Getting Started with OpenCart Module Development

Rupak Nepali

Community Experience Distilled
Getting Started with OpenCart Module Development
Master your OpenCart modules and code!
Rupak Nepali
PACKT open source

Chapter No. 1
"Getting Started with OpenCart Modules"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.1 "Getting Started with OpenCart Modules"

A synopsis of the book's content

Information on where to buy this book

About the Author

Rupak Nepali, a PHP programmer from Nepal, has been working on OpenCart since 2010 and has completed many projects and made many modules on OpenCart to meet client requirements. He handles `http://opencartnepal.com` personally as well as updates his personal site `http://rupaknepali.com`.np with his works Mr. Nepali currently works as a full-time freelancer on oDesk as well as on various freelancer sites. He holds a Bachelor's degree in Computer Information Systems.

I wish to thank my parents, especially my mother Subthara Nepali and my father Bhairab Nepali, who emphasized the importance of literacy, and my brothers who helped at every step, as well as all my friends, and seniors, who provided their support and encouragement to write this book.

Thanks to Packt Publishing who provided me with such a great opportunity and all the team members who assisted me in publishing this book.

Getting Started with OpenCart Module Development

If you can code OpenCart modules, you can customize OpenCart and make e-commerce sites easier to administer and also change the way the default OpenCart system works. This book shows you how to create all sorts of extensions: OpenCart module, Order Total module, ideas for creating payment, shipping modules, and ways to create custom pages and forms on OpenCart module to carry out the insert, edit, delete, and list functions.

This book focuses on teaching you all aspects of OpenCart modules by showing and defining code examples. The book uses default OpenCart module to clone other modules, the process by which one module gets transferred to another. It shows each and every line of code and describes them so readers know what the code does. You will clone the Google_talk module in the first chapter. In the second chapter, you will learn about all the available methods in OpenCart, and at last you will create two custom module feedback pages and the Tips Order Total modules.

Each chapter teaches you to make a new OpenCart module; you will thus be able to make three modules by reading this book. You will be able to create the Hello World module by cloning the Google talk module that you can then change to the Welcome Message module. Likewise, you will get a description of each code of default featured module of OpenCart, and then create the Feedback pages to manage the feedbacks. In the end, you will be able to create an Order Total module called Tips Order Total module.

Each chapter builds a practical module from the ground up using step-by-step instructions and examples.

What This Book Covers

Chapter 1, *Getting Started with OpenCart Module*, shows us how to clone the Google_talk module to the Hello World module and lists ways to install, configure, and uninstall the OpenCart module and show the structure of the file of admin and frontend.

Chapter 2, *Describing The Code of Extensions*, lists all global methods of OpenCart, shows you how to configure the feature module, describes the code of the feature module, shows the way to start the coding for the shipping module, and describes the payment module.

Chapter 3, *Create Custom OpenCart Module*, shows you how to create a feedback module and the Tips Order Total module. It also shows how code works and are managed.

1
Getting Started with OpenCart Modules

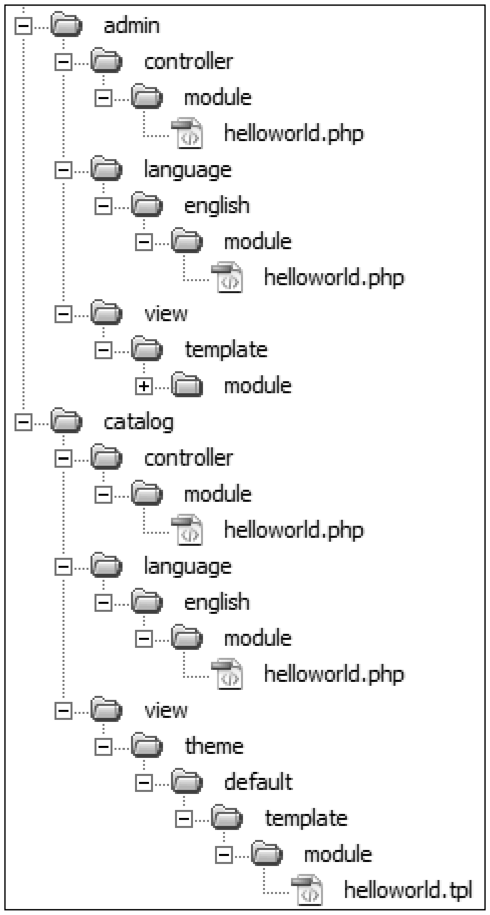
OpenCart is an e-commerce cart application built with its own in-house framework that uses the **Model View Controller** (**MVC**) language pattern; thus each module in OpenCart also follows the MVCL patterns. The controller creates logic and gathers data from the model and passes it to display them in the view. The OpenCart modules have `admin` and `catalog` folders. The files in the `admin` folder help in controlling the settings of modules and the files in the `catalog` folder handle the presentation layer (frontend). Each module has its own files by which it gets modular, and changing one module's file does not affect other modules.

Creating the Hello World module

We assume that you already know PHP and have installed OpenCart, and are familiar with the OpenCart backend and frontend, as well as you have some coding knowledge of PHP.

You are going to create the Hello World module which just has one input box in the admin settings for the module, and the same content is shown on the frontend. The first step to creating a module is using a unique name, so that there will not be a conflict with other modules. The same unique name is used to create the filename and classname to extend the controller and the model.

There are generally six to eight files that need to be created for each module, and they follow a similar structure. If there is an interaction with the database tables, we have to create two extra models. The following screenshot shows the hierarchy of files and folders of an OpenCart module:



```
admin
    controller
        module
            helloworld.php
    language
        english
            module
                helloworld.php
    view
        template
            module
catalog
    controller
        module
            helloworld.php
    language
        english
            module
                helloworld.php
    view
        theme
            default
                template
                    module
                        helloworld.tpl
```

So now you know the basic directory structure of OpenCart module. The file structure is divided into two sections **admin** and **catalog**. The `admin` folders and files deal with the setting of the modules and data handling, while the `catalog` folders and files handle the frontend.

Let's start with an easy way to make a module. You are going to make the duplicate of the default Google Talk module of OpenCart and change it to the Hello World module. We are using Dreamweaver to work with files.

Changes made in the admin folder

Following are the steps to make changes in the admin folder:

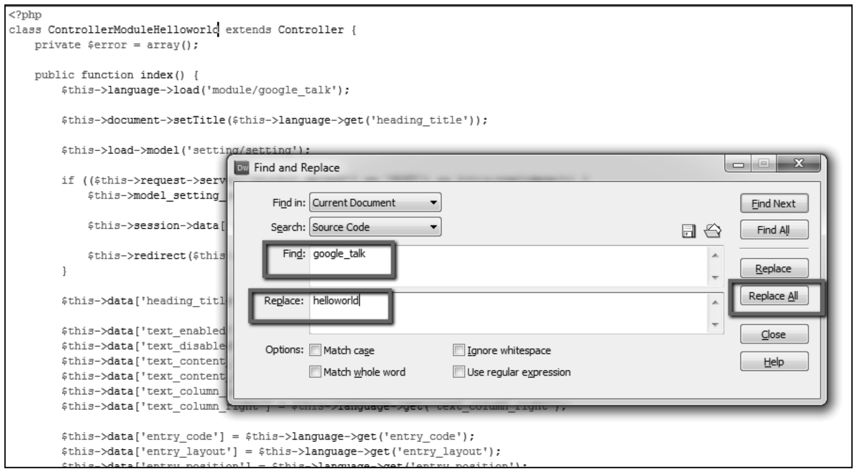
1. Navigate to `admin/controller/module/` and copy `google_talk.php` and paste in the same folder. Rename it to `helloworld.php` and open it in your favorite text editor, then look for the following line of code:

   ```
   classControllerModuleGoogleTalk extends Controller {
   ```

   Change the class name to:

   ```
   classControllerModuleHelloworld extends Controller {
   ```

2. Now find `google_talk` and replace all with `helloworld` as shown in the following screenshot:



```
<?php
class ControllerModuleHelloworld extends Controller {
    private $error = array();

    public function index() {
        $this->language->load('module/google_talk');

        $this->document->setTitle($this->language->get('heading_title'));

        $this->load->model('setting/setting');

        if (($this->request->serv
            $this->model_setting_

            $this->session->data[

            $this->redirect($this
        }

        $this->data['heading_titl

        $this->data['text_enabled
        $this->data['text_disable
        $this->data['text_content
        $this->data['text_content
        $this->data['text_column_
        $this->data['text_column_

        $this->data['entry_code'] = $this->language->get('entry_code');
        $this->data['entry_layout'] = $this->language->get('entry_layout');
```

Find in: Current Document

Search: Source Code

Find: google_talk

Replace: helloworld

Options: Match case, Match whole word, Ignore whitespace, Use regular expression

3. Then, save the file.

4. Navigate to `admin/language/english/module/` and copy `google_talk.php` and paste in the same folder; rename it to `helloworld.php` and open it. Then look for the following line of code:

   ```
   $_['entry_code'] = 'Google Talk Code:<br />
   <span class="help">Goto
     <a href="http://www.google.com/talk/service/badge/New"
       target="_blank">
       <u>Create a Google Talk chatback badge</u>
     </a> and copy &amp; paste the generated code into the
     text box.
   </span>';
   ```

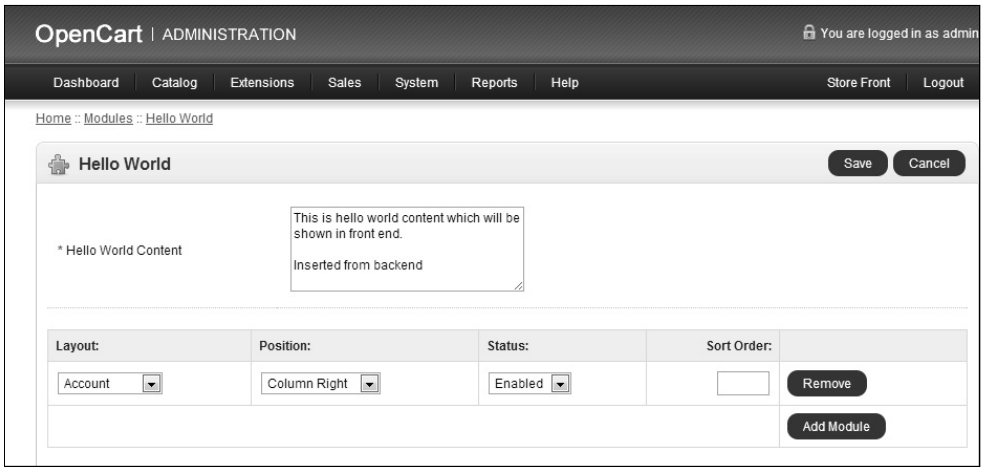
5. And replace with following code:
   ```
   $_['entry_code'] = 'Hello World Content';
   ```
6. Then again find `google_talk` and replace all with `helloworld`.
7. Then, save the file.
8. Navigate to `admin/view/template/module/` and copy the `google_talk.tpl` file and paste it in the same folder and rename it to `helloworld.tpl`; open it and look for `google_talk` and replace it with `helloworld` and save it.

Changes made in the catalog folder

Following are the steps to make changes in the catalog folder:

1. Go to `catalog/controller/module/` and copy the `google_talk.php` file and paste it in the same folder and rename it to `helloworld.php`; open it and look for the following line of code:
   ```
   class ControllerModuleGoogleTalk extends Controller {
   ```
   Change the class name to :
   ```
   class ControllerModuleHelloworld extends Controller {
   ```
2. Now look for `google_talk` and replace all with `helloworld` and save it.
3. Navigate to `catalog/language/english/module/` and copy the `google_talk.php` file and paste it in the same folder and rename it to `helloworld.php`; open it and look for `Live Chat` and replace it with `Hello World` and save it.
4. Navigate to `catalog/view/theme/default/template/module/` and copy the `google_talk.tpl` file and paste it in the same folder and rename it to `helloworld.tpl`.

With the preceding file and code changes complete, our **Hello World** module is ready to be installed. Now log in to the admin section and navigate to **Extensions | Modules**, then look for **Hello World** and click on **[install]**, then click on **[Edit]** of the Hello World module. Then type the content that you would like to show on the frontend in the **Hello World Content** field. Now click on the **Add Module** button and adjust the settings as per your requirements and click on **Save**. With the settings as per the following image, the module will be shown in the User Account links box (**Login**, **My Account**, **Edit Account**, and so on) for the customer to access as per the layout and it will be shown in the right column, as the status is enabled. The following screenshot shows the settings for the **Hello World** module:

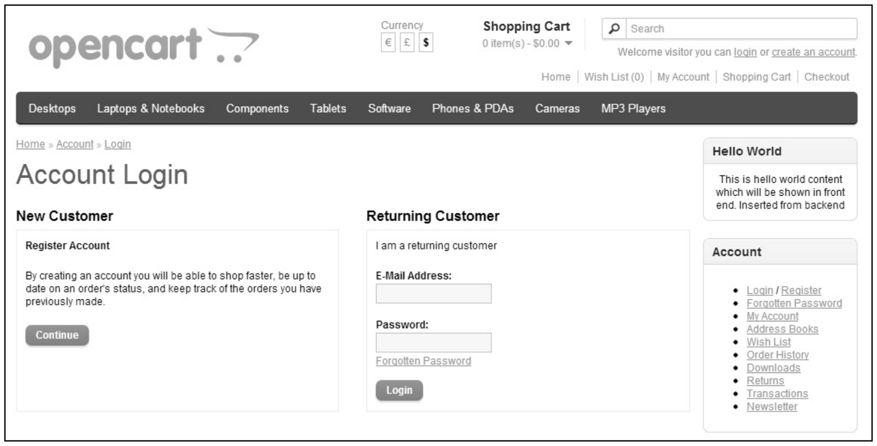


* Hello World Content: This is hello world content which will be shown in front end.

Inserted from backend

Layout:	Position:	Status:	Sort Order:
Account	Column Right	Enabled	

Now navigate to the frontend of the site and click on the **My Account** link on the home page; you will see the **Hello World** module as shown in the following screenshot:



By creating an account you will be able to shop faster, be up to date on an order's status, and keep track of the orders you have previously made.

Hello World

This is hello world content which will be shown in front end. Inserted from backend

Following are the list of files that you need to upload to your live server:

- `admin/language/english/module/helloworld.php`
- `admin/controller/module/helloworld.php`
- `admin/view/template/module/helloworld.tpl`
- `catalog/controller/module/helloworld.php`
- `catalog/language/english/module/helloworld.php`
- `catalog/view/theme/default/template/module/helloworld.tpl`

By uploading the files, installing the module, and providing the settings, your Hello World module is ready to use.

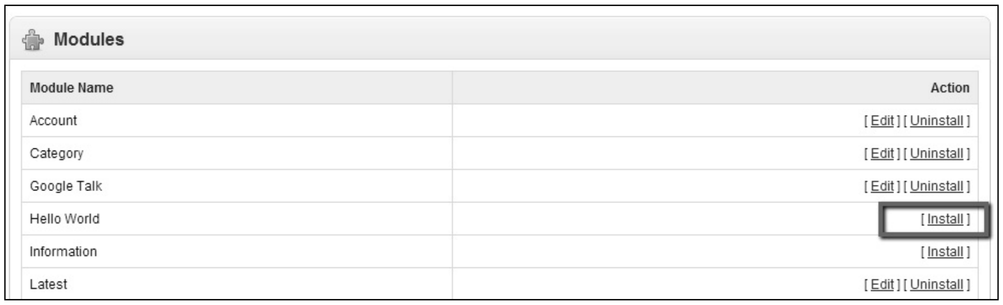
You can change the Hello World text at `catalog/language/english/module/helloworld.php` to your desired text like `Welcome to our Store` and type the welcome message at the **Hello World Content** while setting the module and showing the welcome message at the frontend.

Installing, configuring, and uninstalling a module

There are many default modules in OpenCart. How modules get installed and which are the database tables that hold the settings of the module are really big questions for the developer.

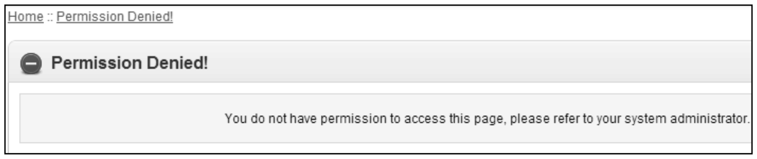
Installing a module

Navigate to **admin** | **Extensions** | **Modules**, where you will find the list of modules. Click on **[Install]** and the module gets installed, as shown in the following screenshot:



Module Name	Action
Account	[Edit] [Uninstall]
Category	[Edit] [Uninstall]
Google Talk	[Edit] [Uninstall]
Hello World	[Install]
Information	[Install]
Latest	[Edit] [Uninstall]

When you click on the **[Install]** module, the extension/module controller's install function is called. Now open `admin/controller/extension/module.php`, you will see the public function `install()`,which performs the permission check. If you get the Permission Denied! message, as shown in the following screenshot, you have to provide the access permission from **admin** | **User** | **User Group** and edit the user and check or tick mark the module/extension, so you will be able to edit the modules.



Home :: Permission Denied!

Permission Denied!

You do not have permission to access this page, please refer to your system administrator.

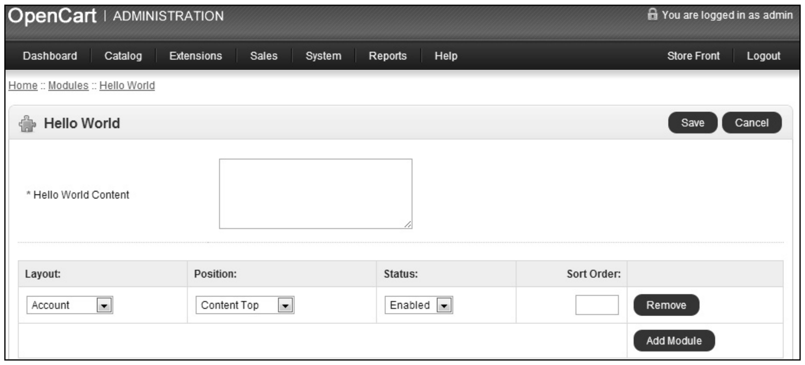
If you are provided the access, it loads the `admin/model/setting/extension.php` function `install()`.

```
public function install($type, $code) {
  $this->db->query("INSERT INTO " . DB_PREFIX ."extension SET `type`
= '" . $this->db->escape($type) . "', `code` = '" . $this->db-
>escape($code) . "'");
}
```

This means that data is inserted into the extension table of the database with type=module, and code=helloworld, in case of our Hello World module.

Configuring the module

After clicking on **[Install]** of the module, **[Edit] [Uninstall]** gets activated; after clicking on **[Edit]**, you will see the configuration section for the module. As per the Hello World module, the following screenshot shows the configuration section on clicking on **[Edit]**:



* Hello World Content

Layout:	Position:	Status:	Sort Order:
Account	Content Top	Enabled	

The **Hello World Content** field is saved in the `setting` table (`oc_setting` or as per the prefixes used during installation of OpenCart) of the database as per the name of the input box with group column of "helloworld". For this module, navigate to the file `admin/view/template/module/helloworld.tpl`, where you will find the following code:

```
<textarea name="helloworld_code" cols="40" rows="5"><?php echo
$helloworld_code; ?></textarea>
```

Thus, the message or text you typed in the text area is passed to the `admin/controller/module/helloworld.php` controller and the following lines of code is processed:

```
if (($this->request->server['REQUEST_METHOD'] == 'POST') && $this-
>validate()) {
  $this->model_setting_setting->editSetting('helloworld',
    $this->request->post);
  $this->session->data['success'] = $this->
    language->get('text_success');
  $this->redirect($this->url->link('extension/module',
    'token=' . $this->session->data['token'], 'SSL'));
}
```

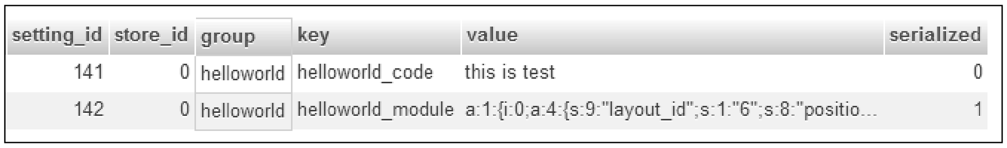
It checks if the form is submitted through the `POST` method and checks whether the **Hello World Content** field is empty or not with the validate function. If the content is not empty and the form is submitted through the `POST` method, it calls the `editSetting` function which is in `admin/model/setting/setting.php`.

```
public function editSetting($group, $data, $store_id = 0) {
  $this->db->query("DELETE FROM " . DB_PREFIX . "setting WHERE
    store_id = '" . (int)$store_id . "' AND `group` = '" . $this-
      >db->escape($group) . "'");
  foreach ($data as $key => $value) {
    if (!is_array($value)) {
      $this->db->query("INSERT INTO " . DB_PREFIX . "setting SET
        store_id = '" . (int)$store_id . "', `group` = '" . $this-
          >db->escape($group) . "', `key` = '" . $this->db-
            >escape($key) . "', `value` = '" . $this->db-
              >escape($value) . "'");
    } else {
    $this->db->query("INSERT INTO " . DB_PREFIX . "setting SET
      store_id = '" . (int)$store_id . "', `group` = '" .
        $this->db->escape($group) . "', `key` = '" . $this->
          db->escape($key) . "', `value` = '" . $this->
            db->escape(serialize($value)) . "', serialized =
              '1'");
    }
  }
}
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com . If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

As given at the controller, `$group=helloworld`, `$data` is the `$_POST`, and `$store_id` is `0`. First it deletes all the Hello World settings and then starts to insert the new values. Following are the rows inserted in the `setting` table of the database:



setting_id	store_id	group	key	value	serialized
141	0	helloworld	helloworld_code	this is test	0
142	0	helloworld	helloworld_module	a:1:{i:0;a:4:{s:9:"layout_id";s:1:"6";s:8:"positio...	1

If the value of the input field of the form is in array, the value is saved with `serialized`. Thus `serialized` becomes `1`, or else the value of `serialized` is `0`.

The `serialize($value)`, serialize function of PHP generates a storable representation of a value for an array.

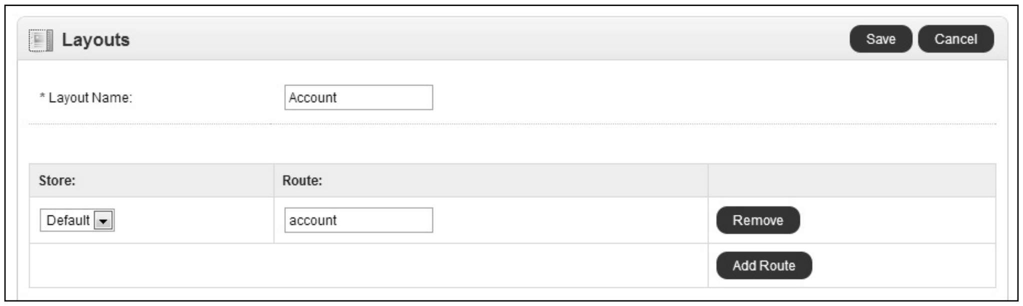
`http://php.net/manual/en/function.serialize.php`

Layouts for the module

OpenCart has default page layouts that are based on the route of the page. Some of the layouts can be found at **admin** | **System** | **Design** | **Layouts**, and they are as follows:

- Account
- Affiliate
- Category
- Checkout
- Contact
- Default
- Home
- Information
- Manufacturer
- Product
- Sitemap

Now edit one of them, let's consider **Account**, as shown in the following screenshot:



* Layout Name:	Account	
Store:	Route:	
Default	account	

The value of **Route** is **account**; this means that the module will be seen where the route value contains account. If your URL is `http://example.com/index.php?route=account/login`, the module is shown as `route=account`. If you want to show the module in the account section, you have to change the layout to `Account`.

If you like to show the module in affiliate section, you have to choose the **Affiliate** layout as the route of **Affiliate**, that is, `route=affiliate` in the URL.

Similarly, for other layouts, check the route at **admin** | **System** | **Setting** | **Design** | **Layouts** | **Edit**, see the route, and check the URL route; you will find where the module will show on choosing the layout name.

Positions for the module

There are four positions for modules. They are as follows:

- Column Left
- Column Right
- Content Top
- Content Bottom

The following table shows the available positions for modules in the frontend.

Header		
Content Left	Content Top	Content Right
	Main Content	
	Content bottom	
Footer		

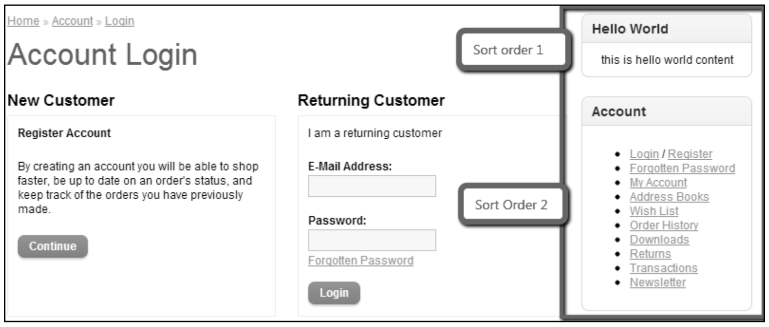
Choose as per your need of module position.

Status of the module

Status shows whether the module is enabled or disabled. If enabled, it is shown at the frontend, else it is not.

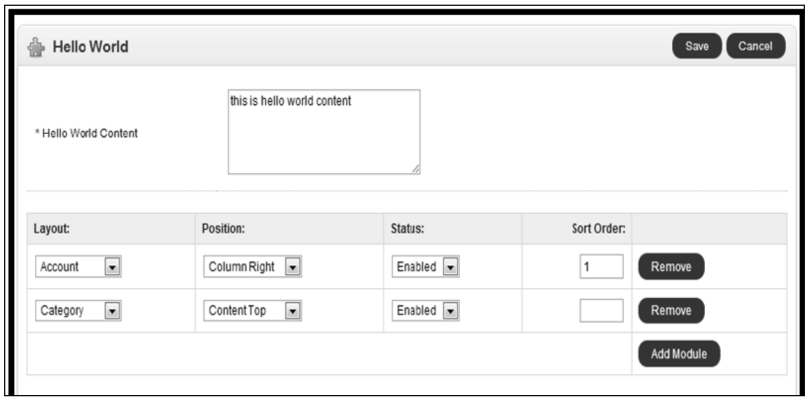
Sort order of the modules

If there is more than one module in any of the positions, sort order plays its role. Let us suppose two modules, **Hello World** and **Account**, are positioned to the right column of layout **Account**, and you like to show **Hello World** first, and then below it, the **Account** module, you have to insert **Sort order 1** for **Hello World** and **Sort order 2** for **Account**. If you do not insert sort order, it shows at the top. You will then be able to see the modules in the right column, as shown in the following screenshot:



Show same module in different layouts

We can easily show the same module in a different layout. To do this, click on the **Add Module** button and another row of the table is added; select the appropriate layout, position, status, and the sort order, then click on the **Save** button. You will be able to see the module in the respective layout. When you click on the **Add Module** button, the next row is added, as shown in the following screenshot:



* Hello World Content: this is hello world content

Layout:	Position:	Status:	Sort Order:
Account	Column Right	Enabled	1
Category	Content Top	Enabled	

Effects of clicking on the Add Module button

The **Add Module** button shows another row for the module setting. Open `admin/catalog/view/template/helloworld.tpl` and you will see the following code, which is for the **Add Module** button:

```
<a onclick="addModule();" class="button"><?php echo
  $button_add_module; ?></a>
```

On clicking the **Add Module** link, the `addModule` function is called; the `addModule` function adds a row just below the previous row.

Uninstalling the module

Navigate to **admin** | **Extensions** | **Modules**, and you will find the list of modules. Just click on **[Uninstall]**, the module gets uninstalled and all settings get deleted. Let's see how it is done. Open `admin/controller/extension/module.php`, you will see the public function `uninstall()`,which performs the permission check and if there is permission access, it loads the model setting/extension uninstall function.

File structure – admin and frontend

When someone uses the module, it is reliable to have the admin section so that the user can handle the module functionality as well as position, layout, status, and sort order by which users can show the module wherever they like.

Creating the language files for the admin module in OpenCart

Language files are also named with `MODULENAME.php`. For example, let's say we want to create a file containing hello world messages or text; we have to create `helloworld.php`. Language files use "constant=value" configuration. The constant name is used in the code; it never changes, only the value for that language changes. If English language is active, it retrieves the constant from the English language folder's file; if another language is active, it retrieves from the other language folder's file. For example, if English language is active, the constant is taken from the English language folder's file.

```
$_['text_review'] = 'Product Review';
```

If Spanish language is active, the constant is taken from the Spanish language folder's file.

```
$_['text_review'] = 'De Revisión de Producto';
```

If German language is active, the constant is taken from the German language folder's file.

```
$_['text_review'] = 'ProduktBewertung';
```

A similar process is followed for the other languages installed.

Within the file, we will assign each line of text to a variable as `$_['variablename']`. The same variable name will be used in the controller to access the text or messages. For example, in the following code:

```
$this->data['heading_title'] = $this->language
  ->get('heading_title');
```

Now on, we will use the `heading_title` controller to access the "Hello World" text.

You can see the following code at `admin/language/english/module/helloworld.php`.

```
<?php
$_['heading_title']       = 'Hello World';
$_['text_module']         = 'Modules';
$_['text_success']        = 'Success: You have modified module
  Hello World!';
$_['text_content_top']    = 'Content Top';
$_['text_content_bottom'] = 'Content Bottom';
$_['text_column_left']    = 'Column Left';
$_['text_column_right']   = 'Column Right';
$_['entry_code']          = 'Hello World Content';
$_['entry_layout']        = 'Layout:';
$_['entry_position']      = 'Position:';
$_['entry_status']        = 'Status:';
$_['entry_sort_order']    = 'Sort Order:';
$_['error_permission']    = 'Warning: You do not have permission
  to modify module Hello World!';
$_['helloworld_content']          = Hello World Content';
?>
```

Creating the controller in the admin section of the OpenCart module

Controller is the core file where all the logic and magic take place. This is also where the variables for values and language are set and passed to the view variables for display. A Controller in OpenCart is simply a class file that is named in a way that can be associated with a URL.

Consider this URL: `http://example.com/index.php?route=module/helloworld`.

In the above example, OpenCart would attempt to find a controller file `helloworld.php` in the module folder with class `ControllerModuleHelloworld`.

We can see the code at `admin/controller/module/helloworld.php` whose functionalities are described as follows:

In OpenCart, controller class names must start with the controller and the folder on which the module is located and the filename without extension. For example, in the Hello World module, the class name for the controller is `ControllerModuleHelloworld` as it is inside the module folder and the filename is `helloworld.php`. Also, always make sure your controller extends the parent controller class.

```
class ControllerModuleHelloworld extends Controller {
```

Whenever the controller is called, the `index` function (public function `index()`) is always loaded by default.

```
$this->language->load('module/helloworld');
```

The preceding line of code loads the language file variables of `helloworld.php` which is in the module folder at `admin/language/*/module/helloworld.php` (`*` represents the language folder) and now you are able to get the text or messages with reference to variables like `$this->language->get('heading_title')`. This means the Hello World text is ready to transfer to the template files.

```
$this->document->setTitle($this->language->get('heading_title'));
```

The preceding line of code sets the title of the document Hello World.

The `$this->load->model('setting/setting')` variable loads the `setting.php` file of the setting folder which is in the model folder. As explained previously, it loads `admin/model/setting/setting.php`. Your module can load any model file in its controller file using the following code, if they are in the same `admin` or `catalog` folder as the controller. You will need to specify the path to the file you want to load from the `admin` folder within the parentheses. The preceding code will load the `settings` class so we have access to the functions within the `ModelSettingSetting` class in our model's controller file. Use the following format in your code to call a function from a loaded model file:

```
$this->model_setting_setting->editSetting('helloworld',
  $this->request->post);

if (($this->request->server['REQUEST_METHOD'] == 'POST') &&
  $this->validate()) {
```

```
$this->model_setting_setting->editSetting('helloworld',
  $this->request->post);
$this->session->data['success'] = $this->
  language->get('text_success');
$this->redirect($this->url->link('extension/module',
  'token=' . $this->session->data['token'], 'SSL'));
  }
```

When a form is saved in the module section, the preceding lines of code, which are at `admin/controller/module/helloworld.php` run. If the code is submitted through the `POST` method and validates function return `true`, all the settings are saved to the database at the `setting` table and a success message is assigned to the success variable and is redirected to the list of the module page.

```
protected function validate() {
  if (!$this->user->hasPermission('modify', 'module/helloworld'))
  {
    $this->error['warning'] = $this->language
      ->get('error_permission');
  }
  if (!$this->request->post['helloworld_code']) {
    $this->error['code'] = $this->language->get('error_code');
  }
  if (!$this->error) {
    return true;
  } else {
    return false;
  }
}
```

When a form is submitted, validation is checked for whether permission is provided or not. It is checked whether the **Hello World Content** consists of the text or not. If no access is provided or no content is entered, error is returned true, by which it shows **Code Required** or **Permission Denied!** and alerts the user to provide the access or insert the content.

```
$this->data['heading_title'] = $this->language
  ->get('heading_title');
$this->data['text_enabled'] = $this->language
  ->get('text_enabled');
```

The `$this->language->get('heading_title')` variable gets the value of the `$_['heading_title']` variable from the language file `helloworld.php`, which is "Hello World" and is assigned to `$this->data['heading_title']`. Likewise, for `$this->language->get('text_enabled')`, "Enabled" is assigned to `$this->data['text_enabled']` and the same for the other files.

```
    if (isset($this->error['warning'])) {
      $this->data['error_warning'] = $this->error['warning'];
    } else {
      $this->data['error_warning'] = '';
    }
```

The Hello World module checks for access permission and gives a warning if the user has no access to the module.

```
    if (isset($this->error['code'])) {
      $this->data['error_code'] = $this->error['code'];
    } else {
      $this->data['error_code'] = '';
    }
```

If no content is inserted in the **Hello World Content** field and the user tries to save the module, it validates whether the content is inserted or not; if content is not inserted, an error is activated by which it will show the error code as "Code Required".

```
$this->data['breadcrumbs'] = array();
$this->data['breadcrumbs'][] = array(
  'text'      => $this->language->get('text_home'),
  'href'      => $this->url->link('common/home', 'token=' .
    $this->session->data['token'], 'SSL'),
  'separator' => false
);
$this->data['breadcrumbs'][] = array(
  'text'      => $this->language->get('text_module'),
  'href'      => $this->url->link('extension/module', 'token=' .
    $this->session->data['token'], 'SSL'),
  'separator' =>' :: '
);
$this->data['breadcrumbs'][] = array(
  'text'      => $this->language->get('heading_title'),
  'href'      => $this->url->link('module/helloworld', 'token=' .
    $this->session->data['token'], 'SSL'),
  'separator' =>' :: '
);
```

Breadcrumbs are defined in an array, and contain elements such as text, href, and separator. Text elements hold the word to show in the template file, href holds the link for the word, and separator holds what to use to separate between words. This is shown in the preceding lines of code.

```
    'text'      => $this->language->get('text_home'),
```

The preceding line of code holds the "Home" word as per the language file.

```
    'href'      => $this->url->link('common/home', 'token=' .
      $this->session->data['token'], 'SSL'),
```

The preceding line of code holds the link to the "Home" word.

```
    'separator' => false
```

The preceding line of code holds the separator between the breadcrumbs; if no separator is needed, `false` is assigned.

```
  $this->data['action'] = $this->url->link('module/helloworld',
    'token=' . $this->session->data['token'], 'SSL');
```

The preceding line of code will create a link and store it into the `action` variable. If we have to create the link in the admin area, we have to use it as explained previously. A token is used to preserve the admin user state.

```
  $this->data['modules'] = array();
  $this->data['modules'] = $this->config->get('helloworld_module');
```

An empty array is defined and we assign `$this->data['modules']` with all the settings of `helloworld_module`.

```
  $this->load->model('design/layout');
```

It loads the `layout.php` file of the `design` folder which is in the `model` folder. As explained previously, it loads `admin/model/design/layout.php`. The preceding code will load the layout class, so we have access to the functions within the `ModelDesignLayout` class in our module's controller file.

```
  $this->data['layouts'] = $this->model_design_layout->getLayouts();
```

The underscores (`model_design_layout`) refer to the file designations for `model/design/layout.php`. The `layouts` variable now holds all the layouts that are created at **System** | **Design** | **Layout** at the admin sections.

```
  $this->template = 'module/helloworld.tpl';
  $this->children = array('common/header','common/footer');
```

In the controller, you will need to load your module's template file in view. To do so, set `$this->template` to `$this->template = 'module/helloworld.tpl'`, and it loads `admin/view/template/module/helloworld.tpl`.

```
$this->response->setOutput($this->render());
```

The `$this->response->setOutput()` variable sends data to the browser whether it's HTML or JSON and `$this->render` constructs the output HTML from the templates/data.

Creating the template file at admin in the OpenCart module

This refers to the template or TPL files. All variables that are passed from the controller to the view can be used for displaying the output of calculations or functionality.

Open the `admin/view/template/module/helloworld.tpl` file; we are describing the code taking some snippets only.

```
<?php echo $header; ?>
<?php echo $footer; ?>
```

The `$header` and `$footer` variables are passed from the controller as the template's children.

```
$this->children = array('common/header','common/footer');
```

With this, the content of the header and footer are shown on the module section.

Breadcrumbs section for the module

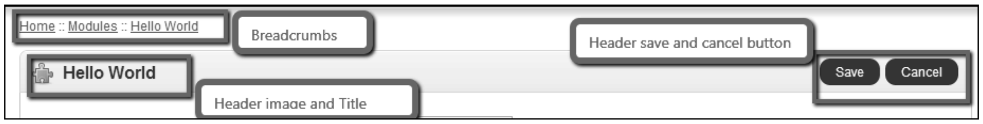
For keeping track of navigation, breadcrumbs are used; in the template file, breadcrumbs are shown by the following lines of code:

```
<div class="breadcrumb">
<?phpforeach ($breadcrumbs as $breadcrumb) {
  ?>
  <?php echo $breadcrumb['separator']; ?><a href="<?php echo
    $breadcrumb['href']; ?>"><?php echo $breadcrumb['text'];
  ?></a>
<?php } ?>
</div>
```

The `$breadcrumbs` array has been passed by the controller files. The `$breadcrumbs` array consists of the separator, URL link, and the text to show. All elements of the `$breadcrumbs` array are managed in the controller.

```
<?php if ($error_warning) {
  ?>
  <div class="warning"><?php echo $error_warning; ?></div>
<?php } ?>
```

A warning will show up if you have no permission to access or edit the module. As for the Hello World module, it checks for permission and shows a warning if the user has no access to the module. The following screenshot shows the **Breadcrumbs**, **Header image and Title**, and **Header save and cancel button:**



The following line of code shows the image icon near the heading title:

```
<h1><imgsrc="view/image/module.png" alt="" /><?php echo
  $heading_title; ?></h1>
```

The following line of code shows the heading title that is passed from the controller:

```
$this->data['heading_title'] = $this->language
  ->get('heading_title');
```

The following lines of code show the buttons to save and cancel:

```
<div class="buttons">
  <a onclick="$('#form').submit();" class="button"><?php echo
    $button_save; ?></a>
  <a href="<?php echo $cancel; ?>" class="button"><?php echo
    $button_cancel; ?></a>

</div>
```

On clicking the **Save** button, the form with ID is submitted; on clicking the **Cancel** button, it calls the extension/module controller, which means it is redirected to the list of modules.

```
<form action="<?php echo $action; ?>"
  method="post"enctype="multipart/form-data" id="form">
```

When the form code is initiated, it has `id=form`, which is used in the **Save** button to submit the form. When we click on the **Save** button, an action to the module / Hello World controller processes the submitted data.

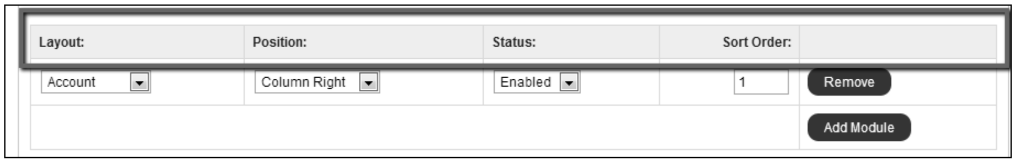
The `<span class="required">*</span>` shows the asterisk (*) in red color by the style class required.

```
<textarea name="helloworld_code" cols="40" rows="5"><?php echo
  $helloworld_code; ?></textarea>
<?php if ($error_code) {
  ?>
  <span class="error"><?php echo $error_code; ?></span>
<?php } ?>
```

This is the text area field which holds some data; if this text area is submitted empty, it shows as an error.

```
<tr>
  <td class="left"><?php echo $entry_layout; ?></td>
  <td class="left"><?php echo $entry_position; ?></td>
  <td class="left"><?php echo $entry_status; ?></td>
  <td class="right"><?php echo $entry_sort_order; ?></td>
  <td></td>
</tr>
```

The table heading is shown by the preceding code and it will look as shown in the following screenshot:



Layout:	Position:	Status:	Sort Order:	
Account	Column Right	Enabled	1	Remove
				Add Module

In the following code snippet, the `$module_row` variable is defined. It is assigned to zero and is increased with the `foreach` loop, so it is the count of the module rows that increases on clicking on the **Add Module** button.

```
<?php $module_row = 0; ?>
<?phpforeach ($modules as $module) { ?>
```

The `$modules` array carries the setting of the module; if it is empty, only the **Add Module** button is shown.

```
<select name="helloworld_module[<?php echo $module_row;
  ?>][layout_id]">
  <?php foreach ($layouts as $layout) {
  ?>
    <?php if ($layout['layout_id'] == $module['layout_id']) {
      ?>
      <option value="<?php echo $layout['layout_id']; ?>"
        selected="selected"><?php echo $layout['name'];
      ?></option>
    <?php } else { ?>
    <option value="<?php echo $layout['layout_id']; ?>"><?php echo
      $layout['name']; ?></option>
    <?php } ?>
  <?php } ?>
</select>
```

The preceding code shows the **Layout** option. If the layout id matches the module layout id, which has been already saved, the selected layout is shown among other layouts, else layouts are shown as default. The `layout` arrays have been passed from the controller. Similarly, for the position, select fieldname as `helloworld_module` with its second element as `position`.

```
<select name="helloworld_module[<?php echo $module_row;
  ?>][position]">
```

As we already know, there are four positions described in OpenCart; they are content top, content bottom, column left, and column right. The position module code for the content top is as follows:

```
<?php if ($module['position'] == 'content_top') {
  ?>
  <option value="content_top" selected="selected"><?php echo
    $text_content_top; ?></option>
<?php } else {
  ?>
  <option value="content_top"><?php echo $text_content_top;
  ?></option>
<?php } ?>
```

If module position is already defined and is equal to `content_top`, content top is selected, else others are selected as default. It works in a similar way for the content bottom, column left, and column right.

```
<select name="helloworld_module[<?php echo $module_row;
   ?>][status]">
  <?php if ($module['status']) {
    ?>
    <option value="1" selected="selected"><?php echo
      $text_enabled; ?></option>
    <option value="0"><?php echo $text_disabled; ?></option>
  <?php } else { ?>
    <option value="1"><?php echo $text_enabled; ?></option>
    <option value="0" selected="selected"><?php echo
      $text_disabled; ?></option>
  <?php } ?>
</select>
```

The preceding code is to show the module status; if module is `enabled`, `option value` is equal to `1`, else it is `0`. If module status is defined or equal to `1`, it shows that the module is already defined, so `enabled` is selected. If it is not defined, `disabled` is selected.

```
<input type="text" name="helloworld_module[<?php echo $module_row;
  ?>][sort_order]" value="<?php echo $module['sort_order']; ?>"
  size="3" />
```

The preceding code holds the sort order of the module.

```
<a onclick="$('#module-row<?php echo $module_row; ?>').remove();"
  class="button"><?php echo $button_remove; ?></a>
```

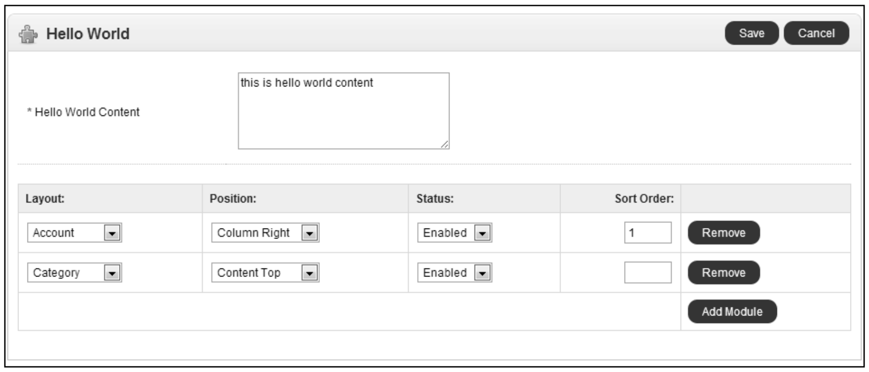
The preceding code line removes the rows when we click on the **Remove** button.

```
<a onclick="addModule();" class="button"><?php echo
  $button_add_module; ?></a>
```

On clicking on the **Add Module** link, function `addModule` is called, which adds a row just below the previous row.

```
function addModule() {}
```

The preceding function adds the rows for the modules setting. We can add as many modules as we like, just keep on clicking on the **Add Module** button. The following screenshot shows multiple rows for setting after clicking on the **Add Module** button:



* Hello World Content: this is hello world content

Layout:	Position:	Status:	Sort Order:
Account	Column Right	Enabled	1
Category	Content Top	Enabled	

Creating the language file for catalog (frontend) module in OpenCart

You can create a language file in a similar way as we did in the admin section. For the frontend, your language file will be located at `catalog/language/english/module/MODULENAME.php`. The filename should be the same as the module name. As per the Hello World module, the language file name is `helloworld.php`, it is created at `catalog/language/english/module/` and consists only of the following code:

```
<?php
  // Heading
  $_['heading_title']  = 'Hello World';
?>
```

The `Hello World` text is assigned to `heading_title`; with the same `heading_title`, it is accessible to the controller.

Creating the controller file for catalog (frontend) module in OpenCart

A controller file of a module for the frontend is found at `catalog/controller/module/MODULENAME.php`; as per the Hello World module, we can see the `helloworld.php` files at `catalog/controller/module`. Since we named the file `helloworld.php` and put it at `module/`folder, the controller classname will be `ControllerModuleHelloworld`.

```
class ControllerModuleHelloworld extends Controller {
```

Also, always make sure your controller extends the parent controller class so that it can inherit all its functions.

```
protected function index() {
```

The `index` function is always loaded by default if the second segment of the URL is empty. We can load the module controller at `http://example.com/index.php?route=module/helloworld/index` or `http://example.com/index.php?route=module/helloworld`.

Here the second segment of the URI is index; if you have created other functions, we can call the function of the module by passing it into the second segment of the URL.

```
$this->language->load('module/helloworld');
```

Loading of language files is done with the preceding line of code. According to the previous line, the `helloworld.php` file at `catalog/language/english/module/` is loaded if English language is active or it will load as per the language activated. For example, if Spanish language is active, it loads from `catalog/language/spanish/module/`.

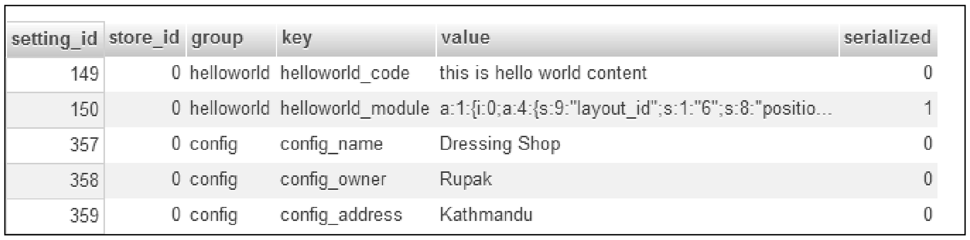
```
$this->data['heading_title'] = $this->language->get('heading_title');
```

The preceding line fetches the text "Hello World" with `$this->language->get('heading_title');` and assigns it to the `heading_title` variable of the `data` array. The `$heading_title` file will show "Hello World" in the template files.

```
if (isset($this->request->server['HTTPS']) && (($this->request
  ->server['HTTPS'] == 'on') || ($this->request->server['HTTPS']
    == '1'))) {
    $this->data['code'] = str_replace('http', 'https',
      html_entity_decode($this->config->get('helloworld_code')));
    } else {
      $this->data['code'] = html_entity_decode($this->config
      ->get('helloworld_code'));
    }
```

The first line of code checks whether SSL is active. If SSL is active, the link's `http` of `$this->config->get('helloworld_code')` is replaced with `https`.

You will be able to get the value of the `setting` table in a database by passing the key. For example, consider the `setting` table of a database that consists of the following rows, as shown in the following screenshot:



setting_id	store_id	group	key	value	serialized
149	0	helloworld	helloworld_code	this is hello world content	0
150	0	helloworld	helloworld_module	a:1:{i:0;a:4:{s:9:"layout_id";s:1:"6";s:8:"positio...	1
357	0	config	config_name	Dressing Shop	0
358	0	config	config_owner	Rupak	0
359	0	config	config_address	Kathmandu	0

If you want to show `Dressing Shop`, you can get it easily wherever you like in the controller, model, or template files. You just have to type the following line of code:

```
echo $this->config->get('config_name');
```

But if `serialized` is equal to `1`, it means that the value is stored in an array.

```
if (file_exists(DIR_TEMPLATE . $this->config
  ->get('config_template') . '/template/module/helloworld.tpl')) {
    $this->template = $this->config->get('config_template') .
      '/template/module/helloworld.tpl';
    } else {
    $this->template = 'default/template/module/helloworld.tpl';
    }
  $this->render();
```

You can get an active template name from `$this->config->get('config_template');` the preceding lines of code check whether the `helloworld.tpl` file is on the active template or not. If the file is found in the active template, it uses it, or it will use one from the default template. It will be better if we keep the files on the default theme.

Creating the template file for catalog (frontend) module in OpenCart

You can find the template file at `catalog/view/theme/<template name>/module;` as for the Hello World module, the file name is `helloworld.tpl`. OpenCart frontend template files have deeper folder structures than the admin ones because admin sections can have only one template. For the frontend, on the other hand, there can be any number of templates; among them, one is selected from the **admin** | **system** | **setting** | **edit** | **the store** and at the **store** tab choose the best template under the **Template** field.

A folder named `<template name>` is created at `catalog/view/theme`. One of the basic rules in OpenCart is never to edit the default theme template file because if OpenCart does not find certain template files on your theme `<template name>` folder, it will find them on the default theme. While upgrading, the changes made on your custom theme will also get overridden. If template files are not found on the default theme, it shows the following error:

Notice: Error: Could not load template catalog/view/theme/customtheme/template/module/helloworld.tpl! in system\engine\controller.php

Here, the theme folder's name is `customtheme`.

If you see this kind of error, it means that `helloworld.tpl` is missing on the `customtheme` and default theme folders. So you need to create the `helloworld.tpl` file at `catalog/view/theme/customtheme/template/module/` or `catalog/view/theme/default/template/module/`. Since the `helloworld.tpl` file is not the default file of OpenCart, we can place it either on `customtheme` or in default theme.

If you require any changes on the default theme template files, you have to copy the files and folders to the `customtheme` folder and make changes on the `customtheme` folder's files, so upgrading it will help in preserving your changes. The following are the code on `catalog/view/theme/default/module/helloworld.tpl`.

```
<div class="box">
  <div class="box-heading"><?php echo $heading_title; ?></div>
  <div class="box-content" style="text-align: center;"><?php echo
    $code; ?></div>
</div>
```

The `$heading_title` file holds the text "Hello World" and `$code` holds the message or text that is inserted into the Hello World module at the backend.

Summary

In this chapter, we duplicated the Google_talk module to create the Hello World module. Hello World is created, installed, configured, and uninstalled. On configuration, we inserted some data and showed the same at the frontend.

We found out how code works in the Hello World module and its file and folder structure. We also described all the code that we used in the Hello World module's files. Taking reference of Hello World module, we should be able to go through other modules and become familiar with the modules of OpenCart.

Where to buy this book

You can buy Getting Started with OpenCart Module Development from the Packt Publishing website: `http://www.packtpub.com/getting-started-with-opencart-module-development/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.