        /* Set validation rule for name field in the form */
        $this->form_validation->set_rules('name', 'Name', 'required');

        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('myform');
        }
        else
        {
            $this->load->view('formsuccess');
        }
    }
}

```

```

    }
}

```

Add the following line in **application/config/routes.php**.

```
$route['validation'] = 'Form';
```

Let us execute this example by visiting the following URL in the browser. This URL may be different based on your site.

```
http://yoursite.com/index.php/validation
```

It will produce the following screen:

The form consists of a single text input field labeled "Name" and a "Submit" button below it. The entire form is contained within a rectangular border.

Figure: Validation Form

We have added a validation in the controller – **Name** is required field before submitting the form. So, if you click the submit button without entering anything in the name field, then you will be asked to enter the name before submitting as shown in the screen below.

The page displays an error message "The Name field is required." above the original validation form. The form structure remains the same: a "Name" input field and a "Submit" button.

Figure: Form Not Validated Successfully

After entering the name successfully, you will be redirected to the screen as shown below.

The page displays a success message "Your form was successfully submitted!" in large, bold, blue text. Below it is a link "Try it again!" in purple text.

Figure: Form Not Validated Successfully

In the above example, we have used the required rule setting. There are many rules available in the CodeIgniter, which are described below.

Validation Rule Reference

The following is a list of all the native rules that are available to use:

Rule	Parameter	Description	Example
required	No	Returns FALSE if the form element is empty.	
matches	Yes	Returns FALSE if the form element does not match the one in the parameter.	matches[form_item]
regex_match	Yes	Returns FALSE if the form element does not match the regular expression.	regex_match[/regex/]
differs	Yes	Returns FALSE if the form element does not differ from the one in the parameter.	differs[form_item]
is_unique	Yes	Returns FALSE if the form element is not unique to the table and field name in the parameter. Note: This rule requires Query Builder to be enabled in order to work.	is_unique[table.field]
min_length	Yes	Returns FALSE if the form element is shorter than the parameter value.	min_length[3]
max_length	Yes	Returns FALSE if the form element is longer than the parameter value.	max_length[12]
exact_length	Yes	Returns FALSE if the form element is not exactly the parameter value.	exact_length[8]
greater_than	Yes	Returns FALSE if the form element is less than or equal to the parameter value or not numeric.	greater_than[8]
greater_than_equal_to	Yes	Returns FALSE if the form element is less than the parameter value, or not numeric.	greater_than_equal_to[8]

less_than	Yes	Returns FALSE if the form element is greater than or equal to the parameter value or not numeric.	less_than[8]
less_than_equal_to	Yes	Returns FALSE if the form element is greater than the parameter value, or not numeric.	less_than_equal_to[8]
in_list	Yes	Returns FALSE if the form element is not within a predetermined list.	in_list[red,blue,green]
alpha	No	Returns FALSE if the form element contains anything other than alphabetical characters.	
alpha_numeric	No	Returns FALSE if the form element contains anything other than alpha-numeric characters.	
alpha_numeric_spaces	No	Returns FALSE if the form element contains anything other than alpha-numeric characters or spaces. Should be used after trim to avoid spaces at the beginning or end.	
alpha_dash	No	Returns FALSE if the form element contains anything other than alpha-numeric characters, underscores or dashes.	
numeric	No	Returns FALSE if the form element contains anything other than numeric characters.	
integer	No	Returns FALSE if the form element contains anything other than an integer.	
decimal	No	Returns FALSE if the form element contains anything other than a decimal number.	

is_natural	No	Returns FALSE if the form element contains anything other than a natural number: 0, 1, 2, 3, etc.	
is_natural_no_zero	No	Returns FALSE if the form element contains anything other than a natural number, but not zero: 1, 2, 3, etc.	
valid_url	No	Returns FALSE if the form element does not contain a valid URL.	
valid_email	No	Returns FALSE if the form element does not contain a valid email address.	
valid_emails	No	Returns FALSE if any value provided in a comma-separated list is not a valid email.	
valid_ip	No	Returns FALSE if the supplied IP is not valid. Accepts an optional parameter of 'ipv4' or 'ipv6' to specify an IP format.	
valid_base64	No	Returns FALSE if the supplied string contains anything other than valid Base64 characters.	

13. Session Management

When building websites, we often need to track user's activity and state and for this purpose, we have to use **session**. CodeIgniter has session class for this purpose.

Initializing a Session

Sessions data are available globally through the site but to use those data we first need to initialize the session. We can do that by executing the following line in constructor.

```
$this->load->library('session');
```

After loading the session library, you can simply use the session object as shown below.

```
$this->session
```

Add Session Data

In PHP, we simply use **\$_SESSION** array to set any data in session as shown below.

```
$_SESSION['key'] = value;
```

Where '**key**' is the key of array and **value** is assigned on right side of equal to sign.

The same thing can be done in CodeIgniter as shown below.

```
$this->session->set_userdata('some_name', 'some_value');
```

set_userdata() function takes two arguments. The first argument, **some_name**, is the name of the session variable, under which, **some_value** will be stored.

set_userdata() function also supports another syntax in which you can pass array to store values as shown below.

```
$newdata = array(
    'username' => 'johndoe',
    'email'     => 'johndoe@some-site.com',
    'logged_in' => TRUE
);

$this->session->set_userdata($newdata);
```

Remove Session Data

In PHP, we can remove data stored in session using the **unset()** function as shown below.

```
unset($_SESSION['some_name']);
```

Removing session data in CodeIgniter is very simple as shown below. The below version of **unset_userdata()** function will remove only one variable from session.

```
$this->session->unset_userdata('some_name');
```

If you want to remove more values from session or to remove an entire array you can use the below version of **unset_userdata()** function.

```
$this->session->unset_userdata($array_items);
```

Fetch Session Data

After setting data in session, we can also retrieve that data as shown below. **Userdata()** function will be used for this purpose. This function will return **NULL** if the data you are trying to access is not available.

```
$name=$this->session->userdata('name');
```

Example

Create a controller class called **Session_controller.php** and save it in **application/controller/Session_controller.php**.

```
<?php
class Session_controller extends CI_Controller {

    public function index()
    {
        //loading session library
        $this->load->library('session');

        //adding data to session
        $this->session->set_userdata('name','virat');

        $this->load->view('session_view');
    }

    public function unset_session_data(){
        //loading session library
    }
}
```

```

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

        //removing session data
        $this->session->unset_userdata('name');
        $this->load->view('session_view');
    }

}

?>

```

Create a view file called **session_view.php** and save it in **application/views/session_view.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter Session Example</title>
</head>
<body>
    Welcome <?php echo $this->session->userdata('name'); ?>
    <br>
    <a href='http://localhost:85/CodeIgniter-3.0.1/CodeIgniter-3.0.1/index.php/sessionex/unset'>Click Here</a> to unset session data.
</body>
</html>

```

Make the changes in the **routes.php** file in **application/config/routes.php** and add the following line at the end of the file.

```
$route['sessionex'] = 'Session_Controller';
```

Execute the above example by using the following address. Replace **yoursite.com** with the URL of your site.

<http://yoursite.com/index.php/sessionex>

14. Flashdata

While building web application, we need to store some data for only one time and after that we want to remove that data. For example, to display some error message or information message. In PHP, we have to do it manually but CodeIgniter has made this job simple for us. In CodeIgniter, flashdata will only be available until the next request, and it will get deleted automatically.

Add Flashdata

We can simply store flashdata as shown below.

```
$this->session->mark_as_flash('item');
```

- **mark_as_flash()** function is used for this purpose, which takes only one argument of the value to be stored. We can also pass an array to store multiple values.
- **set_flashdata()** function can also be used, which takes two arguments, name and value, as shown below. We can also pass an array.

```
$this->session->set_flashdata('item','value');
```

Retrieve Flashdata

Flashdata can be retrieved using the **flashdata()** function which takes one argument of the item to be fetched as shown below. **flashdata()** function makes sure that you are getting only flash data and not any other data.

```
$this->session->flashdata('item');
```

If you do not pass any argument, then you can get an array with the same function.

Example

Create a class called **FlashData_Controller.php** and save it at **application/controller/FlashData_Controller.php**.

```
<?php  
class FlashData_Controller extends CI_Controller {  
  
    public function index()  
    {  
        //Load session library  
        $this->load->library('session');
```

```

        //redirect to home page
        $this->load->view('flashdata_home');

    }

public function add(){
    //Load session library
    $this->load->library('session');
    $this->load->helper('url');

    //add flash data
    $this->session->set_flashdata('item','item-value');

    //redirect to home page
    redirect('flashdata');
}

?>

```

Create a view file called **flashdata_home.php** and save it in **application/views/flashdata_home.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter Flashdata Example</title>
</head>
<body>
    Flash Data Example
<h2><?php echo $this->session->flashdata('item'); ?></h2>
<a href='flashdata/add'>Click Here</a> to add flash data.
</body>
</html>

```

Make the changes in the **routes.php** file in **application/config/routes.php** and add the following line at the end of the file.

```
$route['flashdata'] = 'FlashData_Controller';
$route['flashdata/add'] = 'FlashData_Controller/add';
```

Execute the above example by visiting the following link. Replace the yoursite.com with the URL of your site.

```
http://yoursite.com/index.php/flashdata
```

After visiting the above URL, you will see a screen as shown below.

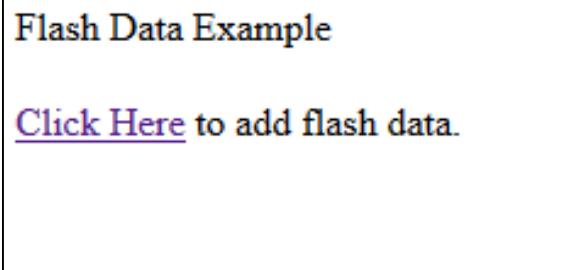
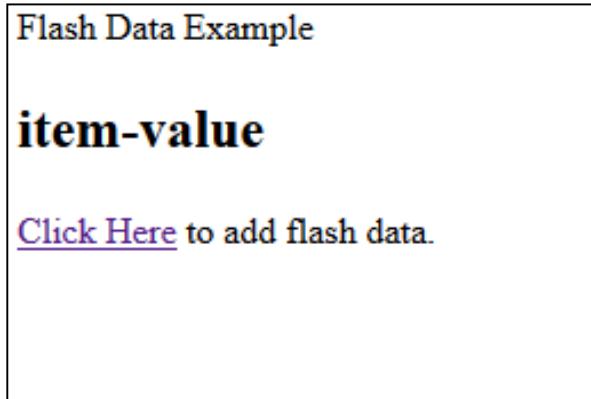


Figure: Flashdata Example

Click on "**Click Here**" link and you will see a screen as shown below. Here, in this screen you will see a value of flash data variable. Refresh the page again and you will see a screen like above and flash data variable will be removed automatically.



15. Tempdata

In some situations, where you want to remove data stored in session after some specific time-period, this can be done using **tempdata** functionality in CodeIgniter.

Add Tempdata

To add data as **tempdata**, we have to use **mark_as_tempdata()** function. This function takes two argument items or items to be stored as **tempdata** and the expiration time for those items are as shown below.

```
// 'item' will be erased after 300 seconds(5 minutes)
$this->session->mark_as_temp('item',300);
```

You can also pass an array to store multiple data. All the items stored below will be expired after 300 seconds.

```
$this->session->mark_as_temp(array('item','item2'),300);
```

You can also set different expiration time for each item as shown below.

```
// 'item' will be erased after 300 seconds, while 'item2'
// will do so after only 240 seconds
$this->session->mark_as_temp(array(
    'item'=>300,
    'item2'=>240
));
```

Retrieve Tempdata

We can retrieve the tempdata using **tempdata()** function. This function assures that you are getting only tempdata and not any other data. Look at the example given below to see how to retrieve tempdata. **tempdata()** function will take one argument of the item to be fetched.

```
$this->session->tempdata('item');
```

If you omit the argument, then you can retrieve all the existing tempdata.

Remove Tempdata

Tempdata is removed automatically after its expiration time but if you want to remove tempdata before that, then you can do as shown below using the **unset_tempdata()** function, which takes one argument of the item to be removed.

```
$this->session->unset_tempdata('item');
```

Example

Create a class called **Tempdata_controller.php** and save it in **application/controller/Tempdata_controller.php**.

```
<?php

class Tempdata_controller extends CI_Controller {

    public function index()
    {
        $this->load->library('session');
        $this->load->view('tempdata_view');

    }

    public function add()
    {
        $this->load->library('session');
        $this->load->helper('url');

        //tempdata will be removed after 5 seconds
        $this->session->set_tempdata('item','item-value',5);

        redirect('tempdata');
    }
}

?>
```

Create a file called **tempdata_view.php** and save it in **application/views/tempdata_view.php**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter Tempdata Example</title>
</head>
<body>
    Temp Data Example
<h2><?php echo $this->session->tempdata('item'); ?></h2>
```

```
<a href='tempdata/add'>Click Here</a> to add temp data.  
</body>  
</html>
```

Make the changes in the **routes.php** file in application/config/routes.php and add the following line at the end of the file.

```
$route['tempdata'] = "Tempdata_controller";  
$route['tempdata/add'] = "Tempdata_controller/add";
```

Execute the above example by visiting the following link. Replace the yoursite.com with the URL of your site.

<http://your-site.com/index.php/tempdata>

After visiting the above URL, you will see a screen as shown below.

Temp Data Example

[**Click Here** to add temp data.](#)

Click on "**Click Here**" link and you will see a screen as shown below.

Here, in this screen you will see a value of temp data variable. Refresh the same page after five seconds again as we have set the temp data for five seconds and you will see a screen like above and temp data variable will be removed automatically after five seconds. If you refresh the same page before 5 seconds, then the temp data will not be removed, as the time period is not over.

Temp Data Example

item-value

[**Click Here** to add temp data.](#)

Destroying a Session

In PHP, we are using the **session_destroy()** function to destroy the session and in CodeIgniter we can destroy the function as shown below.

```
$this->session->sess_destroy();
```

After calling this function, all the session data including the **flashdata** and **tempdata** will be deleted permanently and cannot be retrieved back.

16. Cookie Management

Cookie is a small piece of data sent from web server to store on client's computer. CodeIgniter has one helper called "Cookie Helper" for cookie management.

Syntax	<code>set_cookie(\$name[, \$value = "", \$expire = "", \$domain = "", \$path = '/'[, \$prefix = "[, \$secure = FALSE[, \$httponly = FALSE]]]]])</code>
Parameters	<ul style="list-style-type: none">• \$name (mixed) – Cookie name or associative array of all of the parameters available to this function• \$value (string) – Cookie value• \$expire (int) – Number of seconds until expiration• \$domain (string) – Cookie domain (usually: .yourdomain.com)• \$path (string) – Cookie path• \$prefix (string) – Cookie name prefix• \$secure (bool) – Whether to only send the cookie through HTTPS• \$httponly (bool) – Whether to hide the cookie from JavaScript
Return type	void

In the **set_cookie()** function, we can pass all the values using two ways. In the first way, only array can be passed and in the second way, individual parameters can also be passed.

Syntax	<code>get_cookie(\$index[, \$xss_clean = NULL]])</code>
Parameters	<ul style="list-style-type: none">• \$index (string) – Cookie name• \$xss_clean (bool) – Whether to apply XSS filtering to the returned value
Return	The cookie value or NULL if not found
Return type	mixed

The `get_cookie()` function is used to get the cookie that has been set using the `set_cookie()` function.

Syntax	<code>delete_cookie(\$name[, \$domain = "[", \$path = '/[, \$prefix = "]]])</code>
Parameters	<ul style="list-style-type: none"> • \$name (<i>string</i>) – Cookie name • \$domain (<i>string</i>) – Cookie domain (usually: <code>.yourdomain.com</code>) • \$path (<i>string</i>) – Cookie path • \$prefix (<i>string</i>) – Cookie name prefix
Return type	<code>void</code>

The `delete_cookie()` function is used to delete the cookie().

Example

Create a controller called **Cookie_controller.php** and save it at **application/controller/Cookie_controller.php**

```
<?php
class Cookie_controller extends CI_Controller {

    function __construct()
    {
        parent::__construct();
        $this->load->helper(array('cookie', 'url'));
    }

    public function index()
    {
        set_cookie('cookie_name','cookie_value','3600');
        $this->load->view('Cookie_view');
    }

    public function display_cookie()
    {
        echo get_cookie('cookie_name');
        $this->load->view('Cookie_view');
    }
}
```

```

}

public function deletecookie()
{
    delete_cookie('cookie_name');
    redirect('cookie/display');
}

?>

```

Create a view file called **Cookie_view.php** and save it at **application/views/Cookie_view.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter View Example</title>
</head>
<body>
    <a href='display'>Click Here</a> to view the cookie.<br>
    <a href='delete'>Click Here</a> to delete the cookie.
</body>
</html>

```

Change the routes.php file in application/config/routes.php to add route for the above controller and add the following line at the end of the file.

```

$route['cookie'] = "Cookie_controller";
$route['cookie/display'] = "Cookie_controller/display_cookie";
$route['cookie/delete'] = "Cookie_controller/deletecookie";

```

After that, you can execute the following URL in the browser to execute the example.

```
http://yoursite.com/index.php/cookie
```

It will produce an output as shown in the following screenshot.

[Click Here](#) to view the cookie.
[Click Here](#) to delete the cookie.

17. Common Functions

CodeIgniter library functions and helper functions need to be initialized before they are used but there are some common functions, which do not need to be initialized.

These common functions and their descriptions are given below.

Syntax	is_php(\$version)
Parameters	<ul style="list-style-type: none">• \$version (string) – Version number
Return	TRUE if the running PHP version is at least the one specified or FALSE if not
Return type	void
Description	Determines if the PHP version being used is greater than the supplied version number.

Syntax	is_really_writable(\$file)
Parameters	<ul style="list-style-type: none">• \$file (string) – File path
Return	TRUE if the path is writable, FALSE if not
Return type	bool
Description	checks to see if file is writable or not.

Syntax	config_item(\$key)
Parameters	<ul style="list-style-type: none">• \$key (string) – Config item key
Return	Configuration key value or NULL if not found
Return type	mixed
Description	This function is used to get the configuration item.

Syntax	set_status_header(\$code[, \$text = ""])
Parameters	<ul style="list-style-type: none">• \$code (int) – HTTP Response status code• \$text (string) – A custom message to set with the status code
Return type	void
Description	This function permits you to manually set a server status header.

Syntax	remove_invisible_characters(\$str[, \$url_encoded = TRUE])
Parameters	<ul style="list-style-type: none">• \$str (string) – Input string• \$url_encoded (bool) – Whether to remove URL-encoded characters as well
Return	Sanitized string
Return type	string
Description	This function prevents inserting NULL characters between ASCII characters

Syntax	html_escape(\$var)
Parameters	• \$var (mixed) – Variable to escape (string or array)
Return	HTML escaped string(s)
Return type	mixed
Description	This function acts as a native PHP <code>htmlspecialchars()</code> function.

Syntax	get_mimes()
Return	An associative array of file types
Return type	array
Description	This function returns a <i>reference</i> to the MIMEs array from <code>application/config/mimes.php</code> .

Syntax	is_https()
Return	TRUE if currently using HTTP-over-SSL, FALSE if not
Return type	bool
Description	Returns TRUE if a secure (HTTPS) connection is used and FALSE in any other case (including non-HTTP requests).

Syntax	is_cli()
Return	TRUE if currently running under CLI, FALSE otherwise
Return type	bool
Description	Returns TRUE if the application is run through the command line and FALSE if not.

Syntax	function_usable(\$function_name)
Parameters	• \$function_name (string) – Function name
Return type	bool
Description	Returns TRUE if a function exists and is usable, FALSE otherwise.

Given below is an example, which demonstrates all of the above functions.

Example

Here we have created only one controller in which we will use the above functions. Copy the below given code and save it at **application/controller/CommonFun_Controller.php**.

```
<?php
class CommonFun_Controller extends CI_Controller {

    public function index()
    {
```

```

set_status_header(200);
echo is_php('5.3')."<br>";

var_dump(is_really_writable('./Form.php'));
echo config_item('language')."<br>";
echo remove_invisible_characters('This is a &zwnj;test','UTF-8')."<br>";
$str = '< This > is \' a " test & string';
echo html_escape($str)."<br>";
echo "is_https():".var_dump(is_https())."<br>";
echo "is_cli():".var_dump(is_cli())."<br>";
var_dump(function_usable('test'))."<br>";
echo "get_mimes():".print_r(get_mimes())."<br>";
}

public function test(){
    echo "Test function";
}
?>

```

Change the **routes.php** file at application/config/routes.php to add route for the above controller and add the following line at the end of the file.

```
$route['commonfunctions'] = 'CommonFun_Controller';
```

Type the following URL in the address bar of your browser to execute the example.

```
http://yoursite.com/index.php/commonfunctions
```

18. Page Caching

Caching a page will improve the page load speed. If the page is cached, then it will be stored in its fully rendered state. Next time, when the server gets a request for the cached page, it will be directly sent to the requested browser.

Cached files are stored in **application/cache** folder. Caching can be enabled on per page basis. While enabling the cache, we need to set the time, until which it needs to remain in cached folder and after that period, it will be deleted automatically.

Enable Caching

Caching can be enabled by executing the following line in any of the controller's method.

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

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

Disable Caching

Cache file gets deleted when it expires but when you want to delete it manually, then you have to disable it. You can disable the caching by executing the following line.

```
// Deletes cache for the currently requested URI  
$this->output->delete_cache();  
  
// Deletes cache for /foo/bar  
$this->output->delete_cache('/foo/bar');
```

Example

Create a controller called **Cache_controller.php** and save it in **application/controller/Cache_controller.php**

```
<?php  
class Cache_controller extends CI_Controller {  
  
    public function index()  
    {  
        $this->output->cache(1);  
        $this->load->view('test');  
    }  
}
```

```

public function delete_file_cache()
{
    $this->output->delete_cache('cachecontroller');
}
?>

```

Create a view file called **test.php** and save it in **application/views/test.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter View Example</title>
</head>
<body>
    CodeIgniter View Example
</body>
</html>

```

Change the **routes.php** file in **application/config/routes.php** to add route for the above controller and add the following line at the end of the file.

```

$route['cachecontroller'] = 'Cache_controller';
$route['cachecontroller/delete'] = 'Cache_controller/delete_file_cache';

```

Type the following URL in the browser to execute the example.

```
http://yoursite.com/index.php/cachecontroller
```

After visiting the above URL, you will see that a cache file for this will be created in **application/cache** folder. To delete the file, visit the following URL.

```
http://yoursite.com/index.php/cachecontroller/delete
```

19. Page Redirection

While building web application, we often need to redirect the user from one page to another page. CodeIgniter makes this job easy for us. The **redirect()** function is used for this purpose.

Syntax	redirect(\$uri = "", \$method = 'auto', \$code = NULL)
Parameters	<ul style="list-style-type: none">• \$uri (string) – URI string• \$method (string) – Redirect method ('auto', 'location' or 'refresh')• \$code (string) – HTTP Response code (usually 302 or 303)
Return type	void

The first argument can have two types of URI. We can pass full site URL or URI segments to the controller you want to direct.

The second optional parameter can have any of the three values from auto, location or refresh. The default is auto.

The third optional parameter is only available with location redirects and it allows you to send specific HTTP response code.

Example

Create a controller called **Redirect_controller.php** and save it in **application/controller/Redirect_controller.php**

```
<?php  
class Redirect_controller extends CI_Controller {  
  
    public function index()  
    {  
        /*Load the URL helper*/  
        $this->load->helper('url');  
  
        /*Redirect the user to some site*/  
        redirect('http://www.tutorialspoint.com');  
    }  
}
```

```

public function computer_graphics()
{
    /*Load the URL helper*/
    $this->load->helper('url');

redirect('http://www.tutorialspoint.com/computer_graphics/index.htm');

}

public function version2()
{
    /*Load the URL helper*/
    $this->load->helper('url');

    /*Redirect the user to some internal controller's method*/
    redirect('redirect/computer_graphics');

}
?>

```

Change the **routes.php** file in **application/config/routes.php** to add route for the above controller and add the following line at the end of the file.

```

$route['redirect'] = 'Redirect_controller';
$route['redirect/version2'] = 'Redirect_controller/version2';
$route['redirect/computer_graphics'] = 'Redirect_controller/computer_graphics';

```

Type the following URL in the browser, to execute the example.

```
http://yoursite.com/index.php/redirect
```

The above URL will redirect you to the tutorialspoint.com website and if you visit the following URL, then it will redirect you to the computer graphics tutorial at tutorialspoint.com.

```
http://yoursite.com/index.php/redirect/computer_graphics
```

20. Application Profiling

When building a web application, we are very much concerned about the performance of the website in terms of how much time the controller took to execute and how much memory is used. Not only the performance, but we also need to see the insights of data like POST data, data of database queries, session data etc. for debugging purpose while developing some application. CodeIgniter has made this job easier for us by profiling an application.

Enable Profiling

To enable profiling of your application, simply execute the command given below in any of the method of your controller.

```
$this->output->enable_profiler(TRUE);
```

The report of the profiling can be seen at the bottom of the page after enabling it.

Disable Profiling

To disable profiling of your application, simply execute the command given below in any of the method of your controller.

```
$this->output->enable_profiler(FALSE);
```

Enable/Disable Profiler Section

Profiling can be done on section basis. You can enable or disable profiling of a section by setting a Boolean value TRUE or FALSE. If you want to set profiling on the application then you can do in a file located in **application/config/profiler.php**

For example, the following command will enable profiling queries for the whole application.

```
$config['queries'] = TRUE;
```

In the following table, the key is the parameter, which can be set in the config array to enable or disable a particular profile.

Key	Description	Default
benchmarks	Elapsed time of Benchmark points and total execution time	TRUE
config	CodeIgniterConfig variables	TRUE
controller_info	The Controller class and method requested	TRUE
get	Any GET data passed in the request	TRUE
http_headers	The HTTP headers for the current request	TRUE

memory_usage	Amount of memory consumed by the current request, in bytes	TRUE
post	Any POST data passed in the request	TRUE
queries	Listing of all database queries executed, including execution time	TRUE
uri_string	The URI of the current request	TRUE
session_data	Data stored in the current session	TRUE
query_toggle_count	The number of queries after which the query block will default to hidden.	25

The profiler set in the file in **application/config/profiler.php** can be overridden by using the **set_profiler_sections()** function in controllers as shown below.

```
$sections = array(
    'config' => TRUE,
    'queries' => TRUE
);

$this->output->set_profiler_sections($sections);
```

21. Benchmarking

Setting Benchmark Points

If you want to measure the time taken to execute a set of lines or memory usage, you can calculate it by using Benchmarking points in CodeIgniter. There is a separate “**Benchmarking**” class for this purpose in CodeIgniter.

This class is loaded automatically; you do not have to load it. It can be used anywhere in your controller, view, and model classes. All you have to do is to mark a start point and end point and then execute the **elapsed_time()** function between these two marked points and you can get the time it took to execute that code as shown below.

```
<?php  
    $this->benchmark->mark('code_start');  
  
    // Some code happens here  
  
    $this->benchmark->mark('code_end');  
  
    echo $this->benchmark->elapsed_time('code_start', 'code_end');  
?>
```

To display the memory usage, use the function **memory_usage()** as shown in the following code.

```
<?php  
    echo $this->benchmark->memory_usage();  
?>
```

Example

Create a controller called **Profiler_controller.php** and save it in **application/controller/Profiler_controller.php**

```
<?php  
class Profiler_controller extends CI_Controller {  
  
    public function index()  
    {  
        //enable profiler
```

```

        $this->output->enable_profiler(TRUE);
        $this->load->view('test');

    }

    public function disable()
    {
        //disable profiler
        $this->output->enable_profiler(FALSE);
        $this->load->view('test');
    }
}

?>

```

Create a view file called **test.php** and save it at **application/views/test.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter View Example</title>
</head>
<body>
    CodeIgniter View Example
</body>
</html>

```

Change the routes.php file at **application/config/routes.php** to add route for the above controller and add the following line at the end of the file.

```

$route['profiler'] = "Profiler_controller";
$route['profiler/disable'] = "Profiler_controller/disable";

```

After that, you can type the following URL in the address bar of your browser to execute the example.

```
http://yoursite.com/index.php/profiler
```

The above URL will enable the profiler and it will produce an output as shown in the following screenshot.

CodeIgniter View Example

BENCHMARKS

Loading Time: Base Classes	0.0351
Controller Execution Time (Profiler Controller / Index)	0.0077
Total Execution Time	0.0436

GET DATA

No GET data exists

MEMORY USAGE

2,221,240 bytes

POST DATA

No POST data exists

URI STRING

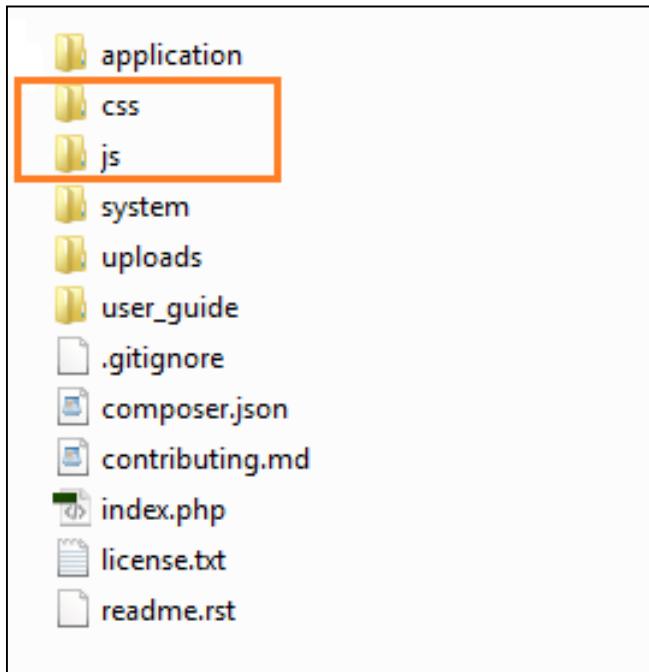
profiler

To disable the profiling, execute the following URL.

<http://yoursite.com/index.php/profiler/disable>

22. Adding JS and CSS

Adding JavaScript and CSS (Cascading Style Sheet) file in CodeIgniter is very simple. You have to create JS and CSS folder in root directory and copy all the .js files in JS folder and .css files in CSS folder as shown in the figure.



For example, let us assume, you have created one JavaScript file **sample.js** and one CSS file **style.css**. Now, to add these files into your views, load URL helper in your controller as shown below.

```
$this->load->helper('url');
```

After loading the URL helper in controller, simply add the below given lines in the view file, to load the sample.js and style.css file in the view as shown below.

```
<link rel="stylesheet" type="text/css" href="php echo base_url();?<br/?>css/style.css">  
<script type='text/javascript' src="php echo base_url();?<br/?>js/sample.js"></script>
```

Example

Create a controller called **Test.php** and save it in **application/controller/Test.php**

```
<?php  
class Test extends CI_Controller {
```

```

public function index()
{
    $this->load->helper('url');
    $this->load->view('test');
}
?>

```

Create a view file called **test.php** and save it at **application/views/test.php**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter View Example</title>
    <link rel="stylesheet" type="text/css" href=<?php echo base_url(); ?>css/style.css">
    <script type='text/javascript' src=<?php echo base_url(); ?>js/sample.js"></script>
</head>
<body>
    <a href='javascript:test()'>Click Here</a> to execute the javascript function.
</body>
</html>

```

Create a CSS file called **style.css** and save it at **css/style.css**

```

body{
    background:#000;
    color:#FFF;
}

```

Create a JS file called **sample.js** and save it at **js/sample.js**

```

function test()
{
    alert('test');
}

```

Change the **routes.php** file in **application/config/routes.php** to add route for the above controller and add the following line at the end of the file.

```
$route['profiler'] = "Profiler_controller";  
$route['profiler/disable'] = "Profiler_controller/disable";
```

Use the following URL in the browser to execute the above example.

```
http://yoursite.com/index.php/test
```

23. Internationalization

The language class in CodeIgniter provides an easy way to support multiple languages for internationalization. To some extent, we can use different language files to display text in many different languages.

We can put different language files in application/language directory. System language files can be found at system/language directory, but to add your own language to your application, you should create a separate folder for each language in application/language directory.

Creating Language files

To create a language file, you must end it with **_lang.php**. For example, you want to create a language file for French language, then you must save it with **french_lang.php**. Within this file you can store all your language texts in key, value combination in **\$lang** array as shown below.

```
$lang['key'] = 'val';
```

Loading Language file

To use any of the language in your application, you must first load the file of that particular language to retrieve various texts stored in that file. You can use the following code to load the language file.

```
$this->lang->load('filename', 'language');
```

- **filename** – It is the name of file you want to load. Don't use extension of file here but only name of file.
- **Language** – It is the language set containing it.

Fetching Language Text

To fetch a line from the language file simply execute the following code.

```
$this->lang->line('language_key');
```

Where **language_key** is the key parameter used to fetch value of the key in the loaded language file.

Autoload Languages

If you need some language globally, then you can autoload it in **application/config/autoload.php** file as shown below.

```

|-----+
| Auto-load Language files
|-----+
| Prototype:
|
| $autoload['language'] = array('lang1', 'lang2');
|
| NOTE: Do not include the "_lang" part of your file. For example
| "codeigniter_lang.php" would be referenced as array('codeigniter');
|
*/
$autoload['language'] = array();

```

Simply, pass the different languages to be autoloaded by CodeIgniter.

Example

Create a controller called **Lang_controller.php** and save it in **application/controller/Lang_controller.php**

```

<?php
class Lang_controller extends CI_Controller {

    public function index()
    {
        //Load form helper
        $this->load->helper('form');

        //Get the selected language
        $language = $this->input->post('language');

        //Choose language file according to selected lanaguage
        if($language == "french")
            $this->lang->load('french_lang','french');
        else if($language == "german")
            $this->lang->load('german_lang','german');
        else
            $this->lang->load('english_lang','english');

        //Fetch the message from language file.
        $data['msg'] = $this->lang->line('msg');

        $data['language'] = $language;
    }
}

```

```

    //Load the view file
    $this->load->view('lang_view',$data);

}
?>

```

Create a view file called **lang_view.php** and save it at **application/views/lang_view.php**

```

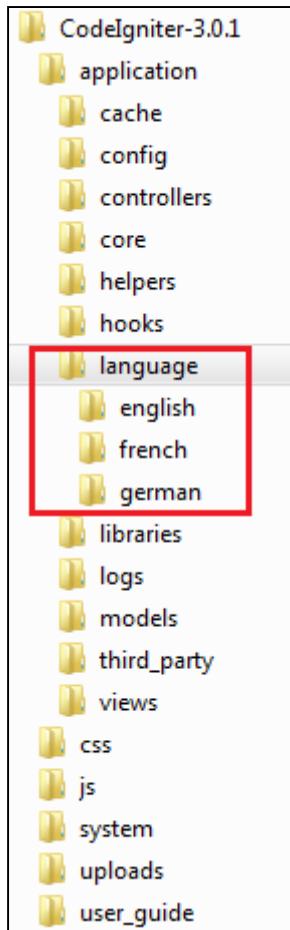
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>CodeIgniter Internationalization Example</title>
</head>
<body>
<?php
    echo form_open('/lang');
?>
<select name="language" onchange="javascript:this.form.submit();">
<?php
    $lang =
array( 'english'=>"English", 'french'=>"French", 'german'=>"German");
    foreach($lang as $key=>$val)
    {
        if($key == $language)
            echo "<option value='".$key."'"
selected>".$val."</option>";
        else
            echo "<option value='".$key."'>".$val."</option>";
    }
?>
</select>
<br>
<?php
    form_close();
    echo $msg;

```

```
?>
</body>

</html>
```

Create three folders called English, French, and German in **application/language** as shown in the figure below.



Copy the below given code and save it in **english_lang.php** file in **application/language/english** folder.

```
<?php
$lang['msg'] = "CodeIgniter Internationalization example.";
?>
```

Copy the below given code and save it in **french_lang.php** file in **application/language/French** folder.

```
<?php
$lang['msg'] = "Exemple CodeIgniter internationalisation.";
?>
```

Copy the below given code and save it in **german_lang.php** file in **application/language/german** folder.

```
<?php  
    $lang['msg'] = "CodeIgniter Internationalisierung Beispiel.";  
?>
```

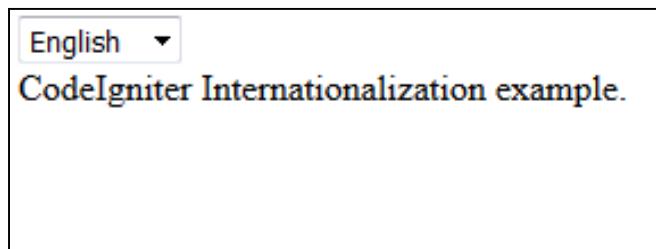
Change the **routes.php** file in **application/config/routes.php** to add route for the above controller and add the following line at the end of the file.

```
$route['lang'] = "Lang_controller";
```

Execute the following URL in the browser to execute the above example.

```
http://yoursite.com/index.php/lang
```

It will produce an output as shown in the following screenshot. If you change the language in the dropdown list, the language of the sentence written below the dropdown will also change accordingly.



24. CodeIgniter – Security

XSS Prevention

XSS means cross-site scripting. CodeIgniter comes with XSS filtering security. This filter will prevent any malicious JavaScript code or any other code that attempts to hijack cookie and do malicious activities. To filter data through the XSS filter, use the **xss_clean()** method as shown below.

```
$data=$this->security->xss_clean($data);
```

You should use this function only when you are submitting data. The optional second Boolean parameter can also be used to check image file for XSS attack. This is useful for file upload facility. If its value is true, means image is safe and not otherwise.

SQL Injection Prevention

SQL injection is an attack made on database query. In PHP, we are use **mysql_real_escape_string()** function to prevent this along with other techniques but CodeIgniter provides inbuilt functions and libraries to prevent this.

We can prevent SQL Injection in CodeIgniter in the following three ways:

- Escaping Queries
- Query Biding
- Active Record Class

Escaping Queries

```
<?php  
$username= $this->input->post('username');  
$query = 'SELECT * FROM subscribers_tbl WHERE user_name='.$this->db->escape($email);  
$this->db->query($query);  
?>
```

\$this->db->escape() function automatically adds single quotes around the data and determines the data type so that it can escape only string data.

Query Biding

```
<?php  
$sql = "SELECT * FROM some_table WHERE id = ? AND status = ? AND author = ?";
```

```
$this->db->query($sql, array(3, 'live', 'Rick'));
?>
```

In the above example, the question mark(?) will be replaced by the array in the second parameter of query() function. The main advantage of building query this way is that the values are automatically escaped which produce safe queries. CodeIgniter engine does it for you automatically, so you do not have to remember it.

Active Record Class

```
<?php
$this->db->get_where('subscribers_tbl',array('status'=> active','email' =>
'info@arjun.net.in'));
?>
```

Using active records, query syntax is generated by each database adapter. It also allows safer queries, since the values escape automatically.

Hiding PHP Errors

In production environment, we often do not want to display any error message to the users. It is good if it is enabled in the development environment for debugging purposes. These error messages may contain some information, which we should not show to the site users for security reasons.

There are three CodeIgniter files related with errors.

PHP Error Reporting Level

Different environment requires different levels of error reporting. By default, development will show errors but testing and live will hide them. There is a file called **index.php** in root directory of CodeIgniter, which is used for this purpose. If we pass zero as argument to **error_reporting()** function then that will hide all the errors.

Database Error

Even if you have turned off the PHP errors, MySQL errors are still open. You can turn this off in **application/config/database.php**. Set the **db_debug** option in **\$db** array to **FALSE** as shown below.

```
$db['default']['db_debug'] = FALSE;
```

Error log

Another way is to transfer the errors to log files. So, it will not be displayed to users on the site. Simply, set the **log_threshold** value in **\$config** array to 1 in **application/config/config.php** file as shown below.

```
$config['log_threshold'] = 1;
```

CSRF Prevention

CSRF stands for cross-site request forgery. You can prevent this attack by enabling it in the **application/config/config.php** file as shown below.

```
$config['csrf_protection']=TRUE;
```

When you are creating form using **form_open()** function, it will automatically insert a CSRF as hidden field. You can also manually add the CSRF using the **get_csrf_token_name()** and **get_csrf_hash()** function. The **get_csrf_token_name()** function will return the name of the CSRF and **get_csrf_hash()** will return the hash value of CSRF.

The CSRF token can be regenerated every time for submission or you can also keep it same throughout the life of CSRF cookie. By setting the value **TRUE**, in config array with key '**csrf_regeneration**' will regenerate token as shown below.

```
$config['csrf_regeneration']=TRUE;
```

You can also whitelist URLs from CSRF protection by setting it in the config array using the key '**csrf_exclude_uris**' as shown below. You can also use regular expression.

```
$config['csrf_exclude_uris']=array('api/person/add');
```

Password Handling

Many developers do not know how to handle password in web applications, which is probably why numerous hackers find it so easy to break into the systems. One should keep in mind the following points while handling passwords:

- DO NOT store passwords in plain-text format.
- Always hash your passwords.
- DO NOT use Base64 or similar encoding for storing passwords.
- DO NOT use weak or broken hashing algorithms like MD5 or SHA1. Only use strong password hashing algorithms like BCrypt, which is used in PHP's own Password Hashing functions.
- DO NOT ever display or send a password in plain-text format.
- DO NOT put unnecessary limits on your users' passwords.