**CodeIgniter 3**

**autoload.php**

In CodeIgniter 3, the autoload.php file is used to specify which libraries, helpers, models, and other resources should be loaded automatically when your application starts. This file is typically located in the application/config directory of your CodeIgniter project.

**Hooks :**

hooks are a feature that allows you to tap into the core execution of the framework at specific points and perform custom actions. Hooks enable you to extend and modify the behavior of the framework without having to directly modify its core files. This can be useful for tasks such as logging, authentication, and custom routing.

Here's how you can use hooks in CodeIgniter 3:

**1.Enable Hooks:** First, you need to enable hooks in your application/config/config.php file. Find the $config['enable\_hooks'] variable and set it to TRUE:

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

**2.Define Hook Points:** In the application/config/hooks.php file, you can define the hook points and specify which functions or methods should be called at those points. Each hook point is associated with a specific event in the framework's execution process.

For example, you might define a hook like this:

**$hook['pre\_controller'] = array(**

**'class' => 'MyHook',**

**'function' => 'preControllerMethod',**

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

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

**);**

In this example, when the "pre\_controller" event is triggered (before the main controller is loaded), it will call the preControllerMethod function from the MyHook class located in the hooks directory.

**3.Create Hook Class:** Create the hook class and method as specified in the hook configuration. In this example, you would create a MyHook.php file in the application/hooks directory and define the preControllerMethod function inside it:

**<?php**

**class MyHook**

**{**

**public function preControllerMethod()**

**{**

**// Your custom code to run before the controller is executed**

**}**

**}**

**4.Run Custom Code:** Your custom code inside the hook method will run at the specified hook point in the CodeIgniter execution process.

Hooks can be very powerful for extending the functionality of your CodeIgniter application in a modular and organized way. They allow you to keep your custom code separate from the core framework code, making your application more maintainable and easier to upgrade.

1. **Example for Hooks :**

Certainly! Let's create a simple example of how to use hooks in CodeIgniter 3. In this example, we'll create a hook that logs the time before and after a controller is executed.

**Enable Hooks :** Ensure that hooks are enabled in your **application/config/config.php** file:

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

**Define Hook Points:** In application/config/hooks.php, define a hook point called pre\_controller that logs the start time and a hook point called post\_controller that logs the end time:

**$hook['pre\_controller'] = array(**

**'class' => 'LogHook',**

**'function' => 'logStartTime',**

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

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

**);**

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

**'class' => 'LogHook',**

**'function' => 'logEndTime',**

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

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

**);**

**Create Hook Class:** Create a LogHook.php file in the application/hooks directory and define the LogHook class with the logStartTime and logEndTime methods:

**<?php**

**class LogHook**

**{**

**public function logStartTime()**

**{**

**$start\_time = microtime(true);**

**log\_message('debug', 'Controller Execution Started at ' . date('Y-m-d H:i:s', $start\_time));**

**}**

**public function logEndTime()**

**{**

**$end\_time = microtime(true);**

**log\_message('debug', 'Controller Execution Ended at ' . date('Y-m-d H:i:s', $end\_time));**

**}**

**}**

**?>**

**View Log Output:** When you load any controller in your CodeIgniter application, the logStartTime method will log the start time, and the logEndTime method will log the end time in the CodeIgniter log files. You can view these logs in the application/logs directory.

Now, when you access any page in your CodeIgniter application, you'll see log entries indicating when the controller execution started and when it ended. This is a simple example of how hooks can be used to perform custom actions at specific points in the CodeIgniter execution process. You can extend this concept to implement more complex functionality or integrate third-party libraries as needed.