

**СПРАВОЧНАЯ ИНФОРМАЦИ**

**по PHP фреймворку CodeIgniter**

**Оглавление.**

|  |  |
| --- | --- |
| **- Глава 1. Введение ……………………………………………………………………………………………....** | **4** |
| - Возможности CodeIgniter | 4 |
| - Ход выполнения приложения | 5 |
| - Модель-Отображение-Контроллер | 5 |
| - Дизайн и архитектура ядра | 6 |
| **Глава 2. Общие темы.……………………………………………………………………………………………** | **7** |
| - URL в CodeIgniter | 7 |
| - Зарезервированные имена | 8 |
| - Контроллеры | 10 |
| - Отображения | 17 |
| - Модели | 22 |
| - Хелперы (помощники) | 26 |
| - Плагины | 29 |
| - Использование библиотек CodeIgniter | 30 |
| - Создание собственных библиотек | 31 |
| - Создание классов ядра | 35 |
| - Hooks - Расширение ядра | 37 |
| - Автоматически загружаемые ресурсы | 39 |
| - Общие функции | 39 |
| - Скаффолдинг | 40 |
| - URI Роутинг | 42 |
| - Обработка ошибок | 44 |
| - Кеширование WEB страниц | 45 |
| - Профилирование приложения | 46 |
| - Управление приложениями | 47 |
| - Альтернативный синтаксис PHP | 48 |
| - Безопасность | 49 |
| **Глава 3. Классы.…………………………………………………………………………………….……………….** | **51** |
| - Benchmarking | 51 |
| - Calendaring | 53 |
| - Config | 57 |
| - Класс для работы с базой данных | 59 |
| --- [Быстрый старт: Примеры использования](http://code-igniter.ru/user_guide/database/examples.html) | 60 |
| --- [Конфигурация для работы с базами данных](http://code-igniter.ru/user_guide/database/configuration.html) | 63 |
| --- [Подключение к базе данных](http://code-igniter.ru/user_guide/database/connecting.html) | 66 |
| --- [Выполнение запросов](http://code-igniter.ru/user_guide/database/queries.html) | 67 |
| --- [Генерация результатов запроса](http://code-igniter.ru/user_guide/database/results.html) | 72 |
| --- [Функции хелпера запросов](http://code-igniter.ru/user_guide/database/helpers.html) | 73 |
| --- [Класс Active Record](http://code-igniter.ru/user_guide/database/active_record.html) | 74 |
| --- [Транзакции](http://code-igniter.ru/user_guide/database/transactions.html) | 92 |
| --- [Метаданные таблиц](http://code-igniter.ru/user_guide/database/table_data.html) | 95 |
| --- [Метаданные полей](http://code-igniter.ru/user_guide/database/fields.html) | 96 |
| --- [Вызов произвольных функций](http://code-igniter.ru/user_guide/database/call_function.html) | 97 |
| --- [Кеширование запросов](http://code-igniter.ru/user_guide/database/caching.html) | 98 |
| --- [Манипуляции базой данных с помощью Database Forge](http://code-igniter.ru/user_guide/database/forge.html) | 101 |
| --- [Утилиты для работы с базами данных](http://code-igniter.ru/user_guide/database/utilities.html) | 106 |
| - Email | 110 |
| - Encryption | 114 |
| - File Uploading | 116 |
| - FTP | 124 |
| - HTML Table | 128 |
| - Класс для работы с изображениями | 133 |
| - Input and Security | 141 |
| - Loader | 144 |
| - Language | 145 |
| - Output | 147 |
| - Pagination | 148 |
| - Класс для работы с сессиями | 150 |
| - Trackback | 155 |
| - Template Parser | 159 |
| - Класс оформления | 163 |
| - Unit Testing | 164 |
| - URI | 167 |
| - User Agent | 170 |
| - Validation | 173 |
| - XML-RPC | 187 |
| - Zip Encoding | 197 |
| **Глава 4. Хелперы…………………………………………………………………………………………………..** | **201** |
| - Array | 201 |
| - Compatibility | 202 |
| - Cookie | 202 |
| - Date | 204 |
| - Directory | 208 |
| - Download | 210 |
| - Email | 211 |
| - File | 212 |
| - Form | 213 |
| - HTML | 221 |
| - Inflector | 228 |
| - Language | 229 |
| - Number | 229 |
| - Path | 230 |
| - Security | 230 |
| - Smiley | 231 |
| - String | 234 |
| - Text | 236 |
| - Typography | 238 |
| - URL | 239 |
| - XML | 243 |
| Глава 5. Дополнительные ресурсы | 244 |
| - Общая схема фреймворка | 244 |
| - Для заметок |  |

**Глава 1. Введение**

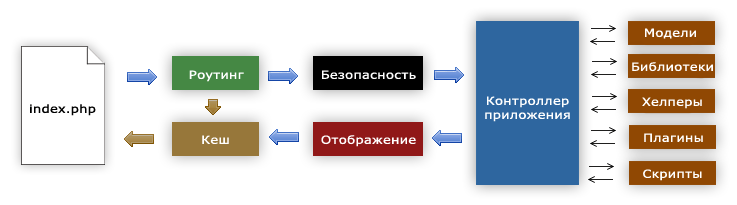
Возможности CodeIgniter

Просто посмотреть возможности — это не самый лучший путь для того чтобы судить о фреймворке. Список возможностей ничего вам не скажет об опыте программистов или о том, на сколько разумно реализованы эти опции. Единственно верный способ судить о качестве продукта — использовать его и посмотреть исходный код.  Установка CodeIgniter проста, как детская игра, поэтому мы рекомендуем вам сделать это прямо сейчас. А в списке ниже мы изложим лишь основные черты фреймворка.

* Базируется на системе Модель-Отображение-Контроллер
* Совместим с PHP 4
* Невероятно легкий
* Полная поддержка многих классов баз данных для разных платформ
* Поддержка Active Record
* Валидация форм и данных
* Безопасность и фильтрация XSS
* Управление сессиями
* Класс отправки Email. Поддерживаются вложения, HTML/Text, различные протоколы (sendmail, SMTP, and Mail) и т.п.
* Библиотека обработки изображений (кадрирование, изменение размера, поворот и т.п.). Поддерживается GD, ImageMagick и NetPBM
* Класс загрузки файлов
* Класс работы с FTP
* Локализации
* Пейджинг (разбиение по страницам)
* Кодирование информации
* Бенчмаркинг
* Полное кеширование страниц
* Протоколирование ошибок
* Профилирование
* Скаффолдинг
* Класс календаря
* Класс обработки браузера пользователя
* Класс работы с ZIP
* Класс обработки шаблонов
* Trackback Class
* Библиотека XML-RPC
* Класс тестирования
* Дружественные поисковым системам URL
* Гибкий роутинг URI
* Поддержка для Hooks, расширений классов и Плагинов.
* Большая библиотека хелперов

Ход выполнения приложения

Следующий рисунок показывает в какой последовательности происходит выполнение приложения:



1. index.php выступает в роли фронт-контроллера, он инициализирует подключение остальных частей кода CodeIgniter.
2. Роутер анализирует HTTP запрос чтобы определить что надо сделать с этим адресом.
3. Если существует кеш этой страницы, то он напрямую выдается в браузер, минуя логику приложения.
4. Безопасность. Перед загрузкой контроллера приложения HTTP запрос и любая информация, отправленная пользователем, фильтруется на предмет безопасности.
5. Контроллер загружает модель, ядро библиотек, плагины, хелперы и прочие ресурсы, необходимые для выполнения этого запроса.
6. Наконец подключается отображение и данные отдаются в браузер пользователя. Если включено кеширование, то отображение записывается на диск, чтобы в следующий раз быть считанным с него.

Модель-Отображение-Контроллер

CodeIgniter основан на шаблоне проектирования Модель-Отображение-Контроллер. MVC — это шаблон, который позволяет отделить логику от отображения. На практике, это позволяет файлам с отображением содержать минимальное количество скриптов.

* **Модель** представляет структуру ваших данных. Обычно класс модели представляет из себя набор функций, которые помогут вам получить, добавить и обновить информацию в базе данных.
* **Отображение** содержит визуальное представление данных для конечного пользователя. В нем, как правило, содержатся обычные html страницы, но в CodeIgniter это могут быть и фрагменты, такие как шапка, колонтитул, меню и т.п.
* **Контроллер** выступает в роли *посредника* между Моделью, Отображением, и любыми другими ресурсами, необходимыми для того чтобы по HTTP запросу сформировать веб-страницу.

В CodeIgniter достаточно просто отойти от архитектуры MVC, т.к. модели не являются обязательными. Если вам не нужно дополнительное разделение, или вы считаете, что взаимодействие с моделью достаточно сложно, то вы вполне можете обойтись только Контроллером и отображением. CodeIgniter также позволяет вам включать любые скрипты и сторонние библиотеки, поэтому вы можете работать так как вам удобно.

Дизайн и архитектура ядра

Нашей целью при разработке CodeIgniter является **максимум производительности, возможностей, и гибкости при минимальном размере дистрибутива**.

Для достижения этой цели мы являемся приверженцами бенчмаркинга, рефакторинга и упрощения на каждом этапе проектирования, мы отбрасываем все, что не соответствует заявленной цели.

С технической точки зрения, CodeIgniter был создан для следующих целей:

* **Динамическое Инстанцирование.** В CodeIgniter компоненты загружаются только тогда, когда это действительно необходимо, а не повсеместно на глобальном уровне. Система работает на предположении, что для функционирования достаточно самого минимума используемых ресурсов, поэтому она очень легкая. Все необходимое может быть догружено на этот минимум уже в процессе вашей работы.
* **Свободные связи.** Чем меньше компоненты зависят друг от друга, тем более гибким становится приложение. Наша цель — сделать как можно меньше зависимостей.
* **Узконаправленные компоненты.** В CodeIgniter каждый класс и его функции в значительной степени автономны, за тем, чтобы получить максимальную полезность.

CodeIgniter является динамично подгружающимся и минимальным количеством обязательных связей. Он стремится к простоте, гибкости и высокой производительности при малых объемах дистрибутива.

**Глава 2. Общие темы**

# URL в CodeIgniter

По умолчанию адреса в CodeIgniter являются дружественными для поисковых систем и понятны для человека. CodeIgniter использует подход, основанный на **сегментах** адреса:

www.your-site.com/**news**/**article**/**my\_article**

**Замечание:** Вариант со строкой запроса в URL тоже может быть использован, но об этом ниже.

## Сегменты URI

Сегменты в URL, соответствуют архитектуре Модель-Отображение-Контроллер и, как правило, строятся таким образом:

www.your-site.com/**class**/**function**/**ID**

1. В первом сегменте описывается контроллер **class** к которому происходит обращение.
2. Во втором сегменте происходит ссылка на функцию **function** или метод, который необходимо вызвать.
3. В третьем и последующих сегментах описываются переменные, которые будут переданы в эту функцию.

[Класс URI](http://code-igniter.ru/user_guide/libraries/uri.html) и [URL хелпер](http://code-igniter.ru/user_guide/helpers/url_helper.html) содержат функции, которые упростят вашу работу с информацией в URI. Кроме того все адреса можно переопределять при помощи [Роутинга](http://code-igniter.ru/user_guide/general/routing.html)

, это поможет сделать приложение гибким.

## [Удаление сегмента «index.php»](http://code-igniter.ru/user_guide/general/routing.html)

[По умолчанию, **index.php** будет подключаться URL:](http://code-igniter.ru/user_guide/general/routing.html)

[www.your-site.com/**index.php**/news/article/my\_article](http://code-igniter.ru/user_guide/general/routing.html)

[Вы можете легко удалить этот файл из запроса при помощи простого запроса в файле .htaccess. Вот пример файла с использованием «негативного» метода, в котором будут перенаправлены все запросы, кроме указанных:](http://code-igniter.ru/user_guide/general/routing.html)

[RewriteEngine on  
RewriteCond $1 !^(index\.php|images|robots\.txt)  
RewriteRule ^(.\*)$ /index.php/$1 [L]](http://code-igniter.ru/user_guide/general/routing.html)

[В приведенном выше примере, любой запрос HTTP, помимо запросов к index.php, изображениям и robots.txt, трактуется как запрос для вашего файла index.php.](http://code-igniter.ru/user_guide/general/routing.html)

## [Добавление суффикса к URL](http://code-igniter.ru/user_guide/general/routing.html)

[В файле **config/config.php** вы можете указать суффикс, который будет добавляться ко всем адресам, созданным CodeIgniter. Например, если URL такой:](http://code-igniter.ru/user_guide/general/routing.html)

[www.your-site.com/index.php/products/view/shoes](http://code-igniter.ru/user_guide/general/routing.html)

[Вы можете добавить произвольный суффикс, например **.html**, сделав все страницы похожими на:](http://code-igniter.ru/user_guide/general/routing.html)

[www.your-site.com/index.php/products/view/shoes.html](http://code-igniter.ru/user_guide/general/routing.html)

## [Подключение строки запроса](http://code-igniter.ru/user_guide/general/routing.html)

[В некоторых случаях вы можете предпочесть использовать строки запроса в URL:](http://code-igniter.ru/user_guide/general/routing.html)

[index.php?c=products&m=view&id=345](http://code-igniter.ru/user_guide/general/routing.html)

[CodeIgniter опционально поддерживает эту возможность, которая может быть подключена в файле **application/config.php**. Если вы откроете ваш конфигурационный файл, то найдете там:](http://code-igniter.ru/user_guide/general/routing.html)

[$config['enable\_query\_strings'] = FALSE;  
$config['controller\_trigger'] = 'c';  
$config['function\_trigger'] = 'm';](http://code-igniter.ru/user_guide/general/routing.html)

[Если изменить «enable\_query\_strings» в положение TRUE, то эта возможность активируется. Ваши контроллеры и функции в них будут доступны при помощи определенных «триггерных» слов:](http://code-igniter.ru/user_guide/general/routing.html)

[index.php?c=controller&m=method](http://code-igniter.ru/user_guide/general/routing.html)

[**Пожалуйста заметьте:** Если вы используете режим со строкой запроса, то вам придется вручную указывать адреса в ссылках, т.к. стандартные хелперы работают только с сегментным режимом.](http://code-igniter.ru/user_guide/general/routing.html)

# [Зарезервированные имена](http://code-igniter.ru/user_guide/general/routing.html)

[Для создания понятного кода, принято называть единицы скрипта осмысленными именами. CodeIgniter придерживается этого правила, и таким образом некоторые названия нельзя использовать в ходе программирования, так как они зарезервированы. Ниже приведен список этих названий.](http://code-igniter.ru/user_guide/general/routing.html)

### [Имена контроллеров](http://code-igniter.ru/user_guide/general/routing.html)

[При создании очередного класса контроллера будьте внимательны, так как ранее предопределенные функции и классы, имеющие то же имя, будут переопределены. Не используйте следующие названия для своих контроллеров и классов:](http://code-igniter.ru/user_guide/general/routing.html)

* [Controller](http://code-igniter.ru/user_guide/general/routing.html)
* [CI\_Base](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_initialize](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_scaffolding](http://code-igniter.ru/user_guide/general/routing.html)

[Если используется PHP 4, то так же недоступны следующие названия:](http://code-igniter.ru/user_guide/general/routing.html)

* [CI\_Loader](http://code-igniter.ru/user_guide/general/routing.html)
* [config](http://code-igniter.ru/user_guide/general/routing.html)
* [database](http://code-igniter.ru/user_guide/general/routing.html)
* [dbutil](http://code-igniter.ru/user_guide/general/routing.html)
* [dbforge](http://code-igniter.ru/user_guide/general/routing.html)
* [file](http://code-igniter.ru/user_guide/general/routing.html)
* [helper](http://code-igniter.ru/user_guide/general/routing.html)
* [helpers](http://code-igniter.ru/user_guide/general/routing.html)
* [language](http://code-igniter.ru/user_guide/general/routing.html)
* [library](http://code-igniter.ru/user_guide/general/routing.html)
* [model](http://code-igniter.ru/user_guide/general/routing.html)
* [plugin](http://code-igniter.ru/user_guide/general/routing.html)
* [plugins](http://code-igniter.ru/user_guide/general/routing.html)
* [scaffolding](http://code-igniter.ru/user_guide/general/routing.html)
* [script](http://code-igniter.ru/user_guide/general/routing.html)
* [view](http://code-igniter.ru/user_guide/general/routing.html)
* [vars](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_assign\_to\_models](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_autoloader](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_init\_class](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_init\_scaffolding](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_is\_instance](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_load](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_load\_class](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_object\_to\_array](http://code-igniter.ru/user_guide/general/routing.html)

### [Функции](http://code-igniter.ru/user_guide/general/routing.html)

* [is\_really\_writable()](http://code-igniter.ru/user_guide/general/routing.html)
* [load\_class()](http://code-igniter.ru/user_guide/general/routing.html)
* [get\_config()](http://code-igniter.ru/user_guide/general/routing.html)
* [config\_item()](http://code-igniter.ru/user_guide/general/routing.html)
* [show\_error()](http://code-igniter.ru/user_guide/general/routing.html)
* [show\_404()](http://code-igniter.ru/user_guide/general/routing.html)
* [log\_message()](http://code-igniter.ru/user_guide/general/routing.html)
* [\_exception\_handler()](http://code-igniter.ru/user_guide/general/routing.html)
* [get\_instance()](http://code-igniter.ru/user_guide/general/routing.html)

### [Переменные](http://code-igniter.ru/user_guide/general/routing.html)

* [$config](http://code-igniter.ru/user_guide/general/routing.html)
* [$mimes](http://code-igniter.ru/user_guide/general/routing.html)
* [$lang](http://code-igniter.ru/user_guide/general/routing.html)

### [Константы](http://code-igniter.ru/user_guide/general/routing.html)

* [EXT](http://code-igniter.ru/user_guide/general/routing.html)
* [FCPATH](http://code-igniter.ru/user_guide/general/routing.html)
* [SELF](http://code-igniter.ru/user_guide/general/routing.html)
* [BASEPATH](http://code-igniter.ru/user_guide/general/routing.html)
* [APPPATH](http://code-igniter.ru/user_guide/general/routing.html)
* [CI\_VERSION](http://code-igniter.ru/user_guide/general/routing.html)
* [FILE\_READ\_MODE](http://code-igniter.ru/user_guide/general/routing.html)
* [FILE\_WRITE\_MODE](http://code-igniter.ru/user_guide/general/routing.html)
* [DIR\_READ\_MODE](http://code-igniter.ru/user_guide/general/routing.html)
* [DIR\_WRITE\_MODE](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_READ](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_READ\_WRITE](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_WRITE\_CREATE\_DESTRUCTIVE](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_READ\_WRITE\_CREATE\_DESTRUCTIVE](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_WRITE\_CREATE](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_READ\_WRITE\_CREATE](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_WRITE\_CREATE\_STRICT](http://code-igniter.ru/user_guide/general/routing.html)
* [FOPEN\_READ\_WRITE\_CREATE\_STRICT](http://code-igniter.ru/user_guide/general/routing.html)

# [Контроллеры](http://code-igniter.ru/user_guide/general/routing.html)

[Контроллеры — это сердце вашего приложения. В контроллере определяется как необходимо реагировать на тот или иной запрос.](http://code-igniter.ru/user_guide/general/routing.html)

* [[Что такое контроллер?](http://code-igniter.ru/user_guide/general/controllers.html#what)](http://code-igniter.ru/user_guide/general/routing.html)
* [[Hello World](http://code-igniter.ru/user_guide/general/controllers.html#hello)](http://code-igniter.ru/user_guide/general/controllers.html#what)
* [[Функции](http://code-igniter.ru/user_guide/general/controllers.html#functions)](http://code-igniter.ru/user_guide/general/controllers.html#hello)
* [[Передача сегментов URL вашим функциям](http://code-igniter.ru/user_guide/general/controllers.html#passinguri)](http://code-igniter.ru/user_guide/general/controllers.html#functions)
* [[Объявление контроллера по умолчанию](http://code-igniter.ru/user_guide/general/controllers.html#default)](http://code-igniter.ru/user_guide/general/controllers.html#passinguri)
* [[Переадресация вызовов функций](http://code-igniter.ru/user_guide/general/controllers.html#remapping)](http://code-igniter.ru/user_guide/general/controllers.html#default)
* [[Результат обработки](http://code-igniter.ru/user_guide/general/controllers.html#output)](http://code-igniter.ru/user_guide/general/controllers.html#remapping)
* [[Защищенные функции](http://code-igniter.ru/user_guide/general/controllers.html#private)](http://code-igniter.ru/user_guide/general/controllers.html#output)
* [[Организация ваших контроллеров в подпапках](http://code-igniter.ru/user_guide/general/controllers.html#subfolders)](http://code-igniter.ru/user_guide/general/controllers.html#private)
* [[Конструкторы классов](http://code-igniter.ru/user_guide/general/controllers.html#constructors)](http://code-igniter.ru/user_guide/general/controllers.html#subfolders)
* [[Зарезервированные имена функций](http://code-igniter.ru/user_guide/general/controllers.html#reserved)](http://code-igniter.ru/user_guide/general/controllers.html#constructors)

## [Что такое контроллер?](http://code-igniter.ru/user_guide/general/controllers.html" \l "reserved)

[**Контроллер это простой файл с классом, названным так чтобы ассоциироваться с URI, по которому он будет вызван.**](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Рассмотрим этот URI:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[www.your-site.com/index.php/**blog**/](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[В приведенном примере CodeIgniter попытается найти контроллер с именем **blog.php** и загрузить его.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[**Когда имя контроллера будет найдено в первом сегменте URL — он будет загружен.**](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

## [Попробуем вывести:  Hello World!](http://code-igniter.ru/user_guide/general/controllers.html" \l "reserved)

[Давайте создавать простой контроллер, чтобы вы смогли увидеть его в действии. Используя ваш текстовый редактор создайте файл **blog.php** и вставьте в него следующий код:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[<?php](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[class Blog extends Controller {](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[function index()](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[{](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[echo 'Hello World!';](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[}](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[}](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[?>](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Затем сохраните этот файл в вашей директории **application/controllers/**.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Теперь зайдите на ваш сайт, используя примерно такой адрес:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[www.your-site.com/index.php/**blog**/](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Если вы все сделали правильно, то должны увидеть **Hello World!**.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Замечание: Имена классов должны начинаться с прописных букв. Проще говоря вот правильное название:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[<?php  
class **Blog** extends Controller {  
}  
?>](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[А так **не**правильно:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[<?php  
class **blog** extends Controller {  
}  
?>](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Так же убедитесь в том, что ваш контроллер **наследует** главный контроллер, чтобы внутри вашего был доступен весь его функционал.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

## [Функции](http://code-igniter.ru/user_guide/general/controllers.html" \l "reserved)

[В приведенном выше примере функция называется **index()**. Функция с названием «index» всегда будет загружаться по умолчанию, если **второй сегмент** в URL не определен. Еще один способ просмотреть ваше сообщение «Hello World» — обратиться по этому адресу:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[www.your-site.com/index.php/**blog**/**index**/](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[**Второй сегмент адреса определяет какая функция контроллера должна быть вызвана.**](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Давайте испытаем. Добавьте новую функцию в ваш контроллер:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[<?php](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[class Blog extends Controller {](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[function index()](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[{](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[echo 'Hello World!';](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[}](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[function comments()](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[{](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[echo 'Взгляни сюда!';](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[}](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[}](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[?>](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Сейчас загрузите следующий адрес, чтобы посмотреть на работу функции **comment**:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[www.your-site.com/index.php/**blog**/**comments**/](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Вы должны увидеть новое сообщение.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

## [Передача сегментов URL вашим функциям](http://code-igniter.ru/user_guide/general/controllers.html" \l "reserved)

[Если URI содержит более двух сегментов — они будут переданы вашей функции в качестве параметров.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Например, можно вызвать такой адрес:](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[www.your-site.com/index.php/**products**/**shoes**/**sandals**/**123**](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[Ваша функция получит 3 и 4 сегменты в URL («sandals» and «123»):](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[<?php  
class Products extends Controller {  
    function shoes($sandals, $id)  
    {  
        echo $sandals;  
        echo $id;  
    }  
}  
?>](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

[**Важно:**  Если вы используете [URI Роутинг](http://code-igniter.ru/user_guide/general/routing.html), то сегменты адреса, направленные в функцию, будут сегментами route-адреса.](http://code-igniter.ru/user_guide/general/controllers.html#reserved)

## [Объявление контроллера по умолчанию](http://code-igniter.ru/user_guide/general/routing.html)

[Можно сказать CodeIgniter какой контроллер необходимо загружать, если URI не объявлен. Это возникает, когда вы запрашиваете корень сайта. Чтобы установить контроллер, выполняемый по умолчанию, откройте файл **application/config/routes.php** и установите эту переменную:](http://code-igniter.ru/user_guide/general/routing.html)

[$route['default\_controller'] = '**Blog**';](http://code-igniter.ru/user_guide/general/routing.html)

[Где **Blog** — это имя контроллера, который вы хотите использовать. Если вы сейчас загрузите ваш основной index.php, то увидите свое приветственное сообщение «Hello world».](http://code-igniter.ru/user_guide/general/routing.html)

## [Переадресация вызова функции](http://code-igniter.ru/user_guide/general/routing.html)

[Как было отмечено выше, второй сегмент URI определяет какая функция должна быть вызвана в этом классе. CodeIgniter позволяет переопределить это поведение при помощи функции **\_remap()**:](http://code-igniter.ru/user_guide/general/routing.html)

[function \_remap()  
{  
    // Здесь любой код...  
}](http://code-igniter.ru/user_guide/general/routing.html)

[**Важно:**  Если ваш контроллер содержит функцию **\_remap()**, то она всегда будет определять какую функцию загружать независимо от того, что написано в URL запросе. Она перекрывает нормальное поведение контроллера, создавая свой локальный роутинг.](http://code-igniter.ru/user_guide/general/routing.html)

[В **\_remap()** передается название вызываемой функции, обычно это второй сегмент в URI:](http://code-igniter.ru/user_guide/general/routing.html)

[function \_remap(**$method**)  
{  
    if ($method == 'some\_method')  
    {  
        $this->$method();  
    }  
    else  
    {  
        $this->default\_method();  
    }  
}](http://code-igniter.ru/user_guide/general/routing.html)

## [Результат обработки](http://code-igniter.ru/user_guide/general/routing.html)

[CodeIgniter имеет класс вывода, который принимает готовое отображение и отдает его браузеру пользователя. Больше информации об этом вы можете найти на страницах [Отображение](http://code-igniter.ru/user_guide/general/views.html) и [Output class](http://code-igniter.ru/user_guide/libraries/output.html). В некоторых случаях вам может понадобиться пост-обработка сформированного отображения и выдача ее браузеру самостоятельно. CodeIgniter позволяет вам добавить в контроллер функцию **\_output()** которая будет принимать готовые к выдаче данные.](http://code-igniter.ru/user_guide/general/routing.html)

[**Важно:**  Если ваш контроллер содержит функцию называющуюся **\_output()**, то в нее **всегда** будут передаваться отрендеренные отображения, вместо того чтобы отдавать их напрямую в браузер. Первый параметр этой функции будет содержать окончательный вывод.](http://code-igniter.ru/user_guide/libraries/output.html)

[Вот пример:](http://code-igniter.ru/user_guide/libraries/output.html)

[function \_output($output)  
{  
    echo $output;  
}](http://code-igniter.ru/user_guide/libraries/output.html)

[Пожалуйста заметьте, что ваша функция **\_output()** будет получать данные в их окончательном состоянии. Бенчмарк и загрузка будут посчитаны, файл будет записан в кеш (если вы его используете), все заголовки будут отправлены (если вы используете [Output Class](http://code-igniter.ru/user_guide/libraries/output.html)), и только после этого будет выполнена функция \_output(). Если вы на странице выводите информацию о времени выполнения и нагрузке скриптом, то эта информация не будет точной, т.к. в ней учитывается только время исполнения до вызова **\_output()**. Есть также альтернативный способ обработки результатов до вывода их в браузер. Для того чтобы ознакомиться с ним прочитайте о возможностях класса [Output Class](http://code-igniter.ru/user_guide/libraries/output.html).](http://code-igniter.ru/user_guide/libraries/output.html)

## [Защищенные функции](http://code-igniter.ru/user_guide/libraries/output.html)

[В некоторых случаях вы можете захотеть скрыть ту или иную функцию от публичного доступа. Чтобы сделать функцию приватной — просто добавьте символ подчеркивания перед ее названием и она не будет отображаться при запросе через URL. Например, если вы назвали функцию примерно так:](http://code-igniter.ru/user_guide/libraries/output.html)

[function \_utility()  
{  
  // какой-то код  
}](http://code-igniter.ru/user_guide/libraries/output.html)

[Попытки получить к ней доступ, набрав ее имя в запросе окажутся неудачными:](http://code-igniter.ru/user_guide/libraries/output.html)

[[www.your-site.com/index.php/**blog**/**\_utility**/](http://www.your-site.com/index.php/blog/_utility/)](http://code-igniter.ru/user_guide/libraries/output.html)

## [Организация ваших контроллеров в подпапках](http://www.your-site.com/index.php/blog/_utility/)

[При разработке большого проекта может оказаться вполне удобным деление контроллеров по подпапкам. CodeIgniter позволяет вам это сделать.](http://www.your-site.com/index.php/blog/_utility/)

[Просто создайте внутри папки **application/controllers** каталог и разместите класс вашего контроллера внутри него.](http://www.your-site.com/index.php/blog/_utility/)

[**Замечание:**  При использовании этой возможности в первом сегменте URL необходимо указать папку с этим контроллером. Например, ваш контроллер расположен здесь:](http://www.your-site.com/index.php/blog/_utility/)

[application/controllers/**products**/shoes.php](http://www.your-site.com/index.php/blog/_utility/)

[URI для вызова этого контроллера будет выглядеть так:](http://www.your-site.com/index.php/blog/_utility/)

[www.your-site.com/index.php/products/shoes/123](http://www.your-site.com/index.php/blog/_utility/)

[Каждая из вложенных папок может содержать контроллер по умолчанию, который будет вызван, если URL не содержит название контроллера. Просто назовите его именем по умолчанию, которое указано в вашем файле **application/config/routes.php**.](http://www.your-site.com/index.php/blog/_utility/)

[Так же вы можете переадресовать ваши контроллеры, используя [URI Роутинг](http://code-igniter.ru/user_guide/general/routing.html).](http://www.your-site.com/index.php/blog/_utility/)

## [Конструкторы классов](http://code-igniter.ru/user_guide/general/routing.html)

[Если вы намерены в каком-либо из ваших контроллеров использовать конструктор, то вам **необходимо** разместить в нем эту строчку:](http://code-igniter.ru/user_guide/general/routing.html)

[parent::Controller();](http://code-igniter.ru/user_guide/general/routing.html)

[Причина этого в том, что ваш собственный конструктор будет перекрывать конструктор главного контроллера, поэтому вам необходимо вызвать его вручную.](http://code-igniter.ru/user_guide/general/routing.html)

[Если вы не знакомы с конструктором, то в PHP 4 конструктор — это просто функция, название которой совпадает с названием класса:](http://code-igniter.ru/user_guide/general/routing.html)

[<?php  
class **Blog** extends Controller {  
       function **Blog()**  
       {  
            **parent::Controller();**  
       }  
}  
?>](http://code-igniter.ru/user_guide/general/routing.html)

[В PHP 5 конструктор использует следующий синтаксис:](http://code-igniter.ru/user_guide/general/routing.html)

[<?php  
class **Blog** extends Controller {  
       function **\_\_construct()**  
       {  
            **parent::Controller();**  
       }  
}  
?>](http://code-igniter.ru/user_guide/general/routing.html)

[Конструктор нужен, если вы хотите установить некоторые переменные в значения по умолчанию до того, как будет вызвана функция класса. Конструктор не может возвращать значение, но он может проделывать некоторую работу.](http://code-igniter.ru/user_guide/general/routing.html)

## [Зарезервированные имена функций](http://code-igniter.ru/user_guide/general/routing.html)

[Поскольку ваши контроллеры выполняются после того как будет инициализирован основной контроллер, вам необходимо быть осторожным с названиями функций в классах, т.к. они могут перекрывать зарезервированные функции, объявленные в главном контроллере. Ниже приведен список зарезервированных имен. Не называйте функции в ваших контроллерах аналогичными именами:](http://code-igniter.ru/user_guide/general/routing.html)

* [Controller](http://code-igniter.ru/user_guide/general/routing.html)
* [CI\_Base](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_initialize](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_scaffolding  
  Если вы работаете с PHP 4, то количество таких названий заметно увеличивается. Имена перечисленные ниже нельзя использовать ТОЛЬКО в том случае, если вы работаете с PHP 4.](http://code-igniter.ru/user_guide/general/routing.html)
* [CI\_Loader](http://code-igniter.ru/user_guide/general/routing.html)
* [config](http://code-igniter.ru/user_guide/general/routing.html)
* [database](http://code-igniter.ru/user_guide/general/routing.html)
* [dbutil](http://code-igniter.ru/user_guide/general/routing.html)
* [dbforge](http://code-igniter.ru/user_guide/general/routing.html)
* [file](http://code-igniter.ru/user_guide/general/routing.html)
* [helper](http://code-igniter.ru/user_guide/general/routing.html)
* [helpers](http://code-igniter.ru/user_guide/general/routing.html)
* [language](http://code-igniter.ru/user_guide/general/routing.html)
* [library](http://code-igniter.ru/user_guide/general/routing.html)
* [model](http://code-igniter.ru/user_guide/general/routing.html)
* [plugin](http://code-igniter.ru/user_guide/general/routing.html)
* [plugins](http://code-igniter.ru/user_guide/general/routing.html)
* [scaffolding](http://code-igniter.ru/user_guide/general/routing.html)
* [script](http://code-igniter.ru/user_guide/general/routing.html)
* [view](http://code-igniter.ru/user_guide/general/routing.html)
* [vars](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_assign\_to\_models](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_autoloader](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_init\_class](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_init\_scaffolding](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_is\_instance](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_load](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_load\_class](http://code-igniter.ru/user_guide/general/routing.html)
* [\_ci\_object\_to\_array](http://code-igniter.ru/user_guide/general/routing.html)

## [Вот и все!](http://code-igniter.ru/user_guide/general/routing.html)

[Вот, в двух словах, и все, что вам надо знать о контроллерах.](http://code-igniter.ru/user_guide/general/routing.html)

# [Отображения](http://code-igniter.ru/user_guide/general/routing.html)

[**Отображение** — это простая веб-страница или ее фрагмент, например шапка, подвал, боковая панель или т.п. По факту отображения могут быть очень гибко встроены в другие отображения.](http://code-igniter.ru/user_guide/general/routing.html)

[Отображения никогда не вызываются напрямую. Они должны быть загружены из [контроллеров](http://code-igniter.ru/user_guide/general/controllers.html). Помните, что в модели MVC контроллер управляет всем происходящим, именно в нем должно быть определено какое отображение загружать. Если вы не читали о [Контроллерах](http://code-igniter.ru/user_guide/general/controllers.html), то вам следует это сделать прямо сейчас, прежде чем продолжать.](http://code-igniter.ru/user_guide/general/routing.html)

[Давайте для контроллера, созданного вами в процессе прочтения о [контроллерах](http://code-igniter.ru/user_guide/general/controllers.html), создадим отображение.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Создание отображения](http://code-igniter.ru/user_guide/general/controllers.html)

[Используя ваш текстовый редактор создайте файл с названием **blogview.php** и вставьте в него следующие строки:](http://code-igniter.ru/user_guide/general/controllers.html)

<html>

<head>

<title>Мой блог</title>

</head>

<body>

<h1>Добро пожаловать в мой блог!</h1>

</body>

</html>

[Затем сохраните этот файл в директории **application/views/**.](Forms.HTML:TextArea.1)

## [Загрузка отображения](Forms.HTML:TextArea.1)

[Чтобы загрузить необходимое вам отображение используйте эту функцию:](Forms.HTML:TextArea.1)

[$this->load->view('**name**');](Forms.HTML:TextArea.1)

[Где **name** это имя вашего файла с отображением. Замечание: расширение .php не надо указывать, только если вы не используете что-то отличное от **.php**.](Forms.HTML:TextArea.1)

[Теперь откройте ваш контроллер, ранее названный **blog.php** и замените в нем функцию echo на функцию загрузки отображения:](Forms.HTML:TextArea.1)

<?php

class Blog extends Controller {

function index()

{

$this->load->view('blogview');

}

}

?>

[Если вы зайдете на сайт, используя URL определенный ранее, то увидите ваше новое отображение. URL должен быть таким:](Forms.HTML:TextArea.1)

[www.your-site.com/index.php/**blog**/](Forms.HTML:TextArea.1)

## [Загрузка нескольких отображений](Forms.HTML:TextArea.1)

[CodeIgniter правильно обработает несколько вызовов $this->load->view из одного контроллера. Если произойдет более одного вызова, то будут загружены все отображения. Например вы можете хранить отдельно отображение шапки, отображение меню, контента и подвала. Это может выглядеть примерно так:](Forms.HTML:TextArea.1)

[<?php  
class Page extends Controller {  
   function index()  
   {  
      $data['page\_title'] = 'Your title';  
      $this->load->view('header');  
      $this->load->view('menu');  
      $this->load->view('content', $data);  
      $this->load->view('footer');  
   }  
}  
?>](Forms.HTML:TextArea.1)

[В приведенном выше примере мы используем «динамическое добавление данных», информацию о котором вы прочтете ниже.](Forms.HTML:TextArea.1)

## [Хранение отображений в подпапках](Forms.HTML:TextArea.1)

[Ваши отображения могут быть сохранены в подпапках, если вы предпочитаете таким образом структурировать данные. При таком подходе необходимо будет включать имя папки в путь загрузки отображения. Например:](Forms.HTML:TextArea.1)

[$this->load->view('**имя\_папки**/**имя\_файла**');](Forms.HTML:TextArea.1)

## [Добавление динамических данных в ваше отображение](Forms.HTML:TextArea.1)

[Данные передаются в отображение из контроллера при помощи **массива** или **объекта** вторым параметром функции загрузки отображения. Вот пример с использованием массива:](Forms.HTML:TextArea.1)

[$data = array(  
               'title' => 'My Title',  
               'heading' => 'My Heading',  
               'message' => 'My Message'  
          );  
$this->load->view('blogview', **$data**);](Forms.HTML:TextArea.1)

[А сейчас пример с использованием объекта:](Forms.HTML:TextArea.1)

[$data = new Someclass();  
$this->load->view('blogview', **$data**);](Forms.HTML:TextArea.1)

[Замечание: Если вы используете объект, то его переменные будут превращены в массив.](Forms.HTML:TextArea.1)

[Давайте все испытаем на практике с вашим контроллером. Откройте его и добавьте следующий код:](Forms.HTML:TextArea.1)

[<?php](Forms.HTML:TextArea.1)

[class Blog extends Controller {](Forms.HTML:TextArea.1)

[function index()](Forms.HTML:TextArea.1)

[{](Forms.HTML:TextArea.1)

[$data['title'] = "My Real Title";](Forms.HTML:TextArea.1)

[$data['heading'] = "My Real Heading";](Forms.HTML:TextArea.1)

[$this->load->view('blogview', $data);](Forms.HTML:TextArea.1)

[}](Forms.HTML:TextArea.1)

[}](Forms.HTML:TextArea.1)

[?>](Forms.HTML:TextArea.1)

[Теперь откройте файл с вашим отображением и смените находящийся там текст на переменные, названия которых соответствуют ключам массива:](Forms.HTML:TextArea.1)

[<html>](Forms.HTML:TextArea.1)

[<head>](Forms.HTML:TextArea.1)

[<title><?php echo $title;?></title>](Forms.HTML:TextArea.1)

[</head>](Forms.HTML:TextArea.1)

[<body>](Forms.HTML:TextArea.1)

[<h1><?php echo $heading;?></h1>](Forms.HTML:TextArea.1)

[</body>](Forms.HTML:TextArea.1)

[</html>](Forms.HTML:TextArea.1)

[Затем загрузите вашу страницу по ранее определенному URL и обратите внимание на то, что все переменные заменены текстом.](Forms.HTML:TextArea.1)

## [Создание циклов](Forms.HTML:TextArea.1)

[Данные массива, который вы передаете в отображение, не ограничиваются простыми переменными. Вы можете передавать многомерные массивы с множеством строк. Например, если вы выбираете данные из базы данных, то они, как правило, будут представлены многомерным массивом.](Forms.HTML:TextArea.1)

[Вот простой пример. Добавьте эти строки в ваш контроллер:](Forms.HTML:TextArea.1)

[<?php](Forms.HTML:TextArea.1)

[class Blog extends Controller {](Forms.HTML:TextArea.1)

[function index()](Forms.HTML:TextArea.1)

[{](Forms.HTML:TextArea.1)

[$data['todo\_list'] = array('Clean House', 'Call Mom', 'Run Errands');](Forms.HTML:TextArea.1)

[$data['title'] = "My Real Title";](Forms.HTML:TextArea.1)

[$data['heading'] = "My Real Heading";](Forms.HTML:TextArea.1)

[$this->load->view('blogview', $data);](Forms.HTML:TextArea.1)

[}](Forms.HTML:TextArea.1)

[}](Forms.HTML:TextArea.1)

[?>](Forms.HTML:TextArea.1)

[Теперь откройте файл с вашим отображением и создайте цикл:](Forms.HTML:TextArea.1)

[<html>](Forms.HTML:TextArea.1)

[<head>](Forms.HTML:TextArea.1)

[<title><?php echo $title;?></title>](Forms.HTML:TextArea.1)

[</head>](Forms.HTML:TextArea.1)

[<body>](Forms.HTML:TextArea.1)

[<h1><?php echo $heading;?></h1>](Forms.HTML:TextArea.1)

[<h3>My Todo List</h3>](Forms.HTML:TextArea.1)

[<ul>](Forms.HTML:TextArea.1)

[<?php foreach($todo\_list as $item):?>](Forms.HTML:TextArea.1)

[<li><?php echo $item;?></li>](Forms.HTML:TextArea.1)

[<?php endforeach;?>](Forms.HTML:TextArea.1)

[</ul>](Forms.HTML:TextArea.1)

[</body>](Forms.HTML:TextArea.1)

[</html>](Forms.HTML:TextArea.1)

[**Замечание:** Заметьте, что в приведенном выше примере мы используем альтернативный синтаксис PHP. Если вы с ним не знакомы, то можете прочитать об этом [здесь](http://code-igniter.ru/user_guide/general/alternative_php.html).](Forms.HTML:TextArea.1)

# [Модели](http://code-igniter.ru/user_guide/general/alternative_php.html)

[Модели используются **опционально**, если вы хотите применять более традиционный подход к MVC.](http://code-igniter.ru/user_guide/general/alternative_php.html)

* [[Что такое модель?](http://code-igniter.ru/user_guide/general/models.html#what)](http://code-igniter.ru/user_guide/general/alternative_php.html)
* [[Устройство модели](http://code-igniter.ru/user_guide/general/models.html#anatomy)](http://code-igniter.ru/user_guide/general/models.html#what)
* [[Загрузка модели](http://code-igniter.ru/user_guide/general/models.html#loading)](http://code-igniter.ru/user_guide/general/models.html#anatomy)
* [[Авто-загрузка модели](http://code-igniter.ru/user_guide/general/models.html#auto_load_model)](http://code-igniter.ru/user_guide/general/models.html#loading)
* [[Соединение с базой данных](http://code-igniter.ru/user_guide/general/models.html#conn)](http://code-igniter.ru/user_guide/general/models.html#auto_load_model)

## [Что такое модель?](http://code-igniter.ru/user_guide/general/models.html#conn)

[Модели — это PHP классы, создаваемые для работы с информацией из вашей базы данных. Например, предположим, что вы используете CodeIgniter для ведения блога. В этом случае вам необходим класс модели с функциями для вставки, обновления и удаления записей, а так же поиска по блогу. Вот пример класса модели, который выполняет некоторые из этих действий:](http://code-igniter.ru/user_guide/general/models.html#conn)

[class Blogmodel extends Model {  
    var $title   = '';  
    var $content = '';  
    var $date    = '';  
    function Blogmodel()  
    {  
        // Call the Model constructor  
        parent::Model();  
    }  
    function get\_last\_ten\_entries()  
    {  
        $query = $this->db->get('entries', 10);  
        return $query->result();  
    }  
    function insert\_entry()  
    {  
        $this->title   = $\_POST['title'];  
        $this->content = $\_POST['content'];  
        $this->date    = time();  
        $this->db->insert('entries', $this);  
    }  
    function update\_entry()  
    {  
        $this->title   = $\_POST['title'];  
        $this->content = $\_POST['content'];  
        $this->date    = time();  
        $this->db->update('entries', $this, array('id' => $\_POST['id']));  
    }  
}](http://code-igniter.ru/user_guide/general/models.html#conn)

[Замечание: Функции, в приведенном выше примере, используют функции работы с базой данных [Active Record](http://code-igniter.ru/user_guide/database/active_record.html).](http://code-igniter.ru/user_guide/general/models.html#conn)

## [Устройство модели](http://code-igniter.ru/user_guide/database/active_record.html)

[Классы моделей хранятся в папке **application/models/**. Они могут располагаться в подпапках, если вы хотите такой организации.](http://code-igniter.ru/user_guide/database/active_record.html)

[Вот основной прототип класса модели:](http://code-igniter.ru/user_guide/database/active_record.html)

[class **Model\_name** extends Model {  
    function **Model\_name**()  
    {  
        parent::Model();  
    }  
}](http://code-igniter.ru/user_guide/database/active_record.html)

[Где **Model\_name** это имя вашего класса. Имена классов **должны** начинаться с заглавной буквы, в то время как остальные буквы должны быть в нижнем регистре. Убедитесь что ваш класс расширяет базовый класс модели.](http://code-igniter.ru/user_guide/database/active_record.html)

[Имя файла должно соответствовать имени модели, но быть целиком в нижнем регистре. Например, вот ваш класс:](http://code-igniter.ru/user_guide/database/active_record.html)

[class **User\_model** extends Model {  
    function **User\_model**()  
    {  
        parent::Model();  
    }  
}](http://code-igniter.ru/user_guide/database/active_record.html)

[Имя файла должно быть таким:](http://code-igniter.ru/user_guide/database/active_record.html)

[application/models/**user\_model.php**](http://code-igniter.ru/user_guide/database/active_record.html)

## [Загрузка модели](http://code-igniter.ru/user_guide/database/active_record.html)

[Ваши модели обычно загружаются и вызываются из [контроллера](http://code-igniter.ru/user_guide/general/controllers.html). Для загрузки моделей вы должны использовать следующую функцию:](http://code-igniter.ru/user_guide/database/active_record.html)

[$this->load->model('**Model\_name**');](http://code-igniter.ru/user_guide/general/controllers.html)

[Если ваша модель находится в под-папке, то укажите относительный путь до нее. Например, если ваша модель располагается по адресу**application/models/blog/queries.php**, то для ее загрузки используйте:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->load->model('blog/queries');](http://code-igniter.ru/user_guide/general/controllers.html)

[После загрузки вы можете получить доступ к модели, используя объект с именем, аналогичным названию модели:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->load->model('**Model\_name**');  
$this->**Model\_name**->function();](http://code-igniter.ru/user_guide/general/controllers.html)

[Если вы хотите обращаться к модели используя не ее, а другое имя, то передайте временное название в функцию загрузки вторым параметром:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->load->model('**Model\_name**', '**fubar**');  
$this->**fubar**->function();](http://code-igniter.ru/user_guide/general/controllers.html)

[Вот пример контроллера, который загружает модель, а потом отображение:](http://code-igniter.ru/user_guide/general/controllers.html)

[class Blog\_controller extends Controller {  
    function blog()  
    {  
        $this->load->model('Blog');  
        $data['query'] = $this->Blog->get\_last\_ten\_entries();  
        $this->load->view('blog', $data);  
    }  
}](http://code-igniter.ru/user_guide/general/controllers.html)

## [Авто-загрузка моделей](http://code-igniter.ru/user_guide/general/controllers.html)

[Если вы поймете, что некоторая модель нужна вам на глобальном уровне, то вы можете сообщить CodeIgniter, что эту модель необходимо загружать при инициализации системы. Вы можете это сделать открыв файл application/config/autoload.php и добавив модель в массив автозагрузки.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Соединение с базой данных](http://code-igniter.ru/user_guide/general/controllers.html)

[При загрузке модели автоматического подключения к базе **НЕ** происходит. Вам доступны следующие варианты подключения к базе данных:](http://code-igniter.ru/user_guide/general/controllers.html)

* [Вы можете соединиться используя стандартные методы базы данных, [описанные здесь](http://code-igniter.ru/user_guide/database/connecting.html) или из классов контроллера или модели.](http://code-igniter.ru/user_guide/general/controllers.html)
* [Можно передать в функцию загрузки модели третий параметр **TRUE** (boolean), который сообщит модели о том, что надо сразу подключиться. При этом будут использованы настройки соединения по умолчанию, которые определены в файле с настройками базы данных:](http://code-igniter.ru/user_guide/database/connecting.html)

[$this->load->model('**Model\_name**', '', **TRUE**);](http://code-igniter.ru/user_guide/database/connecting.html)

* [Через третий параметр так же можно передать настройки подключения, заданные вручную:](http://code-igniter.ru/user_guide/database/connecting.html)

[$config['hostname'] = "localhost";  
$config['username'] = "myusername";  
$config['password'] = "mypassword";  
$config['database'] = "mydatabase";  
$config['dbdriver'] = "mysql";  
$config['dbprefix'] = "";  
$config['pconnect'] = FALSE;  
$config['db\_debug'] = TRUE;  
$this->load->model('**Model\_name**', '', **$config**);](http://code-igniter.ru/user_guide/database/connecting.html)

# [Хелперы](http://code-igniter.ru/user_guide/database/connecting.html)

[Хелперы, как говорит их название, помогают в решении ваших задач. Каждый файл с хелперами — это простая коллекция функций в той или иной категории. Есть **URL Хелперы**, которые помогают создавать ссылки, есть **Хелперы форм**, которые помогут создать элементы форм, **Текстовые хелперы** выполняющие различные функции по форматированию текста, **Cookie Helpers** устанавливающие и читающие cookie, **Файловые хелперы**помогающие управлять файлами и так далее.](http://code-igniter.ru/user_guide/database/connecting.html)

[В отличие от большинства систем, хелперы в CodeIgniter написаны не в объектно-ориентированном формате. Это простые процедурные функции. Каждый хелпер выполняет конкретную задачу без каких-либо зависимостей от остальных хелперов.](http://code-igniter.ru/user_guide/database/connecting.html)

[CodeIgniter не загружает все файлы с хелперами по умолчанию, поэтому первым шагом для их использования является их загрузка. После загрузки хелпер становится доступным глобально в [контроллере](http://code-igniter.ru/user_guide/general/controllers.html) и [отображениях](http://code-igniter.ru/user_guide/general/views.html).](http://code-igniter.ru/user_guide/database/connecting.html)

[Хелперы, как правило, хранятся в папке **system/helpers**. Так же вы можете создать папку с названием **helpers** внутри вашей папки **application** и хранить собственные хелперы там. CodeIgniter сначала будет искать хелпер в папке **system/application/helpers**. Если директория не существует, либо файла с таким названием не найдено, CI продолжит поиск в папке **system/helpers**.](http://code-igniter.ru/user_guide/general/views.html)

## [Загрузка хелперов](http://code-igniter.ru/user_guide/general/views.html)

[Загрузить хелпер очень просто, надо только использовать следующую функцию:](http://code-igniter.ru/user_guide/general/views.html)

[$this->load->helper('**name**');](http://code-igniter.ru/user_guide/general/views.html)

[Где **name** это имя файла с хелпером без расширения .php или части «helper».](http://code-igniter.ru/user_guide/general/views.html)

[Например для загрузки файла с **URL Хелпером**, который называется **url\_helper.php**, вы можете сделать так:](http://code-igniter.ru/user_guide/general/views.html)

[$this->load->helper('**url**');](http://code-igniter.ru/user_guide/general/views.html)

[Хелпер может быть загружен в любом месте внутри вашего контроллера (или даже в отображении, хотя это не очень хорошая практика) до того момента, когда он вам понадобится. Вы можете загрузить хелперы в конструкторе вашего контроллера, если они нужны вам повсеместно, или внутри конкретной функции.](http://code-igniter.ru/user_guide/general/views.html)

[Замечание: Функция, загружающая хелперы, не возвращает никакого значения, поэтому не стоит пытаться присвоить результат ее выполнения переменной. Просто используйте ее как показано.](http://code-igniter.ru/user_guide/general/views.html)

## [Загрузка нескольких хелперов](http://code-igniter.ru/user_guide/general/views.html)

[Если вам нужно загрузить более одного хелпера, вы можете перечислить их в одном массиве, который выглядит следующим образом:](http://code-igniter.ru/user_guide/general/views.html)

[$this->load->helper( **array(**'**helper1**', '**helper2**', '**helper3**'**)** );](http://code-igniter.ru/user_guide/general/views.html)

## [Авто-загрузка хелперов](http://code-igniter.ru/user_guide/general/views.html)

[Если вам необходим хелпер для всего приложения, вы можете сказать CI, чтобы он загружал его при инициализации приложения. Это можно сделать открыв файл **application/config/autoload.php** и добавив хелпер в массив автозагрузки.](http://code-igniter.ru/user_guide/general/views.html)

## [Использование хелперов](http://code-igniter.ru/user_guide/general/views.html)

[После загрузки файла с хелперами, которые вы хотите использовать, обращайтесь к ним как к обычным PHP функциям.](http://code-igniter.ru/user_guide/general/views.html)

[Например, для того чтобы создать ссылку используя функцию **anchor()** вам надо просто написать следующее:](http://code-igniter.ru/user_guide/general/views.html)

[<?=anchor('blog/comments', 'Нажми сюда');?>](http://code-igniter.ru/user_guide/general/views.html)

[Где «Нажми сюда» это имя ссылки, а «blog/comments» это URI до контроллера/функции на которую вы хотите сослаться.](http://code-igniter.ru/user_guide/general/views.html)

## [«Расширение» хелперов](http://code-igniter.ru/user_guide/general/views.html)

[Чтобы «расширить» хелперы, создайте в вашей папке **application/helpers/** файл с идентификационным именем хелпера, но добавив в начало префикс**MY\_** (можно изменить в конфигурации. Смотри ниже.).](http://code-igniter.ru/user_guide/general/views.html)

[Если вам требуется добавить некоторую функциональность к уже существующему хелперу (добавить пару функций или изменить поведение существующей) — совершенно бессмысленно заменять весь хелпер своей версией. В этом случае легче просто «расширить» функциональность хелпера. Термин «расширение» используется не совсем верно, т.к. хелперы используют процедурный подход. Это дает вам возможность легко добавить функции в определенный хелпер или изменить существующие.](http://code-igniter.ru/user_guide/general/views.html)

[Например, чтобы расширить стандартный **Array Helper** вы должны создать файл **application/helpers/MY\_array\_helper.php** и добавить туда перекрывающие функции:](http://code-igniter.ru/user_guide/general/views.html)

[// Функции any\_in\_array() нет в Array Helper, поэтому вы просто создадите новую функцию  
function any\_in\_array($needle, $haystack)  
{  
    $needle = (is\_array($needle)) ? $needle : array($needle);  
    foreach ($needle as $item)  
    {  
        if (in\_array($item, $haystack))  
        {  
            return TRUE;  
        }  
    }  
    return FALSE;  
}  
// Функция random\_element() есть в Array Helper, поэтому объявив ее вы перекроете стандартную функцию  
function random\_element($array)  
{  
    shuffle($array);  
    return array\_pop();  
}](http://code-igniter.ru/user_guide/general/views.html)

### [Установка собственного префикса](http://code-igniter.ru/user_guide/general/views.html)

[Имя префикса для расширения хелперов так же используется для расширения библиотек и классов ядра. Чтобы установить свой префикс, откройте ваш файл **application/config/config.php** и найдите там такую строчку:](http://code-igniter.ru/user_guide/general/views.html)

[$config['subclass\_prefix'] = 'MY\_';](http://code-igniter.ru/user_guide/general/views.html)

[Пожалуйста обратите внимание на то, что стандартный префикс для самого CodeIgniter — **CI\_**. Вам нельзя использовать такой префикс.](http://code-igniter.ru/user_guide/general/views.html)

## [Что теперь?](http://code-igniter.ru/user_guide/general/views.html)

[В содержании вы найдете весь список существующих хелперов с описанием всех функций. Можете пользоваться этим руководством ежедневно.](http://code-igniter.ru/user_guide/general/views.html)

# [Плагины](http://code-igniter.ru/user_guide/general/views.html)

[Плагины работают практически идентично [хелперам](http://code-igniter.ru/user_guide/general/helpers.html). Главное отличие заключается в том, что плагин — это одна функция, в то время как хелпер — это их набор. Хелперы так же считаются частью ядра системы; плагины должны быть работоспособными сразу для нескольких систем.](http://code-igniter.ru/user_guide/general/views.html)

[Плагины необходимо хранить в директории **system/plugins**, или вы можете создать папку **plugins** внутри вашей папки **application** и хранить плагины там. CodeIgniter изначально будет искать их внутри **system/application/plugins**. Если директория не существует, либо там нет указанного плагина, CI будет искать этот плагин в глобальной директории **system/plugins**.](http://code-igniter.ru/user_guide/general/helpers.html)

## [Загрузка плагина](http://code-igniter.ru/user_guide/general/helpers.html)

[Загрузка файла плагина осуществляется очень просто с использованием этой функции:](http://code-igniter.ru/user_guide/general/helpers.html)

[$this->load->plugin('**name**');](http://code-igniter.ru/user_guide/general/helpers.html)

[Где **name** это название файла с плагином без расширения .php и части «plugin».](http://code-igniter.ru/user_guide/general/helpers.html)

[Например, чтобы загрузить плагин **Captcha**, файл которого называется **captcha\_pi.php**, вам надо написать это:](http://code-igniter.ru/user_guide/general/helpers.html)

[$this->load->plugin('**captcha**');](http://code-igniter.ru/user_guide/general/helpers.html)

[Плагин может быть загружен из любого места в вашем [контроллере](http://code-igniter.ru/user_guide/general/controllers.html) (или даже в [отображении](http://code-igniter.ru/user_guide/general/views.html), хотя это плохая практика) до того момента, как вам понадобится его функционал. Вы можете загрузить плагин в конструкторе контроллера, чтобы он был доступен везде, либо в конкретной функции.](http://code-igniter.ru/user_guide/general/helpers.html)

[Замечание: Функция загрузки плагина не возвращает значений, поэтому не стоит пытаться присвоить результат ее выполнения переменной. Пользуйтесь ей как показано.](http://code-igniter.ru/user_guide/general/views.html)

## [Загрузка нескольких плагинов](http://code-igniter.ru/user_guide/general/views.html)

[Если вам необходимо загрузить сразу несколько плагинов, то вы можете перечислить их в массиве:](http://code-igniter.ru/user_guide/general/views.html)

[$this->load->plugin( **array(**'**plugin1**', '**plugin2**', '**plugin3**'**)** );](http://code-igniter.ru/user_guide/general/views.html)

## [Авто-загрузка плагинов](http://code-igniter.ru/user_guide/general/views.html)

[Если вы обнаружите, что вам необходим плагин в глобальном масштабе, то лучше заставить его загрузиться при инициализации системы. Чтобы сделать это, откройте ваш файл **application/config/autoload.php** и добавьте плагин в массив автозагрузки.](http://code-igniter.ru/user_guide/general/views.html)

## [Использование плагина](http://code-igniter.ru/user_guide/general/views.html)

[После загрузки плагина, вы можете обращаться к нему, как к стандартной PHP функции.](http://code-igniter.ru/user_guide/general/views.html)

# [Использование библиотек CodeIgniter](http://code-igniter.ru/user_guide/general/views.html)

[Все доступные библиотеки находятся в папке **system/libraries**. В большинстве случаев для использования этих классов необходимо их инициализовать в[контроллере](http://code-igniter.ru/user_guide/general/controllers.html), используя следующую функцию:](http://code-igniter.ru/user_guide/general/views.html)

[$this->load->library('**class name**');](http://code-igniter.ru/user_guide/general/controllers.html)

[Где **class name** это название файла с классом. Например, для загрузки класса валидации вам необходимо написать:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->load->library('**validation**');](http://code-igniter.ru/user_guide/general/controllers.html)

[После загрузки, вы можете использовать его как указано на соответствующей странице с описанием этого класса.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Создание собственных библиотек](http://code-igniter.ru/user_guide/general/controllers.html)

[Пожалуйста прочтите раздел руководства, в котором описано [как создавать свои библиотеки](http://code-igniter.ru/user_guide/general/creating_libraries.html).](http://code-igniter.ru/user_guide/general/controllers.html)

# [Создание библиотек](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[Когда мы используем термин «библиотеки», то, как правило, ссылаемся на классы, которые лежат в папке **libraries** и описаны в разделе «Классы» этого руководства. Однако сейчас мы будем говорить о том как создавать собственные классы в каталоге **application/libraries** чтобы сохранить разделение между локальными ресурсами и глобальными ресурсами фреймворка.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[В качестве бонуса, CodeIgniter позволяет вам **расширять** родные классы фреймворка, если вам просто необходимо добавить какую-то функциональность. Или вы можете полностью заменить родную библиотеку, просто разместив свою с идентичным названием в папке **application/libraries**.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[Кратко:](http://code-igniter.ru/user_guide/general/creating_libraries.html)

* [Вы можете создавать совершенно новые библиотеки.](http://code-igniter.ru/user_guide/general/creating_libraries.html)
* [Вы можете расширять функционал родных библиотек.](http://code-igniter.ru/user_guide/general/creating_libraries.html)
* [Вы можете полностью заменять родную библиотеку.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[На этой странице объясняются детально эти три понятия.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[**Замечание:** Класс Database не может быть расширен или заменен, а так же для PHP 4 нельзя заменять класс Loader. Все прочие классы можно свободно расширять/заменять.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

## [Хранение](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[Ваши библиотеки классов должны быть расположены в директории **application/libraries**, CodeIgniter будет искать их там во время инициализации.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

## [Правила именования](http://code-igniter.ru/user_guide/general/creating_libraries.html)

* [Имена файлов с большой буквы. Например:  **Myclass.php**](http://code-igniter.ru/user_guide/general/creating_libraries.html)
* [Имя класса с большой буквы. Например:  **class Myclass**](http://code-igniter.ru/user_guide/general/creating_libraries.html)
* [Имя файла и имя класса должны совпадать.](http://code-igniter.ru/user_guide/general/creating_libraries.html)

## [Файл класса](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[Классы должны следовать этому прототипу (Замечание: Мы называем класс **Someclass** только в качестве примера):](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[<?php if (!defined('BASEPATH')) exit('Нет доступа к скрипту');   
class Someclass {  
    function some\_function()  
    {  
    }  
}  
?>](http://code-igniter.ru/user_guide/general/creating_libraries.html)

## [Использование ваших классов](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[Из любой функции вашего [контроллера](http://code-igniter.ru/user_guide/general/controllers.html) можно инициализировать библиотеку следующей стандартной функцией:](http://code-igniter.ru/user_guide/general/creating_libraries.html)

[$this->load->library('**someclass**');](http://code-igniter.ru/user_guide/general/controllers.html)

[Где someclass это имя файла без расширения «.php». Вы можете указать имя файла в любом регистре.](http://code-igniter.ru/user_guide/general/controllers.html)

[После загрузки класса, обращайтесь к нему используя **нижний регистр**:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->**someclass**->some\_function(); // Имя объекта всегда в нижнем регистре](http://code-igniter.ru/user_guide/general/controllers.html)

## [Передача параметров при инициализации вашего класса](http://code-igniter.ru/user_guide/general/controllers.html)

[В функцию загрузки класса можно передать второй параметр, он будет передан в ваш класс:](http://code-igniter.ru/user_guide/general/controllers.html)

[$params = array('type' => 'large', 'color' => 'red');  
$this->load->library('Someclass', **$params**);](http://code-igniter.ru/user_guide/general/controllers.html)

[Если вы используете эту возможность, то вам необходимо настроить в конструкторе класса ожидание переменных:](http://code-igniter.ru/user_guide/general/controllers.html)

[<?php if (!defined('BASEPATH')) exit('Нет доступа к скрипту');  
class Someclass {  
    function Someclass($params)  
    {  
        // Какие-то операции с $params  
    }  
}  
?>](http://code-igniter.ru/user_guide/general/controllers.html)

[Вы так же можете передавать параметры, хранящиеся в конфигурационном файле. Просто создайте файл с именем, аналогичным **имени файла** с классом и сохраните в вашей папке **application/config/**. Заметим, что если вы передаете параметры динамически, то конфигурационный файл будет недоступен.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Использование ресурсов CodeIgniter в вашей библиотеке](http://code-igniter.ru/user_guide/general/controllers.html)

[Для доступа к родным ресурсам CodeIgniter в вашей библиотеке используется функция **get\_instance()**. Эта функция вернет супер-объект CodeIgniter.](http://code-igniter.ru/user_guide/general/controllers.html)

[При помощи этого вы сможете обращаться к любой родной функции при помощи конструкции **$this**:](http://code-igniter.ru/user_guide/general/controllers.html)

[**$this**->load->helper('url');  
**$this**->load->library('session');  
**$this**->config->item('base\_url');  
и т.п.](http://code-igniter.ru/user_guide/general/controllers.html)

[**$this**, однако, работает только в вашем контроллере, модели или отображении. Если вы хотите использовать родные классы CI внутри ваших собственных, то можете сделать это следующим образом:](http://code-igniter.ru/user_guide/general/controllers.html)

[Во первых присвоить указатель на родной объект переменной:](http://code-igniter.ru/user_guide/general/controllers.html)

[$CI =& get\_instance();](http://code-igniter.ru/user_guide/general/controllers.html)

[После того как вы присвоили объект переменной — будете использовать эту переменную вместо **$this**:](http://code-igniter.ru/user_guide/general/controllers.html)

[$CI =& get\_instance();  
$CI->load->helper('url');  
$CI->load->library('session');  
$CI->config->item('base\_url');  
и т.п.](http://code-igniter.ru/user_guide/general/controllers.html)

[**Замечание:** Заметьте, что функция get\_instance() передается по ссылке:   
  
**$CI =& get\_instance();**   
  
**Это очень важно.** Указатель позволяет использовать оригинальный объект CodeIgniter, а не создавать его копию.   
  
**Кроме того имейте ввиду:** Если вы работаете с PHP 4, то лучше избегать вызова **get\_instance()** внутри вашего конструктора класса. PHP 4 имеет проблемы с созданием ссылок на супер-объект CI в контроллере, т.к. объекты не существуют до тех пор, пока класс полностью не обработан.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Замена родных библиотек вашими версиями](http://code-igniter.ru/user_guide/general/controllers.html)

[Вам достаточно назвать собственный класс таким же именем, как и родной, и CI загрузит его, вместо родного. Например, чтобы заменить родной класс**Email**, вы можете создать файл **application/libraries/Email.php** и создать класс внутри него:](http://code-igniter.ru/user_guide/general/controllers.html)

[class CI\_Email {  
}](http://code-igniter.ru/user_guide/general/controllers.html)

[Заметим, что большинство родных классов имеют префикс **CI\_**.](http://code-igniter.ru/user_guide/general/controllers.html)

[Для загрузки вашей библиотеки используйте родную функцию загрузки:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->load->library('**email**');](http://code-igniter.ru/user_guide/general/controllers.html)

[**Замечание:** На данный момент класс Database не может быть заменен на вашу версию.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Расширение родных библиотек](http://code-igniter.ru/user_guide/general/controllers.html)

[Если вам надо расширить функциональность библиотеки, возможно добавить одну-две функции, то вам стоит не переопределять всю библиотеку, а просто расширить ее функциональность. Расширение класса практически идентично его замене за парой исключений:](http://code-igniter.ru/user_guide/general/controllers.html)

* [Объявление класса должно расширять родительский класс.](http://code-igniter.ru/user_guide/general/controllers.html)
* [Новое имя класса должно начинаться с префикса **MY\_** (это можно настроить. Смотри ниже).](http://code-igniter.ru/user_guide/general/controllers.html)

[Например, чтобы расширить родной класс **Email**, вы должны создать файл **application/libraries/MY\_Email.php** и объявить внутри класс:](http://code-igniter.ru/user_guide/general/controllers.html)

[class MY\_Email extends CI\_Email {  
}](http://code-igniter.ru/user_guide/general/controllers.html)

[Замечание: Если вам в вашем классе необходимо использовать конструктор, то не забудьте вызвать в нем родительский конструктор:](http://code-igniter.ru/user_guide/general/controllers.html)

[class MY\_Email extends CI\_Email {  
    function My\_Email()  
    {  
        parent::CI\_Email();  
    }  
}](http://code-igniter.ru/user_guide/general/controllers.html)

### [Загрузка вашего подкласса](http://code-igniter.ru/user_guide/general/controllers.html)

[Для загрузки собственного подкласса, как правило, используется стандартный синтаксис. НЕ указывайте здесь ваш префикс. Так, например, для класса Email, который расширяет стандартный одноименный класс, вы должны использовать:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->load->library('**email**');](http://code-igniter.ru/user_guide/general/controllers.html)

[После загрузки класса, вы будете использовать переменные класса, как при обычном его расширении. В случае с описанным примером все вызовы будут иметь следующий синтаксис:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->**email**->some\_function();](http://code-igniter.ru/user_guide/general/controllers.html)

### [Установка собственного префикса](http://code-igniter.ru/user_guide/general/controllers.html)

[Для установки собственного подклассового префикса откройте файл **application/config/config.php** и напишите там следующее:](http://code-igniter.ru/user_guide/general/controllers.html)

[$config['subclass\_prefix'] = 'MY\_';](http://code-igniter.ru/user_guide/general/controllers.html)

[Пожалуйста учтите, что стандартным для CodeIgniter является префикс **CI\_**. Вам НЕЛЬЗЯ использовать его.](http://code-igniter.ru/user_guide/general/controllers.html)

# [Создание классов ядра](http://code-igniter.ru/user_guide/general/controllers.html)

[При каждом запуске CodeIgniter запускается несколько его базовых классов, которые инициализируются автоматически при загрузке системы. Это части ядра фреймворка. Хотя вы можете заменить любой из системных классов или просто расширить их.](http://code-igniter.ru/user_guide/general/controllers.html)

[**У большинства пользователей никогда не возникнет необходимость менять классы ядра, но сама возможность сделать это делает CodeIgniter очень гибким фреймворком.**](http://code-igniter.ru/user_guide/general/controllers.html)

[**Замечание:**  Смешение ваших наработок со скриптами ядра, как правило, имеет множество последствий, так что подумайте дважды перед тем как сделаете это.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Перечень системных классов](http://code-igniter.ru/user_guide/general/controllers.html)

[Вот перечень основных системных файлов, которые вызываются при инициализации CodeIgniter:](http://code-igniter.ru/user_guide/general/controllers.html)

* [Benchmark](http://code-igniter.ru/user_guide/general/controllers.html)
* [Config](http://code-igniter.ru/user_guide/general/controllers.html)
* [Controller](http://code-igniter.ru/user_guide/general/controllers.html)
* [Exceptions](http://code-igniter.ru/user_guide/general/controllers.html)
* [Hooks](http://code-igniter.ru/user_guide/general/controllers.html)
* [Input](http://code-igniter.ru/user_guide/general/controllers.html)
* [Language](http://code-igniter.ru/user_guide/general/controllers.html)
* [Loader](http://code-igniter.ru/user_guide/general/controllers.html)
* [Log](http://code-igniter.ru/user_guide/general/controllers.html)
* [Output](http://code-igniter.ru/user_guide/general/controllers.html)
* [Router](http://code-igniter.ru/user_guide/general/controllers.html)
* [URI](http://code-igniter.ru/user_guide/general/controllers.html)

## [Замена классов ядра](http://code-igniter.ru/user_guide/general/controllers.html)

[Для использования ваших скриптов вместо скриптов по умолчанию просто разместите их в вашей папке **application/libraries**:](http://code-igniter.ru/user_guide/general/controllers.html)

[application/libraries/**some-class.php**](http://code-igniter.ru/user_guide/general/controllers.html)

[Если такой папки нет — создайте ее.](http://code-igniter.ru/user_guide/general/controllers.html)

[Любое имя файла, идентичное одному из перечисленных выше, будет использоваться вместо настоящего файла из поставки CodeIgnier.](http://code-igniter.ru/user_guide/general/controllers.html)

[Пожалуйста учтите, что ваш класс должен использовать **CI** в качестве префикса. Например, если ваш файл назван **Input.php**, то класс должен объявляться так:](http://code-igniter.ru/user_guide/general/controllers.html)

[class CI\_Input {  
}](http://code-igniter.ru/user_guide/general/controllers.html)

## [Расширение классов ядра](http://code-igniter.ru/user_guide/general/controllers.html)

[Если вам надо расширить функциональность библиотеки, возможно добавить одну-две функции, то вам стоит не переопределять всю библиотеку, а просто расширить ее функциональность. Расширение класса практически идентично его замене за парой исключений:](http://code-igniter.ru/user_guide/general/controllers.html)

* [Объявление класса должно расширять родительский класс.](http://code-igniter.ru/user_guide/general/controllers.html)
* [Новое имя класса должно начинаться с префикса **MY\_** (это можно настроить. Смотри ниже).](http://code-igniter.ru/user_guide/general/controllers.html)

[Например, чтобы расширить родной класс **Input**, вы должны создать файл **application/libraries/MY\_Input.php** и объявить внутри класс:](http://code-igniter.ru/user_guide/general/controllers.html)

[class MY\_Input extends CI\_Input {  
}](http://code-igniter.ru/user_guide/general/controllers.html)

[Замечание: Если вам в классе необходим конструктор — не забудьте в нем вызвать конструктор родителя:](http://code-igniter.ru/user_guide/general/controllers.html)

[class MY\_Input extends CI\_Input {  
    function My\_Input()  
    {  
        parent::CI\_Input();  
    }  
}](http://code-igniter.ru/user_guide/general/controllers.html)

[**Подсказка:**  Любая функция в вашем классе, название которой аналогично функции в родительском классе будет использоваться вместо родной (это называется «метод переопределения»). Это позволяет существенно менять ядро CodeIgniter'а.](http://code-igniter.ru/user_guide/general/controllers.html)

### [Установка собственного префикса](http://code-igniter.ru/user_guide/general/controllers.html)

[Для установки собственного префикса откройте файл **application/config/config.php** и найдите там:](http://code-igniter.ru/user_guide/general/controllers.html)

[$config['subclass\_prefix'] = 'MY\_';](http://code-igniter.ru/user_guide/general/controllers.html)

[Пожалуйста заметьте, что CodeIgniter использует префикс **CI\_**, поэтому вам НЕЛЬЗЯ его использовать.](http://code-igniter.ru/user_guide/general/controllers.html)

# [Хуки — Расширение ядра](http://code-igniter.ru/user_guide/general/controllers.html)

[Технология хуков в CI позволяет вам изменять ход работы фреймворка без изменения файлов ядра. Когда CodeIgniter запускается, происходят вызовы, нарисованные на схеме на странице [Ход выполнения приложения](http://code-igniter.ru/user_guide/overview/appflow.html). Однако, может возникнуть такая ситуация что вы захотите что-то изменить в этой цепочке на этапе ее выполнения. Например, вы можете захотеть вызвать какой-то скрипт прямо перед загрузкой контроллера или сразу после загрузки.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Подключение хуков](http://code-igniter.ru/user_guide/overview/appflow.html)

[Система хуков может быть включена/выключена глобально, просто в файле **application/config/config.php** необходимо найти строку:](http://code-igniter.ru/user_guide/overview/appflow.html)

[$config['enable\_hooks'] = TRUE;](http://code-igniter.ru/user_guide/overview/appflow.html)

## [Объявление хуков](http://code-igniter.ru/user_guide/overview/appflow.html)

[Хуки создаются в файле **application/config/hooks.php**. Каждый хук определяется как массив по этому шаблону:](http://code-igniter.ru/user_guide/overview/appflow.html)

[$hook['pre\_controller'] = array(  
                                'class'    => 'MyClass',  
                                'function' => 'Myfunction',  
                                'filename' => 'Myclass.php',  
                                'filepath' => 'hooks',  
                                'params'   => array('beer', 'wine', 'snacks')  
                                );](http://code-igniter.ru/user_guide/overview/appflow.html)

[**Замечания:**  
Индекс массива говорит о том, каким хуком вы хотите воспользоваться. В приведенном выше примере устанавливается хук типа **pre\_controller**. Список допустимых хуков находится ниже. Следующие пункты должны быть определены в ваших массивах:](http://code-igniter.ru/user_guide/overview/appflow.html)

* [**class**  Название класса, который бы вы хотели использовать. Если вы предпочитаете использовать функцию, а не класс, то оставьте этот пункт пустым.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**function**  Имя функции, которую необходимо вызвать.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**filename**  Имя файла, содержащее эту функцию/класс.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**filepath**  Имя папки, содержащей файл. Замечание: Ваш скрипт обязательно должен находиться ВНУТРИ папки **application**, поэтому путь надо указать относительно этой папки. Например, если ваш скрипт расположен в **application/hooks**, можно написать просто **hooks** в этом элементе массива. Если скрипт лежит в **application/hooks/utilities**, то необходимо указать здесь **hooks/utilities**. Без слешей по краям.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**params**  Любые параметры, которые вы хотите передать в скрипт. Этот пункт опционален.](http://code-igniter.ru/user_guide/overview/appflow.html)

## [Несколько вызовов хука одного типа](http://code-igniter.ru/user_guide/overview/appflow.html)

[Если хотите несколько раз использовать хуки одного типа — просто сделайте ваш массив многомерным:](http://code-igniter.ru/user_guide/overview/appflow.html)

[$hook['pre\_controller']**[]** = array(  
                                'class'    => 'MyClass',  
                                'function' => 'Myfunction',  
                                'filename' => 'Myclass.php',  
                                'filepath' => 'hooks',  
                                'params'   => array('beer', 'wine', 'snacks')  
                                );  
  
$hook['pre\_controller']**[]** = array(  
                                'class'    => 'MyOtherClass',  
                                'function' => 'MyOtherfunction',  
                                'filename' => 'Myotherclass.php',  
                                'filepath' => 'hooks',  
                                'params'   => array('red', 'yellow', 'blue')  
                                );](http://code-igniter.ru/user_guide/overview/appflow.html)

[В этом помогут пустые скобки после ключа массива:](http://code-igniter.ru/user_guide/overview/appflow.html)

[$hook['pre\_controller']**[]**](http://code-igniter.ru/user_guide/overview/appflow.html)

[Это позволит использовать сразу несколько хуков последовательно. Они будут вызваны в таком порядке, в каком вы их объявили.](http://code-igniter.ru/user_guide/overview/appflow.html)

## [Точки хуков](http://code-igniter.ru/user_guide/overview/appflow.html)

[Ниже приведен список доступных типов хуков](http://code-igniter.ru/user_guide/overview/appflow.html)

* [**pre\_system**  
  Вызывается при самом старте системы. Только базовый класс и класс хуков загружены на этот момент. Ни роутинга, ни других процессов маршрутизации еще не произошло.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**pre\_controller**  
  Вызывается непосредственно перед вызовом ваших контроллеров. Все базовые классы, роутинг и безопасность уже выполнились.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**post\_controller\_constructor**  
  Вызывается сразу после обработки контроллера, но до вызова каких-либо функций внутри него.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**post\_controller**  
  Вызывается после полного выполнения контроллера.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**display\_override**  
  Заменяет функцию **\_display()** которая используется для отправки браузеру результата работы контроллера. Это позволяет использовать вам собственные методы для отображения. Заметим, что окончательные данные можно будет получить вызвав **$this->output->get\_output()**.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**cache\_override**  
  Позволяет использовать собственную функцию, вместо функции **\_display\_cache()** выводящего класса. Это позволяет вам использовать собственный механизм кеширования.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**scaffolding\_override**  
  Вызывается при запросе отображения скаффолдинга, позволяя вам заменить его.](http://code-igniter.ru/user_guide/overview/appflow.html)
* [**post\_system**  
  Вызывается после окончательного сбора страницы. После выполнения готовые данные отправляются в браузер.](http://code-igniter.ru/user_guide/overview/appflow.html)

[Автозагрузка ресурсов](http://code-igniter.ru/user_guide/overview/appflow.html)

[CodeIgniter поставляется с возможностью автозагрузки необходимых библиотек, хелперов и плагинов при инициализации системы. Если вам необходим какой-то ресурс в масштабах всего приложения, то вам стоит рассмотреть возможность его автозагрузки.](http://code-igniter.ru/user_guide/overview/appflow.html)

[Следующие вещи могут загружаться автоматически:](http://code-igniter.ru/user_guide/overview/appflow.html)

* [Классы ядра в папке «libraries»](http://code-igniter.ru/user_guide/overview/appflow.html)
* [Хелперы, находящиеся в папке «helpers»](http://code-igniter.ru/user_guide/overview/appflow.html)
* [Плагины, расположенные в «plugins»](http://code-igniter.ru/user_guide/overview/appflow.html)
* [Любые конфигурационные файлы из папки «config»](http://code-igniter.ru/user_guide/overview/appflow.html)
* [Языковые файлы из папки «system/language»](http://code-igniter.ru/user_guide/overview/appflow.html)
* [Модели, расположенные в «models»](http://code-igniter.ru/user_guide/overview/appflow.html)

[Для автозагрузки ресурсов, откройте файл **application/config/autoload.php** и перечислите те элементы, которые вы хотели бы загружать автоматически, в массиве **autoload**. Вы найдете инструкции о том как это сделать внутри этого файла.](http://code-igniter.ru/user_guide/overview/appflow.html)

[**Замечание:** Не указывайте расширение (.php) когда добавляете файл в массив автозагрузки.](http://code-igniter.ru/user_guide/overview/appflow.html)

# [Общие функции](http://code-igniter.ru/user_guide/overview/appflow.html)

[CodeIgniter использует несколько функций, определенных глобально и доступных из любой точки кода без предварительной загрузки/настройки. Они не требуют никаких сторонных библиотек или хелперов.](http://code-igniter.ru/user_guide/overview/appflow.html)

## [is\_really\_writable('путь\к\файлу')](http://code-igniter.ru/user_guide/overview/appflow.html)

[Функция is\_writable() возвращает TRUE на серверах Windows если Вы можете производить запись в файл, или FALSE если файл доступен только для чтения. Тогда следует перед записью изменить права на файл.](http://code-igniter.ru/user_guide/overview/appflow.html)

[if (is\_really\_writable('file.txt'))  
{  
     echo "Файл доступен для записи ";  
}  
else  
{  
     echo "Только для чтения ";  
}](http://code-igniter.ru/user_guide/overview/appflow.html)

## [config\_item('item\_key')](http://code-igniter.ru/user_guide/overview/appflow.html)

[[Библиотека Config](http://code-igniter.ru/user_guide/libraries/config.html) предназначена для доступа к конфигурационным данным, но можно обойтись и без нее функцией config\_item(), возвращающей запись с конф. файла. Более подробно смотрите в описании библиотеки.](http://code-igniter.ru/user_guide/overview/appflow.html)

## [show\_error('message'), show\_404('page'), log\_message('level', 'message')](http://code-igniter.ru/user_guide/libraries/config.html)

[Функции для вывода сообщений об ошибках. См. [Обработка ошибок](http://code-igniter.ru/user_guide/general/errors.html)](http://code-igniter.ru/user_guide/libraries/config.html)

# [Скаффолдинг](http://code-igniter.ru/user_guide/general/errors.html)

[Скаффолдинг в CodeIgniter позволяет быстро и очень удобно добавлять, изменять или удалять информацию из базы данных в ходе разработки приложения.](http://code-igniter.ru/user_guide/general/errors.html)

[**Очень важно:** Скаффолдинг предназначен только для использования в ходе разработки. Он не безопасен, за исключением случая с использованием секретного слова в его названии. Каждый, кто имеет к нему доступ, может свободно модифицировать информацию в вашей базе данных. Если вы использовали скаффолдинг, то убедитесь, что не забыли отключить его. НЕ оставляйте его активным на работающем сайте. И, пожалуйста, переопределите секретное слово перед использованием.](http://code-igniter.ru/user_guide/general/errors.html)

## [Почему кто-то использует скаффолдинг?](http://code-igniter.ru/user_guide/general/errors.html)

[Вот типичный сценарий: Вы создаете новую таблицу в базе данных в ходе разработки, и хотите сразу внести в нее некоторые данные. Без скаффолдинга вы можете либо написать несколько запросов в командной строке, либо использовать какую-нибудь СУБД, вроде phpMyAdmin. А когда закончите работу — сможете легко удалить эти данные.](http://code-igniter.ru/user_guide/general/errors.html)

## [Установка секретного слова](http://code-igniter.ru/user_guide/general/errors.html)

[Прежде чем подключить скаффолдинг, пожалуйста, переопределите секретное слово. Это слово, при вводе которого в URL можно получить доступ к скаффолдингу, поэтому постарайтесь поставить его таким, чтобы никто не догадался.](http://code-igniter.ru/user_guide/general/errors.html)

[Для установки секретного слова откройте ваш файл **application/config/routes.php** и найдите там такую строку:](http://code-igniter.ru/user_guide/general/errors.html)

[$route['scaffolding\_trigger'] = '';](http://code-igniter.ru/user_guide/general/errors.html)

[Смените пустую строку на свое уникальное слово.](http://code-igniter.ru/user_guide/general/errors.html)

[**Замечание:** Секретное слово скаффолдинга **не** может начинаться с подчеркивания.](http://code-igniter.ru/user_guide/general/errors.html)

## [Включение скаффолдинга](http://code-igniter.ru/user_guide/general/errors.html)

[Замечание: Информация далее предполагает, что вы знаете как работают [контроллеры](http://code-igniter.ru/user_guide/general/controllers.html), и у вас уже есть хотя бы один. Так же предполагается, что вы уже настроили подключение к [базе данных](http://code-igniter.ru/user_guide/database/index.html). Если нет, то информация здесь не будет для вас актуальной и вам предлагается ознакомиться с этими разделами в первую очередь. Наконец, полагается, что вы понимаете, что такое конструктор класса. Если нет, то читайте об этом с последней части главы о[контроллерах](http://code-igniter.ru/user_guide/general/controllers.html).](http://code-igniter.ru/user_guide/general/errors.html)

[Для подключения скаффолдинга, вы должны инициализовать его в свой контроллер:](http://code-igniter.ru/user_guide/general/controllers.html)

[<?php  
class Blog extends Controller {  
       function Blog()  
       {  
            parent::Controller();  
            **$this->load->scaffolding('table\_name'**);  
       }  
}  
?>](http://code-igniter.ru/user_guide/general/controllers.html)

[Где **table\_name** это имя таблицы (таблицы, не базы данных) с которой вы бы хотели работать.](http://code-igniter.ru/user_guide/general/controllers.html)

[После того как вы инициализируете скаффолдинг, он будет доступен вам по адресу, похожему на этот:](http://code-igniter.ru/user_guide/general/controllers.html)

[www.your-site.com/index.php/**class**/**секретное\_слово**/](http://code-igniter.ru/user_guide/general/controllers.html)

[Например, при имени контроллера **Blog** и секретном слове **abracadabra** скаффолдинг будет доступен по адресу:](http://code-igniter.ru/user_guide/general/controllers.html)

[www.your-site.com/index.php/**blog**/**abracadabra**/](http://code-igniter.ru/user_guide/general/controllers.html)

[Скаффолдинг имеет свой интерфейс. Вы можете добавлять, изменять и удалять записи в таблице базы данных.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Последнее замечание:](http://code-igniter.ru/user_guide/general/controllers.html)

[Скаффолдинг будет работать только с теми таблицами, в которых определен PRIMARY KEY, т.к. это необходимо для работы с большинством функций базы данных.](http://code-igniter.ru/user_guide/general/controllers.html)

# [URI Роутинг](http://code-igniter.ru/user_guide/general/controllers.html)

[Как правило есть соответствие один-к-одному между строкой URL и соответствующим контроллером/методом. Сегменты URL обычно формируются по этому шаблону:](http://code-igniter.ru/user_guide/general/controllers.html)

[www.your-site.com/**class**/**function**/**id**/](http://code-igniter.ru/user_guide/general/controllers.html)

[Однако, в некоторых случаях, вы можете перенаправить адрес так, чтобы строка вида контроллер/метод указывала на совершенно другие места в приложении.](http://code-igniter.ru/user_guide/general/controllers.html)

[Например, вы хотите чтобы ваши адреса соответствовали этому шаблону:](http://code-igniter.ru/user_guide/general/controllers.html)

[www.your-site.com/product/1/  
www.your-site.com/product/2/  
www.your-site.com/product/3/  
www.your-site.com/product/4/](http://code-igniter.ru/user_guide/general/controllers.html)

[Как правило второй сегмент адресной строки резервируется для имени функции, но в примере выше мы хотим использовать вместо него ID. Чтобы сделать это, необходимо перенаправить адрес на обработчик URL CodeIgniter.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Установка правил собственной маршрутизации](http://code-igniter.ru/user_guide/general/controllers.html)

[Правила маршрутизации объявляются в вашем файле **application/config/routes.php**. В нем вы увидите массив с названием **$route**, при помощи которого вы сможете переопределить адреса. Роуты могут быть определены двумя способами: **маской** или **Регулярными Выражениями**](http://code-igniter.ru/user_guide/general/controllers.html)

## [Маски](http://code-igniter.ru/user_guide/general/controllers.html)

[Обычная маска маршрута может выглядеть примерно так:](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['product/:num'] = "catalog/product\_lookup";](http://code-igniter.ru/user_guide/general/controllers.html)

[В роуте ключ массива содержит URI в котором будет искаться соответствие, а в значении содержится путь, куда необходимо переадресовать обработчик. Если в приведенном выше примере слово «product» находится в первом сегменте, а номер во втором сегменте, то вместо этого адреса будет произведено обращение к классу «catalog» и методу «product\_lookup».](http://code-igniter.ru/user_guide/general/controllers.html)

[Вы можете использовать буквы или два типа заполнителя:](http://code-igniter.ru/user_guide/general/controllers.html)

[:num  
:any](http://code-igniter.ru/user_guide/general/controllers.html)

[**:num** будет соответствовать сегмент, содержащий только цифры.  
**:any** будет соответствовать сегмент с любыми символами.](http://code-igniter.ru/user_guide/general/controllers.html)

[**Замечание:** Роуты будут работать в том порядке, в котором они определены. Ранее определенные всегда будут преобладать над определенными позже.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Примеры](http://code-igniter.ru/user_guide/general/controllers.html)

[Вот несколько примеров маршрутизации:](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['journals'] = "blogs";](http://code-igniter.ru/user_guide/general/controllers.html)

[URL содержащий слово «journals» в первом сегменте будет переадресован к классу «blogs».](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['blog/joe'] = "blogs/users/34";](http://code-igniter.ru/user_guide/general/controllers.html)

[URL содержащий сегменты «blog/joe» будет переадресован к классу «blogs» и методу «users». ID будет установлен в значение «34».](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['product/:any'] = "catalog/product\_lookup";](http://code-igniter.ru/user_guide/general/controllers.html)

[URL со словом «product» в первом сегменте, и любым другим во втором будет перенаправлен к классу «catalog», методу «product\_lookup».](http://code-igniter.ru/user_guide/general/controllers.html)

[**Важно:** Не используйте начинающих/конечных слешей.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Регулярные Выражения](http://code-igniter.ru/user_guide/general/controllers.html)

[При желании вы можете использовать Регулярные Выражения для определения правил роутинга. Допускается любое правильное регулярное выражение, как и обратные ссылки.](http://code-igniter.ru/user_guide/general/controllers.html)

[**Замечание:**  Если вы используете обратные ссылки, то должны пользоваться синтаксисом доллара и затем двойной бэкслеш.](http://code-igniter.ru/user_guide/general/controllers.html)

[Типичный RegEx роут должен выглядеть так:](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['products/([a-z]+)/(\d+)'] = "$1/id\_$2";](http://code-igniter.ru/user_guide/general/controllers.html)

[Например, при запросе адреса **products/shirts/123** будет вызван контроллер **shirts** и метод **id\_123** в нем.](http://code-igniter.ru/user_guide/general/controllers.html)

[Вы, также, можете одновременно использовать Маски и Регулярные Выражения.](http://code-igniter.ru/user_guide/general/controllers.html)

## [Зарезервированные роуты](http://code-igniter.ru/user_guide/general/controllers.html)

[Есть два зарезервированных роута:](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['default\_controller'] = 'welcome';](http://code-igniter.ru/user_guide/general/controllers.html)

[Этот роут показывает что необходимо загружать, когда пользователь ввел корневой адрес, т.е. прямо не показал к какому контроллеру обращаться. В этом примере будет загружен класс «welcome». Вам рекомендуется всегда указывать этот роут, т.к. страницы 404 будут, по умолчанию, ссылаться на него.](http://code-igniter.ru/user_guide/general/controllers.html)

[$route['scaffolding\_trigger'] = 'scaffolding';](http://code-igniter.ru/user_guide/general/controllers.html)

[Здесь вам предлагается установить секретное слово, при наличии которого в URL будет активироваться возможность скаффолдинга. Пожалуйста прочтите главу [Скаффолдинг](http://code-igniter.ru/user_guide/general/scaffolding.html) для выяснения подробностей.](http://code-igniter.ru/user_guide/general/controllers.html)

[**Важно:**  Зарезервированные роуты должны быть объявлены раньше роутов с Масками или Регулярными Выражениями.](http://code-igniter.ru/user_guide/general/scaffolding.html)

# [Обработка ошибок](http://code-igniter.ru/user_guide/general/scaffolding.html)

[CodeIgniter позволяет создавать описание возникающих ошибок при помощи функций, описанных ниже. В дополнение имеется класс, при помощи которого можно вносить в лог все сообщения об ошибках.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[**Замечание:** По умолчанию, CodeIgniter отображает все ошибки PHP. Возможно вы захотите изменить это поведение, когда ваше приложение будет дописано. Вам надо просто найти функцию **error\_reporting()** в самом начале главного файла index.php. Отключение сообщений об ошибках отключает запись в лог, в случае их возникновения.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[В отличие от большинства систем, в CodeIgniter обработка ошибок представлена простыми функциями, которые доступны в любой части приложения. Такой подход позволяет не заботиться об области видимости, в которой мы сейчас находимся и вызывать обработчики откуда угодно.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Следующие функции предназначены для генерации ошибок:](http://code-igniter.ru/user_guide/general/scaffolding.html)

## [show\_error('message')](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Эта функция выведет сообщение об ошибке, переданное в нее, используя шаблон, находящийся по этому адресу:](http://code-igniter.ru/user_guide/general/scaffolding.html)

[**application/errors/error\_general.php**](http://code-igniter.ru/user_guide/general/scaffolding.html)

## [show\_404('page')](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Эта функция отобразит сообщение о том, что файл не найден (ошибка 404), используя шаблон:](http://code-igniter.ru/user_guide/general/scaffolding.html)

[**application/errors/error\_404.php**](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Функция ожидает строку, содержащую путь к странице, которая не была обнаружена. Заметим, что CodeIgniter автоматически покажет сообщение 404, если не найден контроллер, к которому происходит обращение.](http://code-igniter.ru/user_guide/general/scaffolding.html)

## [log\_message('level', 'message')](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Эта функция позволяет вам писать сообщения в лог-файлы. В первом параметре вы должны указать уровень критичности сообщения (debug, error, info), а во втором само сообщение. Например:](http://code-igniter.ru/user_guide/general/scaffolding.html)

[if ($some\_var == "")  
{  
    log\_message('error', 'Некоторая переменная не содержит значения');  
}  
else  
{  
    log\_message('debug', 'Значение переменной установлено верно');  
}  
log\_message('info', 'Цель этой переменной в том, чтобы хранить какое-то значение');](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Есть три типа сообщения:](http://code-igniter.ru/user_guide/general/scaffolding.html)

1. [Error Messages. Это фактические, актуальные ошибки, такие как ошибки PHP или пользователей.](http://code-igniter.ru/user_guide/general/scaffolding.html)
2. [Debug Messages. Это сообщения для отладки, например можно использовать чтобы проверить был ли инициализирован класс.](http://code-igniter.ru/user_guide/general/scaffolding.html)
3. [Informational Messages. Это самый низкий приоритет сообщения, просто предоставление информации в отношении какого-либо процесса. CodeIgniter использует для этого типа сообщений не родные методы языка, но вы вполне можете пользоваться этим типом сообщений.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[**Замечание:** Для того чтобы была возможность вести логи, ваша папка «logs» должна быть доступна для записи. К тому же, вы должны установить уровень для ведения логов. Можно, например, оставить только протоколирование критических ошибок, а остальные игнорировать. Если этот порог установить на ноль, то отладка будет отключена.](http://code-igniter.ru/user_guide/general/scaffolding.html)

# [Кеширование веб-страниц](http://code-igniter.ru/user_guide/general/scaffolding.html)

[CodeIgniter позволяет закешировать страницы, чтобы добиться максимальной производительности.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Хотя сам CodeIgniter и достаточно быстрый, количество динамической информации на выходе в ваших страницах соотносится с ресурсами сервера, памятью, процессорными циклами, привязанными к скорости загрузки страниц. Кэшируя страницы с того момента как они были приведены к окончательному виду, вы можете достичь производительности, сравнимой со статическими страницами.](http://code-igniter.ru/user_guide/general/scaffolding.html)

## [Как работает кеширование?](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Кеширование может быть включено для каждой страницы, а так же указан интервал времени, в течение которого страница должна не генерироваться заново, а браться из кеша. При загрузке страницы в первый раз, кеш-файл будет записан в директорию **system/cache**. При последующих запросах страница будет считываться из кеша и отправляться в браузер пользователя. Если время кеширования истекло, то страница стирается, генерируется заново, записывается в кеш и отдается пользователю в браузер.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Замечание: Показатели производительности не кешируются чтобы можно было просмотреть нагрузку во время кеширования.](http://code-igniter.ru/user_guide/general/scaffolding.html)

## [Включение кеширования](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Для включения кеширования впишите в любую функцию вашего контроллера:](http://code-igniter.ru/user_guide/general/scaffolding.html)

[$this->output->cache(**n**);](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Где **n** это количество **минут** на которые вы хотите оставлять страницу в кеше, до ее обновления.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[Приведенный выше вызов может быть указан в любом месте вашей функции. Место вызова никак не повлияет на результат, поэтому можете сделать это там, где посчитаете наиболее логичным. Как только вызов будет написан — ваша страница окажется в кеше.](http://code-igniter.ru/user_guide/general/scaffolding.html)

[**Предупреждение:** Т.к. CodeIgniter сохраняет в кеше только контент для вывода, то кеширование будет работать только для контроллеров, генерирующих вывод при помощи [отображения](http://code-igniter.ru/user_guide/general/views.html).](http://code-igniter.ru/user_guide/general/scaffolding.html)

[**Замечание:** Т.к. закешированные файлы будут записываться в директорию **system/cache**, вы должны установить системные права на запись в нее.](http://code-igniter.ru/user_guide/general/views.html)

## [Удаление кеша](http://code-igniter.ru/user_guide/general/views.html)

[Если вам больше не надо кешировать некоторое отображение, то просто удалите вызов функции, создающей кеш, и он не будет обновлен после того как истечет. Замечание: Удаление функции не очистит существующий кеш. Он удалится, когда кончится время его жизни. Если вам необходимо стереть кеш немедленно, то придется удалить его из папки с кешем вручную.](http://code-igniter.ru/user_guide/general/views.html)

# [Профилирование приложения](http://code-igniter.ru/user_guide/general/views.html)

[Класс Profiler отобразит результаты производительности приложения, запросы, выполненные вами и $\_POST-данные внизу страницы приложения. Эта информация может быть полезной в ходе разработки для отладки и оптимизации приложения.](http://code-igniter.ru/user_guide/general/views.html)

## [Инициализаци класса](http://code-igniter.ru/user_guide/general/views.html)

[**Важно:**  Этот класс **НЕ** нуждается в инициализации. Он загружается автоматически классом [Output](http://code-igniter.ru/user_guide/libraries/output.html) если профилирование включено, как показано ниже.](http://code-igniter.ru/user_guide/general/views.html)

## [Включение профилирования](http://code-igniter.ru/user_guide/libraries/output.html)

[Для включения профилирования, поместите следующую функцию в любом месте вашего [контроллера](http://code-igniter.ru/user_guide/general/controllers.html):](http://code-igniter.ru/user_guide/libraries/output.html)

[$this->output->enable\_profiler(TRUE);](http://code-igniter.ru/user_guide/general/controllers.html)

[Когда отчет будет сгенерирован — он появится в нижней части страницы.](http://code-igniter.ru/user_guide/general/controllers.html)

[Для отключения профилирования можно использовать:](http://code-igniter.ru/user_guide/general/controllers.html)

[$this->output->enable\_profiler(FALSE);](http://code-igniter.ru/user_guide/general/controllers.html)

## [Установка точек производительности](http://code-igniter.ru/user_guide/general/controllers.html)

[Для того чтобы Profiler мог отобразить результаты производительности в определенном месте приложения, вы можете, пользуясь специальным синтаксисом, установить точки. Пожалуйста прочтите подробнее о точках производительности в описании класса [Benchmark Class](http://code-igniter.ru/user_guide/libraries/benchmark.html).](http://code-igniter.ru/user_guide/general/controllers.html)

# [Управление вашим приложением](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[По умолчанию предполагается, что после установки CodeIgniter вы будете создавать единственное приложение и хранить его в **system/application/**. Однако вполне возможна работа нескольких приложений на едином ядре CodeIgniter или, даже, переименование или перемещение папки **application**.](http://code-igniter.ru/user_guide/libraries/benchmark.html)

## [Переименование папки Application](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[Если вы хотите переименовать вашу папку **application**, то вы можете делать это сколько угодно раз, просто открывайте после этого ваш главный**index.php** и напишите новое имя папки в переменную **$application\_folder**:](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[$application\_folder = "application";](http://code-igniter.ru/user_guide/libraries/benchmark.html)

## [Перемещение папки Application](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[Возможно вы захотите вынести вашу папку **application** за пределы папки **system**. Для этого откройте основной файл **index.php** и установите путь от корня сервера до папки **application** в переменной **$application\_folder**.](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[$application\_folder = "/Path/to/your/application";](http://code-igniter.ru/user_guide/libraries/benchmark.html)

## [Запуск нескольких приложений с единым ядром CodeIgniter](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[Если вы хотите делить ядро CodeIgniter между несколькими приложениями, то просто переместите все директории в **application** в собственные поддиректории.](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[Например, вы хотите создать два приложения «foo» and «bar». Вы должны создать структуру папок, похожую на эту:](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[system/application/**foo**/  
system/application/**foo**/config/  
system/application/**foo**/controllers/  
system/application/**foo**/errors/  
system/application/**foo**/libraries/  
system/application/**foo**/models/  
system/application/**foo**/views/  
system/application/**bar**/  
system/application/**bar**/config/  
system/application/**bar**/controllers/  
system/application/**bar**/errors/  
system/application/**bar**/libraries/  
system/application/**bar**/models/  
system/application/**bar**/views/](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[Для выбора нужного приложения для текущего запроса вам необходимо открыть ваш файл **index.php** и установить переменную **$application\_folder**. Например, для использования приложения «foo» вы должны написать так:](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[$application\_folder = "application/foo";](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[**Замечание:**  Каждое из ваших приложений должно иметь собственный файл **index.php**, в котором будет указана используемая папка с приложением. Файл index.php может быть назван как угодно.](http://code-igniter.ru/user_guide/libraries/benchmark.html)

# [Альтернативный синтаксис PHP для файлов отображений](http://code-igniter.ru/user_guide/libraries/benchmark.html)

[Если вы не используете [шаблонизатор](http://code-igniter.ru/user_guide/libraries/parser.html) CodeIgniter — вы будете писать отображения на чистом PHP. Для уменьшения количества PHP кода и для быстрой идентификации блоков кода рекомендуется использовать альтернативный синтаксис PHP для управляющих структур и пользоваться короткой формой функции echo. Если вы не знакомы с этим синтаксисом, он позволяет устранить фигурные скобки и отказаться от использования echo.](http://code-igniter.ru/user_guide/libraries/benchmark.html)

## [Автоматическая поддержка коротких тегов](http://code-igniter.ru/user_guide/libraries/parser.html)

[**Замечание:** Если вы обнаружите, что синтаксис описанный на этой странице не работает, то на вашем сервере отключена поддержка «коротких тегов» в файле настройки PHP php.ini. CodeIgniter может опционально искать короткие теги на-лету, позволяя вам использовать их независимо от поддержки на уровне сервера. Эта возможность может быть включена в вашем файле **config/config.php**.](http://code-igniter.ru/user_guide/libraries/parser.html)

[Учтите, что если вы используете эту функцию, то в случае обнаружения ошибок в **файле с отображением** информация о них, а так же о строках, на которых они находятся, может быть не точной. Вместо этого все ошибки будут показаны как ошибки функции **eval()**.](http://code-igniter.ru/user_guide/libraries/parser.html)

## [Альтернатива Echo](http://code-igniter.ru/user_guide/libraries/parser.html)

[Обычно, для простого вывода информации вы пишете так:](http://code-igniter.ru/user_guide/libraries/parser.html)

[<?php echo $variable; ?>](http://code-igniter.ru/user_guide/libraries/parser.html)

[Вместо этого, при помощи альтернативного синтаксиса, вы можете писать:](http://code-igniter.ru/user_guide/libraries/parser.html)

[<?=$variable?>](http://code-igniter.ru/user_guide/libraries/parser.html)

## [Альтернативные управляющие структуры](http://code-igniter.ru/user_guide/libraries/parser.html)

[Контрольные структуры, такие как **if**, **for**, **foreach** и **while** могут быть написаны в упрощенной форме. Вот пример использования foreach:](http://code-igniter.ru/user_guide/libraries/parser.html)

[<ul>  
**<?php foreach($todo as $item): ?>**  
<li>**<?=$item?>**</li>  
**<?php endforeach; ?>**  
</ul>](http://code-igniter.ru/user_guide/libraries/parser.html)

[Заметьте, что нет фигурных скобок. Вместо этого последняя скобка заменяется **endforeach**. Каждая из управляющих структур, перечисленных выше, имеет похожий синтаксис окончания: **endif**, **endfor**, **endforeach** и **endwhile**](http://code-igniter.ru/user_guide/libraries/parser.html)

[Кроме того, не забудьте, что после открывающей структуры необходимо использовать двоеточие, а не точку с запятой, как обычно.](http://code-igniter.ru/user_guide/libraries/parser.html)

# [Безопасность](http://code-igniter.ru/user_guide/libraries/parser.html)

[Эта страница описывает «Лучшие практики» в отношении веб-безопасности, а так же о том, как CodeIgniter обеспечивает вашу безопасность.](http://code-igniter.ru/user_guide/libraries/parser.html)

## [Безопасность URI](http://code-igniter.ru/user_guide/libraries/parser.html)

[CodeIgniter сильно ограничивает диапазон символов, которые могут быть указаны в URL, чтобы таким образом не передать в ваше приложение ничего вредоносного. URI могут содержать только следующее:](http://code-igniter.ru/user_guide/libraries/parser.html)

* [Буквенно-цифровой текст](http://code-igniter.ru/user_guide/libraries/parser.html)
* [Тильда: ~](http://code-igniter.ru/user_guide/libraries/parser.html)
* [Точка: .](http://code-igniter.ru/user_guide/libraries/parser.html)
* [Двоеточие: :](http://code-igniter.ru/user_guide/libraries/parser.html)
* [Символ подчеркивания: \_](http://code-igniter.ru/user_guide/libraries/parser.html)
* [Дефис: -](http://code-igniter.ru/user_guide/libraries/parser.html)

## [GET, POST и COOKIE данные](http://code-igniter.ru/user_guide/libraries/parser.html)

[GET запросы просто не принимаются системой, т.к. CodeIgniter использует URI на основе сегментов, вместо традиционной строки запроса (если вы хотите использовать строки запроса, то можете включить эту опцию в конфигурационном файле). Глобальный GET массив **удаляется** классом Input, подключающимся при инициализации.](http://code-igniter.ru/user_guide/libraries/parser.html)

## [Register\_globals](http://code-igniter.ru/user_guide/libraries/parser.html)

[При инициализации системы удаляются все глобальные массивы, за исключением $\_POST и $\_COOKIE. Их отключение равносильно опции register\_globals = off.](http://code-igniter.ru/user_guide/libraries/parser.html)

## [magic\_quotes\_runtime](http://code-igniter.ru/user_guide/libraries/parser.html)

[Директива magic\_quotes\_runtime отключается при загрузке системы, чтобы вам не пришлось удалять слеши при извлечении данных из базы данных.](http://code-igniter.ru/user_guide/libraries/parser.html)

# [Наилучшая практика](http://code-igniter.ru/user_guide/libraries/parser.html)

[Прежде чем работать с любыми данными в вашем приложении, будь то POST из формы на сайте, COOKIE, информация из URI, XML-RPC и даже данными из массива SERVER, вы должны проделать три простых шага:](http://code-igniter.ru/user_guide/libraries/parser.html)

1. [профильтровать данные, как если бы они были «грязными»](http://code-igniter.ru/user_guide/libraries/parser.html)
2. [проверить данные на соответствие ожидаемой длине, типу, размеру и т.п. (в некоторых случаях этот шаг может заменить первый);](http://code-igniter.ru/user_guide/libraries/parser.html)
3. [экранировать данные перед отправкой в базу данных.](http://code-igniter.ru/user_guide/libraries/parser.html)

[CodeIgniter предоставляет следующий функционал для помощи в этом процессе:](http://code-igniter.ru/user_guide/libraries/parser.html)

## [Фильтрация XSS](http://code-igniter.ru/user_guide/libraries/parser.html)

[CodeIgniter поставляется с фильтром на Cross Site Scripting. Этот фильтр ищет популярные методы для встраивания вредоносного Javascript кода в ваш сайт и другие виды кода для блокировки cookies и тому подобных вещей. Фильтр XSS подробно описан [здесь](http://code-igniter.ru/user_guide/libraries/input.html).](http://code-igniter.ru/user_guide/libraries/parser.html)

## [Валидация (проверка) данных](http://code-igniter.ru/user_guide/libraries/input.html)

[CodeIgniter имеет [класс валидации](http://code-igniter.ru/user_guide/libraries/validation.html), который поможет вам проверить ваши данные и подготовить их к записи в базу данных.](http://code-igniter.ru/user_guide/libraries/input.html)

## [Экранирование всей информации перед вставкой в базу данных](http://code-igniter.ru/user_guide/libraries/validation.html)

[Никогда не вставляйте информацию в базу данных без ее экранирования. Пожалуйста просмотрите страницу о [запросах](http://code-igniter.ru/user_guide/database/queries.html) для получения дополнительной информации.](http://code-igniter.ru/user_guide/libraries/validation.html)

**Глава 3. Классы.**

# Класс Benchmarking

В CodeIgniter есть всегда доступный класс Benchmarking, который позволяет получить время, затраченное на выполнение кода между двумя отмеченными точками в коде.

**Примечание:** Данный класс инициализируется приложением автоматически. Вручную инициализировать его не требуется.

В дополнение, таймер всегда стартует в начале работы фреймворка и работает до момента отсылки информации браузеру, что позволяет произвести очень точный замер времени выполнения приложения.

### Содержание

* [Использование класса Benchmark](http://code-igniter.ru/user_guide/libraries/benchmark.html#using)
* [Профилирование меток](http://code-igniter.ru/user_guide/libraries/benchmark.html#profiler)
* [Вывод общего времени выполнения](http://code-igniter.ru/user_guide/libraries/benchmark.html#execution)
* [Вывод количества потребляемой памяти](http://code-igniter.ru/user_guide/libraries/benchmark.html#memory)

## Использование класса Benchmark

Класс Benchmark может быть использован в [контроллерах](http://code-igniter.ru/user_guide/general/controllers.html), [отображениях](http://code-igniter.ru/user_guide/general/views.html) или моделях. Процесс использования таков:

1. Отмечаем стартовую точку
2. Отмечаем конечную точку
3. Вызываем функцию elapsed\_time для просмотра результата

Пример кода:

$this->benchmark->mark('code\_start');  
// Тут ещё какой–то код…  
$this->benchmark->mark('code\_end');  
echo $this->benchmark->elapsed\_time('code\_start', 'code\_end');

**Примечание:** Метки "code\_start" и "code\_end" произвольны. Вы можете использовать любые метки. Также, вы можете выставить несколько меток. Пример:

$this->benchmark->mark('dog');  
// Тут ещё какой–то код…  
$this->benchmark->mark('cat');  
// Тут ещё какой–то код…  
$this->benchmark->mark('bird');  
echo $this->benchmark->elapsed\_time('dog', 'cat');  
echo $this->benchmark->elapsed\_time('cat', 'bird');  
echo $this->benchmark->elapsed\_time('dog', 'bird');

## Профилирование меток

Если вы хотите, чтобы данные замера были доступны классу [Profiler](http://code-igniter.ru/user_guide/general/profiling.html), все метки должны быть парными. Имя стартовой метки должно заканчиваться на **\_start**, а конечной метки — на **\_end**. Пример:

$this->benchmark->mark('my\_mark**\_start**');  
// Тут ещё какой–то код…  
$this->benchmark->mark('my\_mark**\_end**');   
$this->benchmark->mark('another\_mark**\_start**');  
// Тут ещё какой–то код…  
$this->benchmark->mark('another\_mark**\_end**');

Подробнее профилирование описано [на соответствующей странице](http://code-igniter.ru/user_guide/general/profiling.html).

## Вывод общего времени выполнения

Если вы хотите отобразить суммарное время, прошедшее с момента начала работы CodeIgniter до момента отдачи браузеру, поместите следующий код в один из ваших шаблонов отображения:

<?=$this->benchmark->elapsed\_time();?>

Заметьте, что для вычисления прошедшего времени используется та же функция, что и предыдущих примерах. Разница в том, что вы **не указываете**какие–либо параметры. При отсутствии параметров CodeIgniter не останавливает замер до финального вывода в браузер. Не важно, где вы вызвали функцию: таймер будет работать до самого конца.

Альтернативный способ отобразить затраченное время, в том случае, если вы не хотите использовать PHP–код, — использовать псевдо–переменную в файлах отображения:

{elapsed\_time}

**Примечание:** Если вы хотите замерить что–либо в вашем контроллере — задавайте свои стартовые и конечные метки.

## Вывод количества потребляемой памяти

Если PHP на вашем сервере собран с опцией --enable-memory-limit, вы можете вывести количество потребляемой приложением памяти используя следующий код в одном из файлов отображений:

<?=$this->benchmark->memory\_usage();?>

**Примечание:** Данная функция может быть использована только в отображениях. Значение отражает суммарную память, используемую всем приложением.

Альтернативный способ отобразить потребление памяти, в том случае, если вы не хотите использовать PHP:

{memory\_usage}

# Класс Calendar

Класс Calendar позволяет динамически создавать календари. Отображение вашего календаря может быть изменено с помощью использования шаблонов календаря, которые предоставляют полный контроль над каждым аспектом его дизайна. Также вы можете поместить данные в ячейки календаря.

## Инициализация Класса

Как и многие другие классы в CodeIgniter, класс Calendar инициализируется в вашем контроллере с использованием функции **$this->load->library**:

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

Однажды загруженный, объект Calendar будет доступен, с использованием:**$this->calendar**

## Отображение календаря

Простой пример, показывающий, как вы можете отобразить календарь:

$this->load->library('calendar');  
echo $this->calendar->generate();

Вышеприведенный код сгенерирует календарь для текущего месяца/года в соответствии с временем на вашем сервере. Чтобы отобразить календарь с определенным месяцем и годом, вы должны сообщить эту информацию в функцию, генерирующую календарь:

$this->load->library('calendar');  
echo $this->calendar->generate(**2006**, **6**);

Вышеприведенный код сгенерирует календарь для июня 2006 года. Первый параметр соответствует году, второй - месяцу.

## Помещение данных в ячейки вашего календаря

Добавление данных в ячейки вашего календаря включает создание ассоциативного массива, в котором ключи соответствуют дням, в которые вы хотите добавить данные, а значения массива содержат эти данные. Массиву соответствует третий параметр функции, генерирующей календарь. Рассмотрим этот пример:

$this->load->library('calendar');  
$data = array(  
               3  => 'http://your-site.com/news/article/2006/03/',  
               7  => 'http://your-site.com/news/article/2006/07/',  
               13 => 'http://your-site.com/news/article/2006/13/',  
               26 => 'http://your-site.com/news/article/2006/26/'  
             );  
echo $this->calendar->generate(**2006**, **6**, **$data**);

Используя вышеприведенный пример, дни 3, 7, 13, и 26 наделяются ссылками, указывающими на предусмотренные вами страницы.

**Примечание:** По умолчанию предполагается, что ваш массив содержит ссылки. В секции, рассказывающей о шаблонах календаря, вы увидете как вы можете изменить отображение данных в ячейках календаря.

## Установка предпочтений отображения

Вы можете установить семь предпочтений для управления отображением календаря. Предпочтения устанавливаются отправкой массива предпочтений вторым параметром функции загрузки. Пример:

$prefs = array (  
               'start\_day'    => 'saturday',  
               'month\_type'   => 'long',  
               'day\_type'     => 'short'  
             );  
$this->load->library('calendar', $prefs);  
echo $this->calendar->generate();

Вышеприведенный код отображает календарь, начинающийся с субботы, использующий "длинный" заголовок месяца и "короткие" имена дней недели. Больше информации, касательно преподчтений отображения, приведено ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Параметры** | **Описание** |
| **template** | Нет | Нет | Строка, содержащая шаблон вашего календаря. О шаблонах смотрите секцию "Создание шаблонов календаря". |
| **local\_time** | time() | Нет | Соответствующий текущему времени unix timestamp. |
| **start\_day** | sunday | Любой день недели (sunday, monday, tuesday и т.д.) | Устанавливает, с какого дня недели начинать отображение календаря. |
| **month\_type** | long | long, short | Определяет отображение имени месяца в заголовке. long = January, short = Jan. |
| **day\_type** | abr | long, short, abr | Определяет отображение дней недели в заголовках колонок. long = Sunday, short = Sun, abr = Su. |
| **show\_next\_prev** | FALSE | TRUE/FALSE (boolean) | Определяет отображение ссылок на следующий/предыдущий месяц. Смотрите информацию об этой возможности ниже. |
| **next\_prev\_url** | Нет | A URL | Устанавливает базовый путь, используемый ссылками следующий/предыдущий в календаре. |

## Отображение ссылок на следующий/предыдущий месяц

Чтобы разрешить вашему календарю менять месяц и год через ссылки следующий/предыдущий, требуется установить код вашего календаря, подобный этому:

$prefs = array (  
               'show\_next\_prev'  => TRUE,  
               'next\_prev\_url'   => 'http://www.your-site.com/index.php/calendar/show/'  
             );  
$this->load->library('calendar', $prefs);  
echo $this->calendar->generate(**$this->uri->segment(3)**, **$this->uri->segment(4)**);

Несколько важных замечаний о вышеприведенном коде:

* Вы должны установить "show\_next\_prev" в значение TRUE.
* Вы должны установить контроллеру, содержащему ваш календарь, значение URL в предпочтение "next\_prev\_url".
* Вы должны передать "year" и "month" функции, генерирующей календарь, через сегменты URI, когда они показываются. (Примечание: Класс календаря автоматически добавляет год/месяц к базовому URL, предоставленному вами.).

## Создание шаблонов календаря

Создав шаблон календаря, вы получаете полный контроль над отображением вашего календаря. Каждый компонент вашего календаря будет размещен внутри пары псевдо-переменных как показано здесь:

$prefs['template'] = '  
   **{table\_open}<table border="0" cellpadding="0" cellspacing="0">{/table\_open}**  
   **{heading\_row\_start}<tr>{/heading\_row\_start}**  
   **{heading\_previous\_cell}<th><a href="{previous\_url}">&lt;&lt;</a></th>{/heading\_previous\_cell}**  
   **{heading\_title\_cell}<th colspan="{colspan}">{heading}</th>{/heading\_title\_cell}**  
   **{heading\_next\_cell}<th><a href="{next\_url}">&gt;&gt;</a></th>{/heading\_next\_cell}**  
   **{heading\_row\_end}</tr>{/heading\_row\_end}**  
   **{week\_row\_start}<tr>{/week\_row\_start}**  
   **{week\_day\_cell}<td>{week\_day}</td>{/week\_day\_cell}**  
   **{week\_row\_end}</tr>{/week\_row\_end}**  
   **{cal\_row\_start}<tr>{/cal\_row\_start}**  
   **{cal\_cell\_start}<td>{/cal\_cell\_start}**  
   **{cal\_cell\_content}<a href="{content}">{day}</a>{/cal\_cell\_content}**  
   **{cal\_cell\_content\_today}<div class="highlight"><a href="{content}">{day}</a></div>{/cal\_cell\_content\_today}**  
   **{cal\_cell\_no\_content}{day}{/cal\_cell\_no\_content}**  
   **{cal\_cell\_no\_content\_today}<div class="highlight">{day}</div>{/cal\_cell\_no\_content\_today}**  
   **{cal\_cell\_blank}&nbsp;{/cal\_cell\_blank}**  
   **{cal\_cell\_end}</td>{/cal\_cell\_end}**  
   **{cal\_row\_end}</tr>{/cal\_row\_end}**  
   **{table\_close}</table>{/table\_close}**  
';  
$this->load->library('calendar', $prefs);  
echo $this->calendar->generate();

# Класс Config

Класс Config представляет доступ к конфигурационным данным. Они расположены в стандартном конфигурационном файле (**application/config/config.php**) или в Ваших собственных файлах-конфигурациях.

**Важно:** Этот класс инициализируется автоматически. Нет необходимости загружать его вручную.

## Структура конфигурационного файла

Изначально CodeIgniter имеет один конфигурационный файл, размещенный в**application/config/config.php**. Если открыть этот файл, то можно заметить, что все конфигурационные данные входят в массив **$config**.

Вы можете добавлять новые значения в этот файл , или, если Вы решили хранить конфигурационные данные по отдельности (для доступа только к тем данным, которые необходимы в определенный момент), просто создайте новый php-файл и разместите его в директории **config**.

**Примечание:** Если Вы создали собственный файл с данными, желательно сохранить значения в массиве **$config**. CodeIgniter будет взаимодействовать с конф. файлами без проблем, даже если массивы имеют одно имя. Разумеется, имена индексов массивов должны различаться.

## Загрузка ****конфигурационного**** файла

**Примечание:** CodeIgniter автоматически загружает стандартный конф. файл(**application/config/config.php**), так что Вы должны загружать только созданные вручную конф. файлы.

Есть два способа загрузки конфигурационного файла:

1. **Ручная загрузка**

Для загрузки одного из Ваших конф. файлов используйте следующий код в[контроллере](http://code-igniter.ru/user_guide/general/controllers.html), в котором нужны конф. данные:

$this->config->load('**filename**');

Где **filename** - имя Вашего конфигурационного файла без расширения .php.

Если нужно загрузить несколько конф. файлов с одинаковыми массивами, то следует все массивы каждого конф. файла объединить в один. Для этого следует установить второй параметр функции в значение **TRUE** , и весь конфигурационный файл будет представлен мультимассивом, название которого будет соответствовать названию файла. Например:

$this->config->load('**blog\_settings**', **TRUE**);

Подробное описание этого способа в подразделе **Получение конфигурационных данных**

Третий параметр позволяет обрабатывать ошибки, если конф. файл не найден :

$this->config->load('**blog\_settings**', **FALSE**, **TRUE**);

1. **Автоматическая загрузка**

Если нужно создать глобальный доступ к конфигурационному файлу, следует записать его в автозагрузку. Для этого откройте файл **autoload.php** , размещенный по адресу **application/config/autoload.php**, и добавьте имя файла (без .php) в массив автозагрузки под индексом 'config'.

## Получение конфигурационных данных

Для получения значения из конф. файл используется следующая функция :

$this->config->item('**item name**');

Где **item name** является индексом массива $config, значение которого желаете получить. Например, для получения пути к сайту используем этот код:

$lang = $this->config->item('base\_url');

Функция возвращает FALSE (логическое), если данных, которые Вы пытаетесь получить, нет.

Если используется второй параметр функции **$this->config->load** , то для доступа к значению следует установить второй параметр функции **$this->config->item()** , равный имени файла. Например:

// Загружаем конф. файл blog\_settings.php и ассоциируем его с массивом "blog\_settings"  
$this->config->load('**blog\_settings**', '**TRUE**');  
// Получаем значение site\_name из массива blog\_settings  
$site\_name = $this->config->item('**site\_name**', '**blog\_settings**');  
// Альтернативный способ сделать это же самое:  
$blog\_config = $this->config->item('**blog\_settings**');  
$site\_name = $blog\_config['site\_name'];

## Сохранение конфигурационных данных

Если Вы желаете динамически изменять значения конфигурационных данных с последующим сохранением их в файле-конфигураторе, то используйте следующий код:

$this->config->set\_item('**item\_name**', '**item\_value**');

Где **item\_name** это название индекса массива $config, который желаете изменить, а **item\_value** - новое значение этого индекса .

## Вспомогательные функции

Класс Config содержит следующие вспомогательные функции :

## $this->config->site\_url();

Эта функция возвращает URL сайта вместе со значением "index", если Вы его задали в конфигурационном файле .

## $this->config->system\_url();

Эта функция возвращает URL к папке **system**.

Класс работы с базами данных

CodeIgniter поставляется с полнофункциональным и очень быстрым классом абстракции баз данных, который поддерживает как традиционные структуры, так и шаблоны Active Record. Функции интуитивно понятны и имеют простой синтаксис.

* [Быстрый старт: Примеры использования](http://code-igniter.ru/user_guide/database/examples.html)
* [Конфигурация для работы с базами данных](http://code-igniter.ru/user_guide/database/configuration.html)
* [Подключение к базе данных](http://code-igniter.ru/user_guide/database/connecting.html)
* [Выполнение запросов](http://code-igniter.ru/user_guide/database/queries.html)
* [Генерация результатов запроса](http://code-igniter.ru/user_guide/database/results.html)
* [Функции хелпера запросов](http://code-igniter.ru/user_guide/database/helpers.html)
* [Класс Active Record](http://code-igniter.ru/user_guide/database/active_record.html)
* [Транзакции](http://code-igniter.ru/user_guide/database/transactions.html)
* [Метаданные таблиц](http://code-igniter.ru/user_guide/database/table_data.html)
* [Метаданные полей](http://code-igniter.ru/user_guide/database/fields.html)
* [Вызов произвольных функций](http://code-igniter.ru/user_guide/database/call_function.html)
* [Кеширование запросов](http://code-igniter.ru/user_guide/database/caching.html)
* [Манипуляции базой данных с помощью Database Forge](http://code-igniter.ru/user_guide/database/forge.html)
* [Утилиты для работы с базами данных](http://code-igniter.ru/user_guide/database/utilities.html)

# Быстрый старт: Пример кода

Здесь приведён пример использования класса для работы с базами данных. За более подробной информацией обращайтесь к отдельным страницам данного руководства.

## Инициализация класса работы с базами данных

Приведённый ниже код загружает и инициализирует класс работы с базами данных, в соответствии с настройками, указанными в файле [конфигурации](http://code-igniter.ru/user_guide/database/configuration.html):

$this->load->database();

После загрузки класс готов к использованию, как описано ниже.

Замечание: Если все ваши страницы требуют доступа к базе данных, вы можете настроить автоматическое подключение. Смотрите страницу с описанием [подключений](http://code-igniter.ru/user_guide/database/connecting.html) для получения более подробной информации.

## Стандартный запрос с получением множества результатов (объектная версия)

$query = $this->db->query('SELECT name, title, email FROM my\_table');  
foreach ($query->result() as $row)  
{  
    echo $row->title;  
    echo $row->name;  
    echo $row->email;  
}  
echo 'Всего результатов: ' . $query->num\_rows();

Приведённая выше функция **result()** возвращает массив **объектов**. Пример: $row->title

## Стандартный запрос с получением множества результатов (версия с массивами)

$query = $this->db->query('SELECT name, title, email FROM my\_table');  
foreach ($query->result\_array() as $row)  
{  
    echo $row['title'];  
    echo $row['name'];  
    echo $row['email'];  
}

Приведённая выше функция **result\_array()** возвращает массив индексов. Пример: $row['title']

## Тестируем получение результатов выборки

Если вы выполняете запросы, которые могут возвращать **пустой** результ, то для проверки количества возвращаемых строк используйте функцию**num\_rows()** :

$query = $this->db->query("ВАШ ЗАПРОС");  
if ($query->num\_rows() > 0)  
{  
   foreach ($query->result() as $row)  
   {  
      echo $row->title;  
      echo $row->name;  
      echo $row->body;  
   }  
}

## Стандартный запрос с единичым результатом выборки

$query = $this->db->query('SELECT name FROM my\_table LIMIT 1');  
$row = $query->row();  
echo $row->name;

Функция **row()** возвращает **объект**. Пример: $row->name

## Стандартный запрос с единичым результатом выборки (версия с массивами)

$query = $this->db->query('SELECT name FROM my\_table LIMIT 1');  
$row = $query->row\_array();  
echo $row['name'];

Функция **row\_array()** возвращает **массив**. Пример: $row['name']

## Стандартная вставка

$sql = "INSERT INTO mytable (title, name)   
        VALUES (".$this->db->escape($title).", ".$this->db->escape($name).")";  
$this->db->query($sql);  
echo $this->db->affected\_rows();

## Запрос Active Record

Шаблон [Active Record](http://code-igniter.ru/user_guide/database/active_record.html) позволяет упростить использование возвращаемых данных:

$query = $this->db->get('table\_name');  
foreach ($query->result() as $row)  
{  
    echo $row->title;  
}

Функция **get()** возвращает все записи из указанной таблицы. Класс [Active Record](http://code-igniter.ru/user_guide/database/active_record.html) содержит полный набор функций для работы с данными.

## Вставка данных с использованием Active Record

$data = array(  
               'title' => $title,  
               'name' => $name,  
               'date' => $date  
            );  
$this->db->insert('mytable', $data);   
// Создаёт: INSERT INTO mytable (title, name, date) VALUES ('{$title}', '{$name}', '{$date}')

# Конфигурация для работы с базами данных

В CodeIgniter есть файл конфигурации, который позволяет хранить все настройки подключения (имя пользователя, пароль, наименование базы данных и т.д.) в одном месте. Этот файл находится в:

**application/config/database.php**

Конфигурационные настройки сохраняются в виде многомерного массива со следующей структурой:

$db['default']['hostname'] = "localhost";  
$db['default']['username'] = "root";  
$db['default']['password'] = "";  
$db['default']['database'] = "database\_name";  
$db['default']['dbdriver'] = "mysql";  
$db['default']['dbprefix'] = "";  
$db['default']['pconnect'] = TRUE;  
$db['default']['db\_debug'] = FALSE;  
$db['default']['cache\_on'] = FALSE;  
$db['default']['cachedir'] = "";  
$db['default']['char\_set'] = "utf8";  
$db['default']['dbcollat'] = "utf8\_general\_ci";

Причиной хранения данных в многомерном массиве является возможность опционального использования нескольких наборов настроек подключения. Если, к примеру, вы работаете с одним дистрибутивом в различном окружении (тестовое, рабочее, готовый продукт), то у вас появляется возможность устанавливать различные группы соединений для каждого окружения, чтобы по необходимости переключаться. К примеру, чтобы создать группу настроек для тестового окружения, необходимо указать следующее:

$db['test']['hostname'] = "localhost";  
$db['test']['username'] = "root";  
$db['test']['password'] = "";  
$db['test']['database'] = "database\_name";  
$db['test']['dbdriver'] = "mysql";  
$db['test']['dbprefix'] = "";  
$db['test']['pconnect'] = TRUE;  
$db['test']['db\_debug'] = FALSE;  
$db['test']['cache\_on'] = FALSE;  
$db['test']['cachedir'] = "";  
$db['test']['char\_set'] = "utf8";  
$db['test']['dbcollat'] = "utf8\_general\_ci";

Теперь, чтобы система использовала эту группу, следует в конфигурационном файле указать:

$active\_group = "test";

Замечание: наименование "test" является случайным. Оно может быть и любым другим, какое вам понравится. По умолчанию мы используем слово "default" для основного соединения, но и оно может быть изменено на что-то более подходящее вашему проекту.

### Active Record

Класс [Active Record](http://code-igniter.ru/user_guide/database/active_record.html) включается или выключается на глобальном уровне посредством установки параметра $active\_record в значение TRUE или FALSE (boolean) в файле конфигурации баз данных. Если в коде вы не используете шаблон Active Record, то установка значения параметра в FALSE сэкономит ресурсы при инициализации класса работы с базами данных.

$active\_record = TRUE;

**Замечание:** некоторые классы фреймворка, такие как Sessions, требуют Active Records для обеспечения работы своего функционала.

### Пояснение значений:

* **hostname** - Наименование хоста вашего сервера баз данных. Чаще всего это "localhost".
* **username** - Имя пользователя для подключения к базе данных.
* **password** - Пароль для подключения к базе данных.
* **database** - Имя базы данных, к которой необходимо подключиться.
* **dbdriver** - Тип базы данных. К примеру: mysql, postgre, obdc и т.д. Должен указываться в нижнем регистре.
* **dbprefix** - Опциональный префикс таблиц, который добавляется к их именам при работе с запросами [Active Record](http://code-igniter.ru/user_guide/database/active_record.html). Это позволяет работать нескольким дистрибутивам CodeIgniter в пространстве имён одной базы данных.
* **pconnect** - TRUE/FALSE (boolean) - использовать ли постоянные соединения.
* **db\_debug** - TRUE/FALSE (boolean) - Должны ли ошибки базы данных выводиться на экран.
* **cache\_on** - TRUE/FALSE (boolean) - Включено ли кеширование запросов. Смотрите также [Кеширование запросов](http://code-igniter.ru/user_guide/database/caching.html).
* **cachedir** - Абсолютный путь на сервере к директории кеширования запросов.
* **char\_set** - Наименование кодировки, в которой работает база данных.
* **dbcollat** - Сопоставление символов, которое используется базой данных.
* **port** - Номер порта базы данных. Сейчас используется только в драйвере Postgre. Чтобы использовать это значение, вам необходимо добавить строку в файл конфигурации базы данных.

$db['default']['port'] = 5432;

**Замечание:**В зависимости от вашей платформы баз данных (MySQL, Postgre и т.д.) не все значения могут быть задействованы в работе. К примеру, если используется SQLite, то нет необходимости указывать имя пользователя и пароль, а именем базы будет физический путь к ней. По большей части, информация приведённая выше подразумевает, что вы используетеg MySQL.

# Подключение к базе данных

Существует два пути подключения к базе данных:

## Автоматическое подключение

Установка автоматического подключения обеспечивает подгрузку и инициализацию класса работы с базами данных при открытии каждой страницы. Чтобы включить функцию автоматического подключения, добавьте слово **database** к массиву в следующем файле:

**application/config/autoload.php**

## Ручное подключение

Если не все страницы требуют подключения к базе данных, то его можно выполнять вручную, добавив одну строку кода там где требуется обеспечить доступ. Эту строку кода можно добавить в конструктор класса, чтобы сделать подключение доступным в рамках всего класса.

$this->load->database();

Если данная функция **не содержит** никакой информации в качестве первого параметра, то подключение будет выполнено согласно настройкам по-умолчанию, указанным в файле конфигурации. Для большинства случаев это наиболее приемлемый вариант использования функции.

В качестве первого параметра функции **опционально** можно передавать наименование группы настроек подключения из вашего конфигурационного файла, а также, указывать параметры подключения напрямую, без внесения их в файл конфигурации. Примеры:

Чтобы указать конкретную группу настроек из файла конфигурации, можно выполнить следующее:

$this->load->database('**group\_name**');

Где **group\_name** это имя группы настроек из вашего файла конфигурации работы с базами данных.

Чтобы подключиться вручную к нужной базе, можно передать массив параметров:

$config['hostname'] = "localhost";  
$config['username'] = "myusername";  
$config['password'] = "mypassword";  
$config['database'] = "mydatabase";  
$config['dbdriver'] = "mysql";  
$config['dbprefix'] = "";  
$config['pconnect'] = FALSE;  
$config['db\_debug'] = TRUE;  
$this->load->database("**$config**");

За более подробной информацией по каждому из этих значений обращайтесь к странице [Конфигурация](http://code-igniter.ru/user_guide/database/configuration.html).

Также вы можете передать параметры подключения в виде DSN. Строка DSN должна быть составлена согласно образцу:

$dsn = 'dbdriver://username:password@hostname/database';  
$this->load->database("**$dsn**");

Обратите внимание, что при передаче DSN вы не сможете использовать некоторые значения по-умолчанию, которые могли бы задействовать в случае работы с конфигурационным массивом.

## Подключение к нескольким базам данных

Если требуется одновременное подключение к более чем одной базе данных, следует выполнить следующее:

$DB1 = $this->load->database('group\_one', TRUE);  
$DB2 = $this->load->database('group\_two', TRUE);

Замечание: измените слова "group\_one" и "group\_two" на необходимые названия групп из вашего файла конфигурации (или вы можете передать параметры напрямую, как показано выше).

Если установить второй параметр в TRUE (boolean), то функция вернёт объект базы данных.

Если вы подключаетесь таким образом, то необходимо использовать имя получаемого объекта для вызова функций, вместо синтаксиса, указанного в данном руководстве. Другими словами, вместо:

$this->db->query();  
$this->db->result();  
и т.д...

необходимо ставить:

$DB1->query();  
$DB1->result();  
и т.д...

# Запросы

## $this->db->query();

Чтобы отправить запрос, используйте следующую функцию:

$this->db->query('ЗДЕСЬ ВАШ ЗАПРОС');

Функция **query()** возвращает **объект** результатов выборки из базы данных (если выполняются запросы на чтение). Полученный объект можно использовать для [вывода результатов](http://code-igniter.ru/user_guide/database/results.html). Когда выполняются запросы на вставку или изменение данных, возвращается только TRUE или FALSE в зависимости от того, успешно или нет был выполнен запрос. В процессе получения данных, чаще всего, необходимо присваивать результаты запроса отдельной переменной. Пример:

**$query** = $this->db->query('ЗДЕСЬ ВАШ ЗАПРОС');

## $this->db->simple\_query();

Это упрощённая версия функции **$this->db->query()**. Она возвращает ТОЛЬКО TRUE или FALSE в зависимости от успешности выполнения запроса. Функция НЕ возвращает никакого набора данных и НЕ выполняет ничего кроме обычной отправки запроса к базе данных. Большинство пользователей использует эту функцию очень редко.

# Ручное добавление префиксов к базе данных

Если в конфигурации вы установили префикс и хотите вручную указать его в своём коде, то необходимо выполнить следующее:

$this->db->dbprefix('таблица');  
// выводит конструкцию "префикс\_таблица"

# Защита идентификаторов

В большинстве баз данных рекомендуется производить защиту имён таблиц и полей. Запросы Active Record защищены автоматически, но если вы хотите выполнить защиту идентификатора вручную, используйте следующее:

$this->db->protect\_identifier('имя\_таблицы');

# Экранирование в запросах

Экранирование данных в запросах перед отправкой к базе данных - это очень хороший стиль. В CodeIgniter есть две функции, которые вам помогут:

1. **$this->db->escape()** Эта функция определяет тип передаваемых данных, поэтому может экранировать только строки. Также, функция автоматически добавляет одинарные кавычки, поэтому вам не надо прибегать к дополнительным действиям:

$sql = "INSERT INTO table (title) VALUES(".$this->db->escape($title).")";

1. **$this->db->escape\_str()** Эта функция экранирует все поступающие в неё данные не взирая на их тип. Большую часть времени вы будете использовать скорее не эту, а предыдущую функцию. Пример использования:

$sql = "INSERT INTO table (title) VALUES('".$this->db->escape\_str($title)."')";

# Привязка переменных к запросам

Используя привязку переменных вы можете упростить синтаксис ваших запросов, позволяя системе самой подставить переменные в запрос. Рассмотрим следующий пример:

$sql = "SELECT \* FROM some\_table WHERE id = **?** AND status = **?** AND author = **?**";   
$this->db->query($sql, array(3, 'live', 'Rick'));

Знаки вопроса автоматически замещаются значениями переменных, переданных в виде массива вторым параметром функции запроса.

Дополнительным преимуществом использования привязки является автоматическое экранирование переменных, что в результате даёт более безопасные запросы. В таком случае вам не придётся вручную контролировать экранирование данных.

# Получение результатов запроса

Существует несколько способов получения результатов запроса:

## result()

Эта функция возвращает результаты запроса в виде массива **объектов**, или**пустой массив** в случае неудачи. Лучше всего такой результат обрабатывается перебором массива с помощью foreach:

$query = $this->db->query("Ваш запрос");  
foreach ($query->result() as $row)  
{  
   echo $row->title;  
   echo $row->name;  
   echo $row->body;  
}

Упомянутая **function** является псевдонимом **result\_object()**.

Если выполняется запрос, у которого возможен **пустой результат**, то узнать количество полученых записей можно так:

$query = $this->db->query("Ваш запрос");  
if ($query->num\_rows() > 0)  
{  
   foreach ($query->result() as $row)  
   {  
      echo $row->title;  
      echo $row->name;  
      echo $row->body;  
   }  
}

## result\_array()

Эта функция возвращает результат запроса, но только в виде массива (если результатов нет, то массив пустой). Удобно использовать foreach для перебора результатов:

$query = $this->db->query("Ваш запрос");  
foreach ($query->result\_array() as $row)  
{  
   echo $row['title'];  
   echo $row['name'];  
   echo $row['body'];  
}

## row()

Эта функция возвращает одну строку из результатов запроса. Если результат содержит несколько строк, то выбраной будет только первая. Результат возвращается сразу в виде **объекта**. Пример использования:

$query = $this->db->query("Ваш запрос");  
if ($query->num\_rows() > 0)  
{  
   $row = $query->row();   
   echo $row->title;  
   echo $row->name;  
   echo $row->body;  
}

Если вам нужна какая-то конкретная строка, вы можете указать её номер в качестве первого параметра:

$row = $query->row(**5**);

## row\_array()

Так же как и **row()**, данная функция возвращает одну строку результата, но только в виде массива. Пример:

$query = $this->db->query("Ваш запрос");  
if ($query->num\_rows() > 0)  
{  
   $row = $query->row\_array();   
   echo $row['title'];  
   echo $row['name'];  
   echo $row['body'];  
}

Если вам нужна какая-то конкретная строка, вы также можете указать её номер в качестве первого параметра:

$row = $query->row\_array(**5**);

Дополнительно, вы можете перемещаться вперёд/назад/к первому/к последнему элементу ваших результатов, используя следующие функции:

**$row = $query->first\_row()**  
**$row = $query->last\_row()**  
**$row = $query->next\_row()**  
**$row = $query->previous\_row()**

По умолчанию они возвращают объект, кроме тех случаев, когда в параметрах указано слово «array» (массив):

**$row = $query->first\_row('array')**  
**$row = $query->last\_row('array')**  
**$row = $query->next\_row('array')**  
**$row = $query->previous\_row('array')**

# Хелперы для обработки результатов выборки

## $query->num\_rows()

Количество строк, полученых по выполнению запроса. Замечание: в данном примере, **$query** это переменная, которой присваиваются результаты выполнения запроса в виде объекта:

$query = $this->db->query('SELECT \* FROM my\_table');  
echo $query->num\_rows();

## $query->num\_fields()

Количество ПОЛЕЙ (колонок), возвращаемых по выполнению запроса. Убедитесь, что вызываете функцию для объекта результатов запроса:

$query = $this->db->query('SELECT \* FROM my\_table');  
echo $query->num\_fields();

## $query->free\_result()

Функция высвобождает память, предназначенную для хранения результатов выборки и удаляет идентификатор ресурса. Обычно, PHP автоматически высвобождает память по окончанию работы скрипта, но если вы выполняете большое количество запросов в рамках одного скрипта, то в целях экономии ресурсов есть смысл очищать память после получения каждого результата. Пример:

$query = $this->db->query('SELECT title FROM my\_table');  
foreach ($query->result() as $row)  
{  
   echo $row->title;  
}  
$query->free\_result(); // Объект $query более не будет доступен для работы с результатами выборки  
$query2 = $this->db->query('SELECT name FROM some\_table');  
$row = $query2->row();  
echo $row->name;  
$query2->free\_result(); // Объект $query2 более не будет доступен для работы с результатами выборки

# Функции хелпера запросов

## $this->db->insert\_id()

Идентификатор последней вставки записи в базу данных.

## $this->db->affected\_rows()

Отображает количество записей, на которые повлиял запрос (при выполнении вставок или правок).

Замечание: в MySQL "DELETE FROM TABLE" возвращает 0 затронутых строк. В класс для работы с базами данных встроен небольшой модификатор, который позволяет возвращать правильное количество затронутых записей. По умолчанию этот модификатор доступен, но может быть выключен в файле драйвера к базе данных.

## $this->db->count\_all();

Позволяет определять количество строк в конкретной таблице. Передавайте название таблицы в качестве первого параметра. Пример:

echo $this->db->count\_all('**my\_table**');  
// Возвращает целое число, к примеру 25

## $this->db->platform()

Выводит наименование платформы СУБД, с которой вы взаимодействуете (MySQL, MS SQL, Postgre, и т.д.):

echo $this->db->platform();

## $this->db->version()

Выводит версию используемой СУБД:

echo $this->db->version();

## $this->db->last\_query();

Возвращает последний выполненый запрос (строку запроса, но не объект с результатами). Пример:

$str = $this->db->last\_query();  
// Создаёт: SELECT \* FROM sometable....

Две следующие функции помогут упростить процесс записи в базу данных (INSERT и UPDATE).

## $this->db->insert\_string();

Функция позволяет упростить процесс выполнения вставок. Возвращает корректно сформированую строку SQL-запроса на вставку. Пример:

$data = array('name' => $name, 'email' => $email, 'url' => $url);  
$str = $this->db->insert\_string('table\_name', $data);

Первый параметр - наименование таблицы, второй параметр - ассоциативный массив с данными для вставки. Предыдущий пример создаёт:

INSERT INTO table\_name (name, email, url) VALUES ('Rick', 'rick@your-site.com', 'www.your-site.com')

Замечание: Значения автоматически экранируются, создавая более безопасные запросы.

## $this->db->update\_string();

Данная функция упрощает процесс записи правок к данным. Она возвращает корректно сформированый SQL-запрос на выполнение правки. Пример:

$data = array('name' => $name, 'email' => $email, 'url' => $url);  
$where = "author\_id = 1 AND status = 'active'";   
$str = $this->db->update\_string('table\_name', $data, $where);

Первый параметр - наименование таблицы, второй параметр - ассоциативный массив с данными для вставки и в третьем параметре указано условие "where". Предыдущий пример приводит к результату:

UPDATE exp\_weblog SET name = 'Rick', email = 'rick@your-site.com', url = 'www.your-site.com' WHERE author\_id = 1 AND status = 'active'

Замечание: Значения автоматически экранируются, создавая более безопасные запросы.

# Класс Active Record

CodeIgniter использует модифицированную версию паттерна Active Record Database. Этот паттерн позволяет извлекать, вставлять и обновлять информацию базы данных с помощью написания минимального количества кода. В некоторых случаях для выполнения действия с базой данных достаточно одной или двух строк кода. CodeIgniter не требует наличия отдельного файла с классом, соответствующего каждой таблице базы данных. Вместо этого CodeIgniter предоставляет более простой интерфейс.

Помимо простоты, главная особенность использования функционала Active Record заключается в том, что он позволяет вам создавать приложения независимые от какой-то конкретной базы данных, поскольку синтаксис запросов генерируется адаптером соответствующей базы данных. Так же он позволяет делать запросы более безопасными за счет автоматического экранирования значений.

**Примечание:** Если вы собираетесь писать свои собственные запросы, то вы можете отключить этот класс в вашем конфигурационном файле, тем самым позволив родной библиотеке базы данных и адаптеру использовать меньше системных ресурсов.

* [Выбор данных](http://code-igniter.ru/user_guide/database/active_record.html#select)
* [Вставка данных](http://code-igniter.ru/user_guide/database/active_record.html#insert)
* [Обновление данных](http://code-igniter.ru/user_guide/database/active_record.html#update)
* [Удаление данных](http://code-igniter.ru/user_guide/database/active_record.html#delete)
* [Метод сцепления данных](http://code-igniter.ru/user_guide/database/active_record.html#chaining)
* [Кэширование Active Record](http://code-igniter.ru/user_guide/database/active_record.html#caching)

# Выбор данных

Следующие функции позволяют создавать SQL операторы **SELECT**.

**Примечание: Если вы используете PHP 5, то можете использовать метод сцепления данных для получения более компактной записи. Этот метод описан в конце страницы.**

## $this->db->get();

Выполняет запрос выбора данных и возвращает результат. Сама по себе может быть использована для выбора всех записей из таблицы:

$query = $this->db->get('mytable');  
// Генерирует: SELECT \* FROM mytable

Второй и третий параметры позволяют задавать количество извлекаемых записей и начальное смещение:

$query = $this->db->get('mytable', 10, 20);  
// Генерирует: SELECT \* FROM mytable LIMIT 20, 10 (для MySQL. В других базах данных синтаксис будет немного отличаться.)

Заметьте, что вышеприведенная функция связана с переменной **$query**, которая может быть использована для отображения результатов запроса:

$query = $this->db->get('mytable');  
foreach ($query->result() as $row)  
{  
    echo $row->title;  
}

За более подробной информацией относительно генерирования результатов обратитесь к странице [функции результатов](http://code-igniter.ru/user_guide/database/results.html).

## $this->db->get\_where();

Идентична вышеприведенной функции за исключением того, что она позволяет с помощью второго параметра добавить к запросу оператор «where», не прибегая к использованию функции db->where():

$query = $this->db->get\_where('mytable', array('id' => $id), $limit, $offset);

Более подробная информация о функции where приведена ниже.

Примечание: get\_where() ранее была известна как getwhere().

## $this->db->select();

Позволяет задать имена выбираемых полей в вашем SELECT запросе:

$this->db->select('title, content, date');  
$query = $this->db->get('mytable');  
// Генерирует: SELECT title, content, date FROM mytable

**Примечание:** Если вы выбираете все (\*) поля таблицы, то вам нет необходимости использовать эту функцию. Когда она опущена, CodeIgniter полагает, что вы хотите выполнить SELECT \*.

$this->db->select() принимает второй необязательный параметр. Если вы установите его в FALSE, то CodeIgniter не будет пытаться обезопасить имена полей или таблиц с помощью одинарных кавычек. Это удобно, если вам нужно создать вложенный оператор select.

$this->db->select('(SELECT SUM(payments.amount) FROM payments WHERE payments.invoice\_id=4') AS amount\_paid', FALSE);   
$query = $this->db->get('mytable');

## $this->db->select\_max();

Записывает «SELECT MAX(field)» в ваш запрос. Вы также можете во втором необязательном параметре указать новое имя результирующего поля.

$this->db->select\_max('age');  
$query = $this->db->get('members');  
// Генерирует: SELECT MAX(age) as age FROM members  
$this->db->select\_max('age', 'member\_age');  
$query = $this->db->get('members');  
// Генерирует: SELECT MAX(age) as member\_age FROM members

## $this->db->select\_min();

Записывает «SELECT MIN(field)» в ваш запрос. Так же как и в **select\_max()**, вы можно задать второй необязательный параметр, для указания нового имени результирующего поля.

$this->db->select\_min('age');  
$query = $this->db->get('members');  
// Генерирует: SELECT MIN(age) as age FROM members

## $this->db->select\_avg();

Записывает «SELECT AVG(field)» в ваш запрос. Так же как и в **select\_max()**, вы можно задать второй необязательный параметр, для указания нового имени результирующего поля.

$this->db->select\_avg('age');  
$query = $this->db->get('members');  
// Генерирует: SELECT AVG(age) as age FROM members

## $this->db->select\_sum();

Записывает «SELECT SUM(field)» в ваш запрос. Так же как и в **select\_max()**, вы можно задать второй необязательный параметр, для указания нового имени результирующего поля.

$this->db->select\_sum('age');  
$query = $this->db->get('members');  
// Генерирует: SELECT SUM(age) as age FROM members

## $this->db->from();

Позволяет задать FROM часть вашего запроса:

$this->db->select('title, content, date');  
$this->db->from('mytable');  
$query = $this->db->get();  
// Генерирует: SELECT title, content, date FROM mytable

Примечание: Как было показано ранее, FROM часть запроса можно задать и с помощью функции **$this->db->get()**, так что используйте любой метод, который вам больше нравится.

## $this->db->join();

Позволяет задать JOIN часть вашего запроса:

$this->db->select('\*');  
$this->db->from('blogs');  
$this->db->join('comments', 'comments.id = blogs.id');  
$query = $this->db->get();  
// Генерирует:   
// SELECT \* FROM blogs  
// JOIN comments ON comments.id = blogs.id

Возможно несколько вызовов функции, если необходимо сделать несколько объединений в одном запросе.

Если же вам нужно что-то отличное от обычного JOIN, вы может сделать это посредством третьего параметра функции. Допустимые значения: left, right, outer, inner, left outer и right outer.

$this->db->join('comments', 'comments.id = blogs.id', **'left'**);  
// Генерирует: LEFT JOIN comments ON comments.id = blogs.id

## $this->db->where();

Эта функция позволяет задать оператор **WHERE** с помощью одного из четырех способов:

**Примечание:** Для получения более безопасных запросов, все значения переданные в эту функцию автоматически заключаются в одинарные кавычки.

1. **Простой метод ключ/значение:**

$this->db->where('name', $name);   
// Генерирует: WHERE name = 'Joe'

Заметьте, что знак равенства добавляется автоматически.

Если производится несколько вызовов функции, то их результаты связывается друг с другом с помощью **AND**:

$this->db->where('name', $name);  
$this->db->where('title', $title);  
$this->db->where('status', $status);   
// WHERE name = 'Joe' AND title = 'boss' AND status = 'active'

1. **Более гибкий метод ключ/значение:**

Для задания собственного метода сравнения, вы можете включить необходимый оператор в первый параметр:

$this->db->where('name !=', $name);  
$this->db->where('id <', $id);   
// Генерирует: WHERE name != 'Joe' AND id < 45

1. **Метод использования ассоциативного массива:**

$array = array('name' => $name, 'title' => $title, 'status' => $status);  
$this->db->where($array);   
// Генерирует: WHERE name = 'Joe' AND title = 'boss' AND status = 'active'

Так же вы можете ипользовать собственные операторы сравнения:

$array = array('name !=' => $name, 'id <' => $id, 'date >' => $date);  
$this->db->where($array);

1. **Обычная строка:**

Вы можете вручную написать свой собственный оператор:

$where = "name='Joe' AND status='boss' OR status='active'";  
$this->db->where($where);

$this->db->where() допускает использование третьего необязательного параметра. Если вы установите его в FALSE, то CodeIgniter не будет пытаться обезопасить имена полей или таблиц с помощью одинарных кавычек.

$this->db->where('MATCH (field) AGAINST ("value")', NULL, FALSE);

## $this->db->or\_where();

Эта функция идентична предыдущей за тем исключением, что при нескольких вызовах, результаты объединяются с помощью OR:

$this->db->where('name !=', $name);  
$this->db->or\_where('id >', $id);   
// Генерирует: WHERE name != 'Joe' OR id > 50

Примечание: or\_where() ранее была известна как orwhere().

## $this->db->where\_in();

Генерирует SQL запрос WHERE field IN ('item', 'item') присоединяя его при необходимости с помощью AND

$names = array('Frank', 'Todd', 'James');  
$this->db->where\_in('username', $names);  
// Генерирует: AND WHERE username IN ('Frank', 'Todd', 'James')

## $this->db->or\_where\_in();

Генрирует SQL запрос WHERE field IN ('item', 'item') присоеденяя его при необходимости с помощью OR

$names = array('Frank', 'Todd', 'James');  
$this->db->or\_where\_in('username', $names);  
// Генерирует: OR WHERE username IN ('Frank', 'Todd', 'James')

## $this->db->where\_not\_in();

Генрирует SQL запрос WHERE field NOT IN ('item', 'item') присоеденяя его при необходимости с помощью AND

$names = array('Frank', 'Todd', 'James');  
$this->db->where\_not\_in('username', $names);  
// Генерирует: AND WHERE username NOT IN ('Frank', 'Todd', 'James')

## $this->db->or\_where\_not\_in();

Генрирует SQL запрос WHERE field NOT IN ('item', 'item') присоеденяя его при необходимости с помощью OR

$names = array('Frank', 'Todd', 'James');  
$this->db->or\_where\_not\_in('username', $names);  
// Генерирует: OR WHERE username NOT IN ('Frank', 'Todd', 'James')

## $this->db->like();

Эта функция позволяет генерировать оператор **LIKE**, используемый при поиске данных.

**Примечание:** Все значения переданные в эту функцию автоматически заключаются в одинарные кавычки.

1. **Простой метод ключ/значение:**

$this->db->like('title', 'match');   
// Генерирует: WHERE title LIKE '%match%'

Если вы производите несколько вызовов функции, то их результаты объединяются с помощью **AND**:

$this->db->like('title', 'match');  
$this->db->like('body', 'match');   
// WHERE title LIKE '%match%' AND body LIKE '%match%

Если вы хотите контроллировать расположение специального символа (%), то вы можете использовать третий необязательный параметр. Доступные значения: 'before', 'after' и 'both' (по-умолчанию).

$this->db->like('title', 'match', 'before');   
// Генерирует: WHERE title LIKE '%match'   
$this->db->like('title', 'match', 'after');   
// Генерирует: WHERE title LIKE 'match%'   
  
$this->db->like('title', 'match', 'both');   
// Генерирует: WHERE title LIKE '%match%'

1. **Метод использования асоциативного массива:**

$array = array('title' => $match, 'page1' => $match, 'page2' => $match);  
$this->db->like($array);   
// WHERE title LIKE '%match%' AND page1 LIKE '%match%' AND page2 LIKE '%match%'

## $this->db->or\_like();

Эта функция идентична предыдущей за тем исключением, что при нескольких вызовах результаты объединяются с помощью OR:

$this->db->like('title', 'match');  
$this->db->or\_like('body', $match);   
// WHERE title LIKE '%match%' OR body LIKE '%match%'

Примечание: or\_like() ранее была известна как orlike().

## $this->db->not\_like();

Эта функция идентична **like()** за тем исключением, что она генерирует NOT LIKE оператор:

$this->db->not\_like('title', 'match');  
// WHERE title NOT LIKE '%match%

## $this->db->or\_not\_like();

Эта функция идентична **not\_like()** за тем исключением, что при нескольких вызовах результаты объединяются с помощью OR:

$this->db->like('title', 'match');  
$this->db->or\_not\_like('body', 'match');   
// WHERE title LIKE '%match% OR body NOT LIKE '%match%'

## $this->db->group\_by();

Позволяет задавать GROUP BY часть вашего запроса:

$this->db->group\_by("title");   
// Генерирует: GROUP BY title

Вы так же можете передать массив с несколькими значениями:

$this->db->group\_by(array("title", "date"));   
// Генерирует: GROUP BY title, date

Примечание: group\_by() ранее была известна как groupby().

## $this->db->distinct();

Добавляет ключевое слово «DISTINCT» в запрос

$this->db->distinct();  
$this->db->get('table');  
// Генерирует: SELECT DISTINCT \* FROM table

## $this->db->having();

Позволяет задавать HAVING часть запроса. Возможно 2 вида синтаксиса, с одним аргументом или с двумя:

$this->db->having('user\_id = 45');   
// Генерирует: HAVING user\_id = 45  
$this->db->having('user\_id', 45);   
// Генерирует: HAVING user\_id = 45

Вы так же можете передать массив с несколькими значениями:

$this->db->having(array('title =' => 'My Title', 'id <' => $id));   
// Генерирует: HAVING title = 'My Title', id < 45

Если вы используете базу данных, для которой CodeIgniter заключает запросы в одинарные кавычки, то вы можете отключить заключение содержания в одинарные кавычки, установив третий необязательный аргумент в FALSE.

$this->db->having('user\_id', 45);   
// Генерирует: HAVING `user\_id` = 45 в некоторых базах данных, таких как MySQL   
$this->db->having('user\_id', 45, FALSE);   
// Генерирует: HAVING user\_id = 45

## $this->db->or\_having();

Идентична having(), только несколько операторов разделяются «OR».

## $this->db->order\_by();

Позволяет задавать оператор ORDER BY. Первый параметр содержит имя столбца по которому необходимо упорядочить результаты. Второй параметр позволяет задать направление упорядочивания. Возможные значения: **asc**,**desc** или **random**.

$this->db->order\_by("title", "desc");   
// Генерирует: ORDER BY title DESC

Вы так же можете передать первым параметром собственную сроку:

$this->db->order\_by('title desc, name asc');   
// Генерирует: ORDER BY title DESC, name ASC

Или вызвать функцию несколько раз, если необходимо упорядочить результаты по нескольким полям.

$this->db->order\_by("title", "desc");  
$this->db->order\_by("name", "asc");   
// Генерирует: ORDER BY title DESC, name ASC

Примечание: order\_by() ранее была известна как orderby().

Примечание: Упорядочивание типа random не поддерживается Oracle или MSSQL драйверами. И по-умолчанию будет установленно в «ASC».

## $this->db->limit();

Позволяет задать ограничение на количество строк возвращаемых в результате запроса:

$this->db->limit(10);  
// Генерирует: LIMIT 10

Второй параметр позволяет задать смещение с которого будут отсчитываться выводимые результаты.

$this->db->limit(10, 20);  
// Генерирует: LIMIT 20, 10 (в MySQL. Другие базы данных имеют немного другой синтаксис)

## $this->db->count\_all\_results();

Позволяет определить количество строк в текущем запросе Active Record. Запросы будут содержать ограничения Active Record, такие как where(), or\_where(), like(), or\_like(), и т. д. Пример:

echo $this->db->count\_all\_results('**my\_table**');  
// Возвращает целое число, например 25  
$this->db->like('title', 'match');  
$this->db->from('**my\_table**');  
echo $this->db->count\_all\_results();  
// Возвращает целое число, наприме 17

## $this->db->count\_all();

Позволяет определить количество строк в текущей таблице. Принимает имя таблицы в качестве первого параметра. Пример:

echo $this->db->count\_all('**my\_table**');  
// Возвращает целое число, наприме 25

# Вставка данных

## $this->db->insert();

На основе предоставленных данных генерирут запрос вставки в выполняет его. Так же в функцию можно передать массив или объект. Вот пример использования массива:

$data = array(  
               'title' => 'My title' ,  
               'name' => 'My Name' ,  
               'date' => 'My date'  
            );  
$this->db->insert('mytable', $data);   
// Генерирует: INSERT INTO mytable (title, name, date) VALUES ('My title', 'My name', 'My date')

Первый параметр содержит имя таблицы, второй — ассоциативцый массив значений.

А вот пример использования объекта:

/\*  
    class Myclass {  
        var $title = 'My Title';  
        var $content = 'My Content';  
        var $date = 'My Date';  
    }  
\*/  
$object = new Myclass;  
$this->db->insert('mytable', $object);   
// Генерирует: INSERT INTO mytable (title, content, date) VALUES ('My Title', 'My Content', 'My Date')

Первый параметр содержит имя таблицы, второй — объект.

**Примечание:** Для генерации более безопасных запросов все значения автоматически заключаются в одинарные кавычки.

## $this->db->set();

Эта функция позволяет задавать значения для **вставляемых (insert)** или**обновляемых (update)** данных.

**Она может использоваться вместо непосредственной передачи массива данных в функцию insert или update:**

$this->db->set('name', $name);   
$this->db->insert('mytable');   
// Генерирует: INSERT INTO mytable (name) VALUES ('{$name}')

Если вы производите несколько вызовов функции, то метод их объединения будет определяться исходя из того, какой запрос вы производите — insert или update:

$this->db->set('name', $name);  
$this->db->set('title', $title);  
$this->db->set('status', $status);  
$this->db->insert('mytable');

**set()** также позволяет задать третий необязательный параметр ($escape), который, будучи установленным в FALSE, предотвращает заключение данных в одинарные кавычки. Для иллюстрации различий, ниже показан результат вызова set() с третьим параметром и без него.

$this->db->set('field', 'field+1', FALSE);  
$this->db->insert('mytable');   
// получается INSERT INTO mytable (field) VALUES (field+1)  
$this->db->set('field', 'field+1');  
$this->db->insert('mytable');   
// получатся INSERT INTO mytable (field) VALUES ('field+1')

Вы также можете передать в функцию ассоциативный массив:

$array = array('name' => $name, 'title' => $title, 'status' => $status);  
$this->db->set($array);  
$this->db->insert('mytable');

... или объект:

/\*  
    class Myclass {  
        var $title = 'My Title';  
        var $content = 'My Content';  
        var $date = 'My Date';  
    }  
\*/  
$object = new Myclass;  
$this->db->set($object);  
$this->db->insert('mytable');

# Обновление данных

## $this->db->update();

На основании предоставленных данных генерирует и выполняет запрос обновления данных. Вы можете передать в функцию массив или объект. Вот пример использования массива:

$data = array(  
               'title' => $title,  
               'name' => $name,  
               'date' => $date  
            );  
$this->db->where('id', $id);  
$this->db->update('mytable', $data);   
// Генерирует:  
// UPDATE mytable   
// SET title = '{$title}', name = '{$name}', date = '{$date}'  
// WHERE id = $id

Или вы можете передать в функцию объект:

/\*  
    class Myclass {  
        var $title = 'My Title';  
        var $content = 'My Content';  
        var $date = 'My Date';  
    }  
\*/  
$object = new Myclass;  
$this->db->where('id', $id);  
$this->db->update('mytable', $object);   
// Produces:  
// UPDATE mytable   
// SET title = '{$title}', name = '{$name}', date = '{$date}'  
// WHERE id = $id

**Примечание:** Для генерации более безопасных запросов все значения автоматически заключаются в одинарные кавычки.

Заметьте, вы можете использовать функцию **$this->db->where()**, для задания оператора WHERE. При желании вы можете передать эту информацию непосредственно в функцию update в виде строки:

$this->db->update('mytable', $data, "id = 4");

... или массива:

$this->db->update('mytable', $data, array('id' => $id));

Также вы можете использовать описанную выше функцию **$this->db->set()**для задания обновляемых полей.

# Удаление данных

## $this->db->delete();

Генерирует SQL запрос delete и выполняет его.

$this->db->delete('mytable', array('id' => $id));   
// Генерирует:  
// DELETE FROM mytable   
// WHERE id = $id

Первый параметр — имя таблицы, второй — оператор where. Вы также можете использовать функции **where()** или **or\_where()** вместо непосредственной передачи данных во второй параметр функции:

$this->db->where('id', $id);  
$this->db->delete('mytable');   
// Генерирует:  
// DELETE FROM mytable   
// WHERE id = $id

Если вы хотите удалить данные из более чем одной таблицы, то вы можете передать в функцию массив с именами таблиц.

$tables = array('table1', 'table2', 'table3');  
$this->db->where('id', '5');  
$this->db->delete($tables);

Если вы хотите удалить все данные из таблицы, то можете использовать функцию **truncate()** или **empty\_table()**.

## $this->db->empty\_table();

Генерирует SQL запрос delete и выполняет его.

$this->db->empty\_table('mytable');   
// Генерирует  
// DELETE FROM mytable

## $this->db->truncate();

Генерирует SQL запрос truncate и выполняет его.

$this->db->from('mytable');   
$this->db->truncate();   
// or   
$this->db->truncate('mytable');   
// Генерирует:  
// TRUNCATE mytable 

**Примечание:** Если команда TRUNCATE не доступна, truncate() выполняется как «DELETE FROM table».

# Метод сцепления данных

Метод сцепления данных позволяет вам упростить соединение результатов нескольких функций. Рассмотрим следующий пример:

**$this->db->select**('title')**->from**('mytable')**->where**('id', $id)**->limit**(10, 20);  
$query = $this->db->get();

**Примечание:** Метод сцепления данных работает только в PHP 5.

# Кэширование Active Record

Пока отсутствует «настоящее» кэширование, Active Record позволяет сохранять (или «кэшировать») определенные части ваших запросов, для последующего их использования. Обычно, когда завершается вызов Active Record, то вся хранящаяся информация сбрасывается для выполнения следующего вызова. Кэширование позволяет предотвратить это сбрасывание и сделать повторное использование информации более простым.

Кэшированные запросы кумулятивны. Если вы делаете 2 кэшированных вызова select() и 2 некэшированных вызова select(), то в результате получите 4 вызова select(). Доступны три функции кэширования:

## $this->db->start\_cache()

Эта функция должна быть вызванна для начала кэширования. Все Active Record запросы определенных типов (см. ниже поддерживаемые запросы) будут сохраненны для последующего использования.

## $this->db->stop\_cache()

Эта функция может быть вызванна для прекращения кэширования.

## $this->db->flush\_cache()

Эта функция удаляет все элементы из кэша Active Record.

Вот пример использования:

$this->db->start\_cache();  
$this->db->select('field1');  
$this->db->stop\_cache();  
$this->db->get('tablename');  
// В итоге:  
// SELECT `field1` FROM (`tablename`)  
$this->db->select('field2');  
$this->db->get('tablename');  
// В итоге:  
// SELECT `field1`, `field2` FROM (`tablename`)  
$this->db->flush\_cache();  
$this->db->select('field2');  
$this->db->get('tablename');  
// В итоге:  
// SELECT `field2` FROM (`tablename`)

**Примечание:** Кэшироваться могут следующие поля: ‘select’, ‘from’, ‘join’, ‘where’, ‘like’, ‘groupby’, ‘having’, ‘orderby’, ‘set’

# Транзакции

Абстракция базы данных CodeIgniter'а позволяет использовать **транзакции** с базами данных, которые поддерживают транзакционно-безопасные типы таблиц. В MySQL для этого необходимо использовать таблицы типов InnoDB или BDB, вместо более общих MyISAM. Большинство других платформ баз данных поддерживают родной механизм транзакций.

Если вы не очень хорошо знакомы с транзакциями, то мы рекомендуем вам найти хороший онлайн ресурс и почитать про использование транзакций в вашей базе данных. Информация ниже предполагает, что вы понимаете основы механизма транзакций.

## Подход CodeIgniter'а при работе с транзакциям

CodeIgniter использует подход работы с транзакциям, который очень похож на подход используемый в популярном классе базы данных ADODB. Мы выбрали это подход потому, что он очень упрощает процесс работы с транзакциями. В большинстве случаев все, что неоходимо — это написать две строчки кода.

Традиционно, реализация механизма транзакции требует достаточно большого объема работы, поскольку необходимо отслеживать выполнение всех запросов транзакции и определять, когда следует **завершить**транзакцию, а когда **откатить**. Это особенно затруднительно, если приходится работать с вложенными запросами. В отличии от этого, мы реализовали систему умных транзакций, которая делает все это автоматически (тем не менее вы можете управлять транзакциями вручную, но в этом нет никакой выгоды).

## Обработка транзакций

Для обработки запросов с использованием транзакций, необходимо использовать функции **$this->db->trans\_start()** и **$this->db->trans\_complete()**, как показано ниже:

**$this->db->trans\_start();**  
$this->db->query('AN SQL QUERY...');  
$this->db->query('ANOTHER QUERY...');  
$this->db->query('AND YET ANOTHER QUERY...');  
**$this->db->trans\_complete();**

Вы можете ипользовать любое количество запросов между начальной и завершающей фунциями и они все будут либо выполнены, либо нет, в зависимости от того, будет ли выполнен или нет каждый из представленных запросов.

## Строгий режим (Strict Mode)

По-умолчанию CodeIgniter выполняет все транзакции в **строгом режиме**. Если включен строгий режим и выполняется несколько транзакций, то в случае неудачного завершения одной из транзакций, произойдет откат всех транзакций. Если строгий режим отключен, то каждая транзакция будет обработываться независимо от остальных, т. е. неудачное завершение одной из транзакций никак не повлияет на обработку остальных.

Строгий режим может быть отключен следующим образом:

$this->db->trans\_strict(FALSE);

## Управление ошибками

Если в файле **config/database.php** включен режим уведомлени об ошибках, то в случае неудачного завершение транзакции вы увидите стандартное сообщение об ошибке. Если режим отладки отключен, то вы можете управлять ошибками самостоятельно, например так:

$this->db->trans\_start();  
$this->db->query('AN SQL QUERY...');  
$this->db->query('ANOTHER QUERY...');  
$this->db->trans\_complete();  
if (**$this->db->trans\_status()** === FALSE)  
{  
    // отобразите ошибку... или используейте функцию log\_message() для добавления ошибки в лог  
}

## Включение транзакций

Транзакции включаются автоматически в момент использовании **$this->db->trans\_start()**. Если вы хотите отключить транзакции, то вы можете сделать это с помощью **$this->db->trans\_off()**:

**$this->db->trans\_off()**  
$this->db->trans\_start();  
$this->db->query('AN SQL QUERY...');  
$this->db->trans\_complete();

Когда транзакции отключенны, запросы автоматически завершаются, как если бы они выполнялись без использования транзакций.

## Режим тестирования (Test Mode)

При желании можно перевести систему транзакций в «режим тестирования», в котором запросы откатываются даже, если они сгенерировали верный результат. Для использования режима тестирования, просто установите первый параметр функции **$this->db->trans\_start()** в **TRUE**:

$this->db->trans\_start(**TRUE**); // Будет происходить откат запросов  
$this->db->query('AN SQL QUERY...');  
$this->db->trans\_complete();

## Выполнение транзакций вручную

Если вы хотите выполянять транзакции вручную, то вы можете сделать это следующим образом:

$this->db->trans\_begin();  
$this->db->query('AN SQL QUERY...');  
$this->db->query('ANOTHER QUERY...');  
$this->db->query('AND YET ANOTHER QUERY...');  
  
if ($this->db->trans\_status() === FALSE)  
{  
    $this->db->trans\_rollback();  
}  
else  
{  
    $this->db->trans\_commit();  
}

**Примечание:** При выполнении транзакций вручную, убедитесь, что вы используете именно **$this->db->trans\_begin()**, а НЕ **$this->db->trans\_start()**.

# Данные таблицы

Эти функции позволяют получать данные о таблицах базы.

## $this->db->list\_tables();

Возвращает массив, содержащий имена всех таблиц в текущей базе данных. Пример:

$tables = $this->db->list\_tables();  
foreach ($tables as $table)  
{  
   echo $table;  
}

## $this->db->table\_exists();

Иногда бывает полезным проверять, существует ли некоторая таблица, прежде чем выполнять запросы к ней. Возвращает булево значение TRUE/FALSE. Пример использования:

if ($this->db->table\_exists('table\_name'))  
{  
   // some code...  
}

Замечание: Замените table\_name на имя искомой таблицы.

# Метаданные полей

## $this->db->list\_fields()

Возвращает массив, содержащий имена полей. Этот запрос может быть вызван двумя способами:

1. Можно указать имя таблицы и вызвать запрос через объект **$this->db->**:

$fields = $this->db->list\_fields('table\_name')  
foreach ($fields as $field)  
{  
   echo $field;  
}

2. Можно получить имена полей связанных с выполняющимся запросом, вызвав функцию из объекта результатов запроса:

$query = $this->db->query('SELECT \* FROM some\_table')   
foreach ($query->list\_fields() as $field)  
{  
   echo $field;  
}

## $this->db->field\_exists()

Иногда перед выполнением действия полезно знать существует ли определенное поле. Возврящает TRUE/FALSE (boolean). Пример использования:

if ($this->db->field\_exists('field\_name', 'table\_name'))  
{  
   // какой-то код...  
}

Примечание: Замените field\_name на искомое имя столбца, а также заменитеtable\_name на искомое имя таблицы.

## $this->db->field\_data()

Возвращает массив объектов содержащих информацию о поле.

Иногда полезно получить имена полей или другие метаданные, вроде типа столбца, максимальной длины и т. д.

Примечание: Не все базы данных предоставляют метаданные.

Пример ипользования:

$fields = $this->db->field\_data('table\_name')  
foreach ($fields as $field)  
{  
   echo $field->name;  
   echo $field->type;  
   echo $field->max\_length;  
   echo $field->primary\_key;  
}

Если запрос уже выполнен, то вместо указания имени таблицы, можно использовать объект результатов запроса:

$query = $this->db->query("YOUR QUERY")  
$fields = $query->field\_data()

С помощью этой функци можно получить следующие данные, если они подерживаются вашей базой данных:

* name - имя столбца
* max\_length - максимальная длина столбца
* primary\_key - 1, если столбец является первичным ключом
* type - тип столбца

# Вызов произвольных функций

## $this->db->call\_function();

Данная функция позволяет вызывать отсутствующие в CodeIgniter функции PHP для работы с базой данных, соблюдая платформонезависимый стиль. К примеру, вам нужно вызвать функцию **mysql\_get\_client\_info()**, которая **не поддерживается** фреймворком CodeIgniter. Для этого выполняем:

$this->db->call\_function('**get\_client\_info**');

Вам необходимо передать имя функции, **без** префикса **mysql\_** в первом параметре. Префикс добавляется автоматически, на основе того драйвера, который в данный момент используется. Такой подход позволяет задействовать одни и те же функции на разных СУБД. Конечно, не все функции для разных платформ называются в одинаковом стиле, поэтому данный способ не обеспечивает полноценной кроссплатформенности.

Любые параметры, необходимые для выполнения вызываемой функции передаются как второй и последующие параметры.

$this->db->call\_function('**some\_function**', $param1, $param2...);

Зачастую, необходимо указывать ID подключения или ID результата выборки. ID подключения можно получить так:

$this->db->conn\_id;

В свою очередь, ID результата выборки можно получить, используя объект результатов. К примеру:

$query = $this->db->query("Ваш запрос");  
**$query->result\_id;**

# Класс кэширования базы данных

Класс кэширования базы данных позволяет кэшировать запросы в текстовых файлах для снижения нагрузки на базу данных.

**Важное замечание:**  Этот класс инициализируется автоматически драйвером базы данных, когда включена поддержка кэширования. Не загружайте этот класс вручную.  
  
**Также следует отметить:**  При кэшировании возможно использование НЕ всех функций результатов запроса. Пожалуйста, читайте эту страницу внимательнее.

## Включение кэширования

Кэширование включается в три шага:

* Создайте директорию на сервере, с правами на запись, где будут храниться кэш-файлы.
* Пропишите путь к папке с кэшем в файле **application/config/database.php**.
* Включите функции кэширования либо глобально, установив соответствующую настройку в файле **application/config/database.php**, либо вручную, как описанно ниже.

Однажды включенное, кэширование будет происходить автоматически всякий раз при загрузке страницы, содержащей запросы к базе данных.

## Как работает кэширование?

Кэширования запроса CodeIgniter'а происходит динамически при просмотре страниц. Когда кэширование включено, то при первой загрузке страницы объект результата запроса сериализуется и сохраняется в текстовом файле на сервере. При последующей загрузке страницы, вместо подключения к базе данных, загружается кэш-файл. Таким обазом, использование базы данных может быть фактически сниженно до нуля при использовании кешированных страниц.

Кэшироваться могут только запросы на **чтение** (SELECT) потому, что только запросы этого типа генерируют результат. Запросы на **запись** (INSERT, UPDATE и т. д.) не могут кэшироваться системой потому, что не генерируют результат.

Срок жизни кэш-файлов неограничен. Любые запросы, будучи закэшированными, остаются кэшированными до тех пор, пока их не удалят. Система кэширования позволяет очищать кэш ассоциированный с определенной страницей или вы можете удалить целую коллекцию кэш-файлов. Обычно используются служебные функции, описанные ниже, для удаления кэш-файлов после наступления определенных событий, таких как, например, добавление информации в базу данных.

## Повысит ли кэширование производительность вашего сайта?

Повышение производительности, достигающееся за счет кэширования, зависит от многих факторов. Если имеется высокооптимизированная база данных с очень низкой загрузкой, то возможно повышение производительности останется незаметным. Если же имеется база данных с высокой нагрузкой, то возможно вы заметите повышение скорости реакции приложения, при условии, что имеется достаточный запас места в файловой системе. Запомните, что кэширование просто изменяет местоположение информации, перемещяя ее из базы данных в файловую систему.

Например, применение кэширования на сервере с ресурсами кластерного типа, может оказаться неэффективным из-за большого количества операций происходящих в его файловой системе. На одиночных же серверах с общими ресурсами, использование кэширования вероятно будет оправданно. К сожалению, нет четкого ответа на вопрос о том, когда именно следует применять кэширования. На самом деле это зависит от конкретной ситуации.

## Как хранятся кэш-файлы?

CodeIgniter размещает результаты КАЖДОГО запроса в его собственном кэш-файле. Затем наборы кэш-файлов упорядочиваются в поддиректориях соответствующих функциям контроллера. Точнее, поддиректории именуются в соответвии с первыми двумя сегментами URI (имя класс контроллера и имя функции).

Например, допустим имеется контроллер **blog** с функцией **comments**, которая содержит три запроса, тогда система кэширования создаст кэш-директорию с именем **blog+comments**, в которую она запишет три кэш-файла.

Если используются динамические запросы, которые изменяются в зависимости от информации в URI (например, при использовании пагинации — разбиения на страницы), то в этом случае каждый экземпляр запроса сгенерирует свой собственный кэш-файл. Следовательно, в результате может получится, что кэш-файлов будет больше, чем запросов.

## Управление кэш-файлами

Поскольку время жизни кэш-файлов неограниченно, то в приложении необходимо создавать подсистему их удаления. Например, допустим у вас есть блог, который позволяет пользователям оставлять комментарии. Всякий раз после появления нового комментария, необходимо удалять кэш-файлы, ассоциированные с функцией контроллера, обслуживающей комментарии. Ниже вы найдете две функции, которые помогают очистить данные.

## Не все функции базы данных работают с кэшированием

В заключении, обратим внимание на то, что кэшированный результирующий объект — это упрощенная версия полного результирующего объекта. По этой причине некоторые функции результатов запроса недоступны для использования.

Следующие функции НЕДОСТУПНЫ при использовании кэшированного результирующего объекта:

* num\_fields()
* field\_names()
* field\_data()
* free\_result()

Также при кэшировании недоступны два ресурса базы данных (result\_id и conn\_id), поскольку результирующие ресурсы относятся только к операциям, выполняемым в данный момент.

# Справочник функций

## $this->db->cache\_on()  /   $this->db->cache\_off()

Ручное включение/отключение кэширования. Эта функция может оказаться полезной, если необходимо оградить какой-то конкретный запрос от кэширования. Пример:

// Включение кэширования  
$this->db->cache\_on();  
$query = $this->db->query("SELECT \* FROM mytable");  
// Отключение кэширования для этого одного запроса  
$this->db->cache\_off();  
$query = $this->db->query("SELECT \* FROM members WHERE member\_id = '$current\_user'");  
// Обратное включение кэширования  
$this->db->cache\_on();  
$query = $this->db->query("SELECT \* FROM another\_table");

## $this->db->cache\_delete()

Удаляет кэш-файлы ассоциированные с определенной страницей. Эта функция полезна, когда необходимо очистить кэш после обновления базы данных.

Система кэширования сохраняет кэш-файлы в директориях соответвующих URI просмотренных страниц. Например, если вы просматриваете страницу**www.your-site.com/index.php/blog/comments**, то система кэширования поместит кэш-файлы, ассоциированные с этой страницей, в директорию**blog+comments**. Для удаления этих кэш-файлов, необходимо использовать:

$this->db->cache\_delete('blog', 'comments');

Если не указывать параметры, то при определнии того, что необходимо очистить, будет использоваться текущий URI.

## $this->db->cache\_delete\_all()

Очищает все существующие кэш-файлы. Пример:

$this->db->cache\_delete\_all();

# Класс Database Forge

Класс Database Forge содержит функции, помогающие манипулировать базой данных и структурой таблиц.

### Содержание

* [Инициализация класса Forge](http://code-igniter.ru/user_guide/database/forge.html#init)
* [Создание БД](http://code-igniter.ru/user_guide/database/forge.html#create)
* [Удаление БД](http://code-igniter.ru/user_guide/database/forge.html#drop)
* [Добавление полей](http://code-igniter.ru/user_guide/database/forge.html#add_field)
* [Добавление ключей](http://code-igniter.ru/user_guide/database/forge.html#add_key)
* [Создание таблиц](http://code-igniter.ru/user_guide/database/forge.html#create_table)
* [Удаление таблиц](http://code-igniter.ru/user_guide/database/forge.html#drop_table)
* [Изменение таблиц](http://code-igniter.ru/user_guide/database/forge.html#modifying_tables)

## Инициализация класса Forge

**Важно:**  Перед запуском данного класса должен быть запущен драйвер для работы с БД!

Загрузка класса производится этим кодом :

$this->load->dbforge()

После инициализации объект можно использовать как **$this->dbforge**:

$this->dbforge->some\_function()

## $this->dbforge->create\_database('db\_name')

Создает базу данных с именем, заданным в параметре. Возвращает TRUE/FALSE в зависимости от результата:

if ($this->dbforge->create\_database('my\_db'))  
{  
    echo 'База создана!';  
}

## $this->dbforge->drop\_database('db\_name')

Удаляет базу данных с именем, заданным в параметре. Возвращает TRUE/FALSE в зависимости от результата:

if ($this->dbforge->drop\_database('my\_db'))  
{  
    echo 'База удалена!';  
}

# Создание и удаление таблиц

При создании таблиц можно добавлять новые поля и индексы. CodeIgniter поддерживает механизм для этих действий.

## Добавление полей

Поля таблицы задаются как ассоциативный массив. Для каждого поля задается его ключ и значение. Например, ключ - 'type', возможные значения - INT, VARCHAR, TEXT, и т.д.

$fields = array(  
                        'users' => array(  
                                                 'type' => 'varchar',  
                                                 'constraint' => '100',  
                                          ),  
                );  
// будет создано поле "users VARCHAR(100)"

Так же могут использоваться следующие ключи/значения:

* unsigned/true : создание типа "UNSIGNED" в описании поля
* default/value : создание значения по умолчанию
* null/true : создание "NULL" (не обязательного поля). По умолчанию - "NOT NULL"
* auto\_increment/true : установка флага auto\_increment. Примечание: только для числовых типов.

$fields = array(  
                        'blog\_id' => array(  
                                                 'type' => 'INT',  
                                                 'constraint' => 5,   
                                                 'unsigned' => TRUE,  
                                                 'auto\_increment' => TRUE  
                                          ),  
                        'blog\_title' => array(  
                                                 'type' => 'VARCHAR',  
                                                 'constraint' => '100',  
                                          ),  
                        'blog\_author' => array(  
                                                 'type' =>'VARCHAR',  
                                                 'constraint' => '100',  
                                                 'default' => 'King of Town',  
                                          ),  
                        'blog\_description' => array(  
                                                 'type' => 'TEXT',  
                                                 'null' => TRUE,  
                                          ),  
                );  
);

После определения полей следует их добавить в таблицу функцией **$this->dbforge->add\_field($fields)** перед созданием таблицы ф-цией**create\_table()**.

### $this->dbforge->add\_field()

Добавление полей к таблице

### Использование строк как полей

Если Вы знаете, как задавать описание поля строкой, то можно использовать ее в качестве параметра функции add\_field()

$this->dbforge->add\_field("label varchar(100) NOT NULL DEFAULT 'default label'");

Важно: Можно вызывать **add\_field()** несколько раз. Тогда строки буду соединены воедино.

### Создание поля типа id

Это специальное решение для создания полей типа id. Тип - INT(9), флаг auto\_incremen, задано как Primary Key.

$this->dbforge->add\_field('id');  
// = id INT(9) NOT NULL AUTO\_INCREMENT

## Добавление ключей

Добавление ключей таблицы производится функцией **$this->dbforge->add\_key('field')**. Второй необязательный параметр, заданный как TRUE создает первичный ключ. Заметьте, что вызов **add\_key()** должен следовать перед созданием таблицы **create\_table()**.

$this->dbforge->add\_key('blog\_id', TRUE);  
// = PRIMARY KEY (blog\_id)  
$this->dbforge->add\_key('blog\_name');  
// = KEY (blog\_name)

## Создание таблицы

После того, как будут определены поля и ключи, нужно создать таблицу:

$this->dbforge->create\_table('table\_name');  
// = CREATE TABLE table\_name

Установленный необязательный второй параметр в TRUE добавляет условие "IF NOT EXISTS" при создании таблицы

$this->dbforge->create\_table('table\_name', TRUE);  
// = CREATE TABLE IF NOT EXISTS table\_name

## Удаление таблицы

Равносильно sql-коду DROP TABLE :

$this->dbforge->drop\_table('table\_name');  
// = DROP TABLE IF EXISTS table\_name

# Изменение таблиц

## $this->dbforge->add\_column()

Функция add\_column() используется для добавления новых полей в таблицу.

$fields = array(  
                        'preferences' => array('type' => 'TEXT')  
);  
$this->dbforge->add\_column('table\_name', $fields);  
// = ALTER TABLE sites ADD preferences TEXT

## $this->dbforge->drop\_column()

Удаляет поле из таблицы.

$this->dbforge->drop\_column('table\_name', 'column\_to\_drop');

## $this->dbforge->modify\_column()

Эта функция идентична add\_column(), только позволяет еще и редактировать существующие поля.

$fields = array(  
                        'old\_name' => array(  
                                                         'name' => 'new\_name',  
                                                         'type' => 'TEXT',  
                                                ),  
);  
$this->dbforge->modify\_column('sites', $fields);  
// = ALTER TABLE sites CHANGE old\_name new\_name TEXT

# Утилиты для работы с базами данных

Утилиты для работы с базами данных включают в себя функции, помогающие управлять БД(базой данных).

### Содержание

* [Инициализация класса](http://code-igniter.ru/user_guide/database/utilities.html#init)
* [Список Ваших БД](http://code-igniter.ru/user_guide/database/utilities.html#list)
* [Оптимизация таблиц](http://code-igniter.ru/user_guide/database/utilities.html#opttb)
* [Исправление баз данных](http://code-igniter.ru/user_guide/database/utilities.html#repair)
* [Оптимизация БД](http://code-igniter.ru/user_guide/database/utilities.html#optdb)
* [Конвертирование результата в CSV файл](http://code-igniter.ru/user_guide/database/utilities.html#csv)
* [Конвертирование результата в XML файл](http://code-igniter.ru/user_guide/database/utilities.html#xml)
* [Бэкап базы данных](http://code-igniter.ru/user_guide/database/utilities.html#backup)

## Инициализация класса

**Важно:** Перед запуском данного класса должен быть запущен драйвер для работы с БД!

Загрузка класса следующей функцией :

$this->load->dbutil()

После инициализации объект можно использовать как **$this->dbutil**:

$this->dbutil->some\_function()

## $this->dbutil->list\_databases()

Возвращает имена баз данных :

$dbs = $this->dbutil->list\_databases();  
foreach($dbs as $db)  
{  
    echo $db;  
}

## $this->dbutil->optimize\_table('table\_name');

**Важно:**  Эта функция доступна только на MySQL/MySQLi базах.

Позволяет оптимизировать таблицу, имя которой указано в параметре. Возвращает TRUE/FALSE при успешном/неуспешном выполнении:

if ($this->dbutil->optimize\_table('table\_name'))  
{  
    echo 'Success!';  
}

## $this->dbutil->repair\_table('table\_name');

**Важно:**  Эта функция доступна только на MySQL/MySQLi базах.

Исправляет таблицу, имя которой указано в параметре. Возвращает TRUE/FALSE при успешном/неуспешном выполнении:

if ($this->dbutil->repair\_table('table\_name'))  
    echo 'Success!';

## $this->dbutil->optimize\_database();

**Важно:**  Эта функция доступна только на MySQL/MySQLi базах.

Позволяет оптимизировать базу данных, к которой сейчас подключен класс . Возвращает массив статуса БД или FALSE при ошибке.

$result = $this->dbutil->optimize\_database();  
if ($result !== FALSE)  
    print\_r($result);

## $this->dbutil->csv\_from\_result($db\_result)

Позволяет создавать CSV файл из результата запроса. Первый параметр содержит текст запроса:

$this->load->dbutil();  
$query = $this->db->query("SELECT \* FROM mytable");  
echo $this->dbutil->csv\_from\_result($query);

Второй и третий параметры позволяют установить разделитель и символ перехода на новую строку. По умолчанию разделитель - "\t"(знак табуляции), переход на новую строку - "\n". Например:

$delimiter = ";";  
$newline = "\r\n";  
echo $this->dbutil->csv\_from\_result($query, $delimiter, $newline);

**Важно:**  Эта функция НЕ ЗАПИСЫВАЕТ данные в файл. Она просто создает структуру CSV . Если необходимо сохранить в файл, воспользуйтесь[хелпером File](http://code-igniter.ru/user_guide/helpers/file_helper.html).

## $this->dbutil->xml\_from\_result($db\_result)

Позволяет создавать XML файл из результата запроса. Параметры: запрос и конфигурационный массив. Например:

$this->load->dbutil();  
$query = $this->db->query("SELECT \* FROM mytable");  
$config = array (  
                  'root'    => 'root',  
                  'element' => 'element',   
                  'newline' => "\n",   
                  'tab'    => "\t"  
                );  
echo $this->dbutil->xml\_from\_result($query, $config);

**Важно:**  Эта функция НЕ ЗАПИСЫВАЕТ данные в файл. Она просто создает структуру XML . Если необходимо сохранить в файл, воспользуйтесь [хелпером File](http://code-igniter.ru/user_guide/helpers/file_helper.html).

## $this->dbutil->backup()

Создает бэкап (резервную копию) всех бд или определенных таблиц. Бэкап может быть сжат в Zip или Gzip формат.

**Важно:**  Эта функция доступна только для MySQL баз.

Примечание: В связи с ограничениями на размер памяти и файла, бэкап большой бд не может быть выполнен. Если база большая, можно подключится к SQL серверу и выполнить бэкап из командной строки, или попросить об этом администратора сервера, если вы не имеете необходимых прав.

### Использование на примере

**// Загружаем класс DB utility**  
$this->load->dbutil();  
**// Создаем бэкап текущей бд и ассоциируем его с переменной**  
$backup =& $this->dbutil->backup();   
**// Загружаем хелпер file и записываем бэкап в файл**  
$this->load->helper('file');  
write\_file('/path/to/mybackup.gz', $backup);   
**// Загружаем хелпер download и отправляем бэкап пользователю**  
$this->load->helper('download');  
force\_download('mybackup.gz', $backup);

### Установка настроек бэкапа

Настройки бэкапа задаются массивом - первым параметром функции :

$prefs = array(  
      'tables'      => array('table1', 'table2'),  // Массив таблиц для бэкапа.  
      'ignore'      => array(),           // Массив таблиц, не нужных в бэкапе   
      'format'      => 'txt',             // gzip, zip, txt  
      'filename'    => 'mybackup.sql',    // Имя файла - ТОЛЬКО ДЛЯ ZIP ФАЙЛОВ   
      'add\_drop'    => TRUE,      // позволяем дописать параметр DROP TABLE в бэкап   
      'add\_insert'  => TRUE,              // позволяем дописать параметр INSERT в бэкап   
      'newline'     => "\n"               // Символ новой строки   
      );  
$this->dbutil->backup($prefs);

### Описание настроек бэкапа

|  |  |  |  |
| --- | --- | --- | --- |
| **Индекс** | **Значение по умолчанию** | **Опции** | **Описание** |
| **tables** | пустой массив | Нет | Массив таблиц для бэкапа. Если пусто - все таблицы |
| **ignore** | пустой массив | Нет | Массив таблиц, исключенных для бэкапа |
| **format** | gzip | gzip, zip, txt | Формат экспортируемого файла |
| **filename** | текущее дата/время | Нет | Имя файла. Нужно только при использовании ZIP-сжатия |
| **add\_drop** | TRUE | TRUE/FALSE | Включение параметра DROP TABLE в экспортируемый SQL файл. |
| **add\_insert** | TRUE | TRUE/FALSE | Включение параметра INSERT в экспортируемый SQL файл. |
| **newline** | "\n" | "\n", "\r", "\r\n" | Тип символа новой строки в SQL файле. |

# Класс Email

Класс Email поддерживает следующие возможности:

* Работа с Mail, Sendmail и SMTP
* Массовая рассылка
* CC и BCCs
* HTML или Обычный текст писем
* Вложения
* Перенос строк
* Приоритеты
* BCC Batch Mode, позволяет разбивать большие списки адресов на мелкие пакеты.
* Инструменты отладки

## Отправка сообщений

Отправка сообщений проста. Вы можете настроить её «на лету» или задать настройки в файле конфигурации.

Вот основной пример, демонстрирующий как можно отправить email. Замечание: Предполагается, что вы отправляете письмо из одного из ваших[контроллеров](http://code-igniter.ru/user_guide/general/controllers.html).

$this->load->library('email');  
$this->email->from('your@your-site.com', 'Your Name');  
$this->email->to('someone@example.com');   
$this->email->cc('another@another-example.com');   
$this->email->bcc('them@their-example.com');   
$this->email->subject('Тест Email');  
$this->email->message('Тестирование класса отправки сообщений');   
$this->email->send();  
echo $this->email->print\_debugger();

## Настройка параметров письма

Существует 17 различных параметров. Вы можете устанавливать их автоматически, подгружая из файла конфигурации или вручную, как это описано ниже:

Параметры помещаются в массив и затем инициализируются с помощью функции **initialize**. Вот пример того, как это можно сделать:

$config['protocol'] = 'sendmail';  
$config['mailpath'] = '/usr/sbin/sendmail';  
$config['charset'] = 'iso-8859-1';  
$config['wordwrap'] = TRUE;  
$this->email->initialize($config);

**Замечание:** Большинство параметров имеют значения по-умолчанию, которые будут использоваться в том случае, если вы не задали свои значения.

### Настройка параметров в файле конфигурации

Если вы не хотите задавать параметры вышеуказанным методом, то можете поместить их в файл конфигурации. Просто создайте новый файл и назовите его **email.php**, добавьте массив **$config** в этот файл. Затем сохраните его в**config/email.php** и он будет использоваться автоматически. Вам НЕ придется использовать функцию **$this->email->initialize()**, если Вы будете хранить параметры в файле конфигурации.

## Параметры письма

Ниже приведен список всех параметров, которые могут быть установлены при отправке email.

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Значение по-умолчанию** | **Опции** | **Описание** |
| **useragent** | CodeIgniter | Нет | Почтовый клиент. |
| **protocol** | mail | mail, sendmail, или smtp | Протокол. |
| **mailpath** | /usr/sbin/sendmail | Нет | Серверный путь к Sendmail. |
| **smtp\_host** | Не определено | Нет | Адрес SMTP-сервера. |
| **smtp\_user** | Не определено | Нет | SMTP логин. |
| **smtp\_pass** | Не определено | Нет | SMTP пароль. |
| **smtp\_port** | 25 | Нет | SMTP порт. |
| **smtp\_timeout** | 5 | Нет | SMTP тайм-аут (в секундах). |
| **wordwrap** | TRUE | TRUE или FALSE (boolean) | Включение переносов. |
| **wrapchars** | 76 |  | Число символов до переноса. |
| **mailtype** | text | text или html | Тип письма. Если отсылаете письмо в виде HTML, то Вы должны отправить его как полноценную веб-страницу. Убедитесь, что отсутствуют относительные ссылки и относительные пути изображений, иначе они не будут работать. |
| **charset** | utf-8 |  | Установка кодировки письма (utf-8, iso-8859-1 и т.д.). |
| **validate** | FALSE | TRUE или FALSE (boolean) | Валидация email-адреса. |
| **priority** | 3 | 1, 2, 3, 4, 5 | Email-приоритеты. 1 = самый высокий. 5 = самый низкий. 3 = нормальный. |
| **newline** | \n | "\r\n" или "\n" | Тип переноса на новую строку. (Используйте "\r\n" для соблюдения стандарта RFC 822). |
| **bcc\_batch\_mode** | FALSE | TRUE или FALSE (boolean) | Включение пакетного режима BCC. |
| **bcc\_batch\_size** | 200 | Нет | Количество адресов в одном BCC-пакете. |

## Функции класса Email

### $this->email->from()

Задает адрес электронной почты и имя лица-отправителя:

$this->email->from('**you@your-site.com**', '**Ваше Имя**');

### $this->email->reply\_to()

Устанавливает адрес для ответа, если не была указана информация в функции "from". Пример:

$this->email->reply\_to('**you@your-site.com**', '**Ваше Имя**');

### $this->email->to()

Задает адрес(а) получателя (получателей). Может быть указан один адрес электронной почты, список через запятую или массив:

$this->email->to('**someone@example.com**');$this->email->to('**one@example.com**, **two@example.com**, **three@example.com**');$list = array('**one@example.com**', '**two@example.com**', '**three@example.com**');  
$this->email->to(**$list**);

### $this->email->cc()

Задает CC адрес(а). Точно так же, как и в «to», может быть указан один адрес электронной почты, список через запятую или массив.

### $this->email->bcc()

Задает BCC адрес(а). Точно так же, как и в «to», может быть указан один адрес электронной почты, список через запятую или массив:

### $this->email->subject()

Устанавливает тему письма:

$this->email->subject('**Это тема письма**');

### $this->email->message()

Устанавливает текст сообщения:

$this->email->message('**Вот мое сообщение**');

### $this->email->set\_alt\_message()

Задает текст альтернативного сообщения:

$this->email->set\_alt\_message('**Альтернативное сообщение**');

Это — дополнительная строка сообщения, которая может быть использована, если вы посылаете письмо HTML-формата. Это позволяет вам определить альтернативное сообщение без HTML-формата, которое будет добавлено к строке заголовка для людей, которые не принимают письма в HTML-формате. Если Вы не установите альтернативное сообщение, CodeIgniter сам извлечет сообщение из HTML письма и удалит все теги.

### $this->email->clear()

Очищает все email-переменные. Эта функция обычно используется в том случае, когда вы запускаете функцию отправки писем в цикле, и возникает необходимость сбросить значения между циклами.

foreach ($list as $name => $address)  
{  
    $this->email->clear();  
    $this->email->to($address);  
    $this->email->from('your@your-site.com');  
    $this->email->subject('Here is your info '.$name);  
    $this->email->message('Hi '.$name.' Here is the info you requested.');  
    $this->email->send();  
}

Если вы установите параметр TRUE, то также очистятся и все вложения:

$this->email->clear(TRUE);

### $this->email->send()

Функция отправки письма. Возвращает TRUE или FALSE в зависимости от успешного или неуспешного выполнения функции. Пример условного использования:

if ( ! $this->email->send())  
{  
    // Генерация ошибки  
}

### $this->email->attach()

Позволяет отсылать письма с вложением. В качестве первого параметра выступает путь к файлу и его имя. Замечание: Используйте реальный путь к файлу, а не URL. Если необходимо сделать несколько вложений, то функция пишется количество раз равное количеству вложений. Например:

$this->email->attach('/path/to/photo1.jpg');  
$this->email->attach('/path/to/photo2.jpg');  
$this->email->attach('/path/to/photo3.jpg');  
$this->email->send();

### $this->email->print\_debugger()

Возвращает строку, содержащую любые сообщения сервера, заголовки и содержания писем. Необходима при отладке.

## Перенос слов

Если включен перенос (соблюдайте стандарт RFC 822) и в письме есть очень длинные ссылки, то они могут быть перенесены и станут нерабочими. CodeIgniter позволяет решить эту проблему:

Текст письма,  
который будет перенесен как обычно.  
**{unwrap}**http://www.example.com/a\_long\_link\_that\_should\_not\_be\_wrapped.html**{/unwrap}**  
Еще какой-то текст,  
который будет перенесен как обычно.

Помещайте текст, который не надо переносить, между: **{unwrap}{/unwrap}**

# Класс Encryption

Класс Encryption (класс шифрования) представляет два способа шифрования. Это шифрование на основе XOR и случайного хеша, и использование библиотеки Mcrypt. Если библиотека Mcrypt не доступна на Вашем сервере, зашифрованные данные все еще будут обеспечивать определенную защиту и работать по более "лёгким" алгоритмам шифрования. Если Mcrypt доступна, то Ваши данные буду иметь высокую криптостойкость, а Вашем распоряжении будем множество алгоритмов, такие как AES, BlowFish или Tea.

## Установка ключа

*Ключ* - это часть контроля криптографической защиты и возможность расшифровать кодированные данные. К сведению, Вы можете расшифровать строку **только** с тем же ключом, с каким она была зашифрована.

Таким образом, ключ следует хранить в безопасном месте. Как только становится известен ключ, то расшифровка данных не составит большого труда.

Для максимальной надежности ключ должен быть 128-битным (32 символа), содержать строчные и прописные литеры, символы и цифры. Он не должен быть простым текстом, так как его легко подобрать или выведать.

Ваш ключ можно задать в файле **application/config/config.php**, или Вы должны создать собственный механизм, подставляющий ключ при каждой шифровке/дешифровке.

Для сохранения ключа в файле **application/config/config.php** откройте его и задайте параметр encryption\_key:

$config['encryption\_key'] = "ВАШ КЛЮЧ";

## Размер данных

Важно знать, что длина зашифрованных данных в среднем в 2.6 раза больше размера исходных данных. Например, если зашифровать строку "Супер-пупер секретные данные" длиной в 29 символов, получится шифрованная строка длиной 77 символов ( изменение длины строк неравномерно, так как при шифровании используются кластеры в 64 бита). Учитывайте данные сведения при проектировании защитной структуры. Cookies, например, могут содержать только 4 килобайта информации.

## Инициализация класса

Как и многие другие классы в CodeIgniter, класс Encryption инициализируется функцией **$this->load->library**:

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

После загрузки, объект будет доступен по вызову **$this->encrypt**

## $this->encrypt->encode()

Выполняет шифрование данных и представляет результат в виде строки. Например:

$msg = 'My secret message';  
$encrypted\_string = $this->encrypt->encode($msg);

Вы также можете задавать ключ для шифрования вторым параметром функции, если не хотите использовать заданный в config.php

$msg = 'My secret message';  
$key = 'super-secret-key';  
$encrypted\_string = $this->encrypt->encode($msg, $key);

## $this->encrypt->decode()

Расшифровка кодированных данных. Пример:

$encrypted\_string = 'APANtByIGI1BpVXZTJgcsAG8GZl8pdwwa84';  
$plaintext\_string = $this->encrypt->decode($encrypted\_string);

## $this->encrypt->set\_cipher();

Устанавливает алгоритм шифрования. По умолчанию **MCRYPT\_RIJNDAEL\_256**. Например:

$this->encrypt->set\_cipher('MCRYPT\_BLOWFISH');

На сайте php.net Вы можете увидеть список [доступных алгоритмов](http://php.net/mcrypt).

Если вы хотите протестировать сервера на поддержку библиотеки Mcrypt, используйте следующий код:

echo ( ! function\_exists('mcrypt\_encrypt')) ? 'Не поддерживается' : 'Поддерживается';

## $this->encrypt->set\_mode();

Позволяет устанавливать метод для шифрования. По умолчанию используется метод **MCRYPT\_MODE\_ECB**. Пример:

$this->encrypt->set\_mode('MCRYPT\_MODE\_CFB');

Для списка всех режимов посетите сайт [php.net/mcrypt](http://php.net/mcrypt).

## $this->encrypt->sha1();

SHA1 - функция кодирования. Возвращает 160-битный хеш данных.**Примечание**: SHA1, как и MD5, является не-декодируемой функцией. Пример:

$hash = $this->encrypt->sha1('Some string');

Почти всегда на серверах с PHP есть поддержка хеширования по алгоритму SHA 1. В таком случае лучше использовать стандартную функцию:

$hash = sha1('Some string');

Если Ваш сервер не поддерживает эту функцию, то следует использовать функцию SHA 1 из данного класса.

# Класс File Uploading

Этот класс позволяет заливать файлы на сервер. При этом Вы можете задавать различные критерии заливки, например, тип файлов или их максимальный размер.

## Как это происходит

Загрузка файла включает в себя следующие этапы:

* Отрисовывается форма загрузки, позволяющая пользователю выбрать файл и загрузить его.
* После сабмита формы файл закачивается в выбранную пользователем папку.
* Далее файл проверяется по заданным пользователем критериям.
* Если загрузка прошла успешно, пользователю показывается сообщение об этом.

Чтобы продемонстрировать этот процесс в действии, давайте разберем конкретный пример. Ну а в конце Вы найдете описание класса.

## Создаем форму загрузки

В текстовом редакторе создайте файл и назовите его **upload\_form.php**. Вставьте туда код, представленный ниже, и сохраните в папке **application/views/**:

<html>

<head>

<title>Форма загрузки</title>

</head>

<body>

<?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>

Как видно из примера, мы использовали хелпер форм для создания открывающего тэга формы. Для загрузки файлов необходима форма с атрибутом multipart, а хелпер создаст нам правильный синтаксис. Также мы использовали переменную $error. Она нужна для того, чтобы показать сообщения об ошибках, если пользователь сделает что-то не так.

## Страница уведомления о завершенной загрузке

В текстовом редакторе создайте файл и назовите его **upload\_success.php**. Вставьте туда код, представленный ниже и сохраните в папке**application/views/**:

<html>

<head>

<title>Форма загрузки</title>

</head>

<body>

<h3>Файл успешно загружен!</h3>

<ul>

<?php foreach($upload\_data as $item => $value):?>

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

<?php endforeach; ?>

</ul>

<p><?php echo anchor('upload', 'Загрузить еще!'); ?></p>

</body>

</html>

## Контроллер

В текстовом редакторе создайте файл и назовите его **upload.php**. Это будет контроллер. Вставьте в файл код, представленный ниже и сохраните в папке**application/controllers/**:

<?php

class Upload extends Controller {

function Upload()

{

parent::Controller();

$this->load->helper(array('form', 'url'));

}

function index()

{

$this->load->view('upload\_form', array('error' => ' ' ));

}

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())

{

$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);

}

}

}

?>

## Папка загрузки

Ещё нам понадобится папка, куда будут сохраняться загруженные файлы. Создайте папку там, где у вас установлен CodeIgniter, назовите её **uploads** и выставьте права доступа 777.

## Пробуем!

Чтобы проверить форму в действии, зайдите на сайт, используя URL, похожий на этот:

example.com/index.php/**upload**/

По идее, Вы должны увидеть форму загрузки. Попробуйте загрузить картинку (jpg, gif или png). Если путь в контроллере прописан верно, то все должно работать.

# Описание класса

## Инициализация класса Upload

Так же, как и большинство классов в CodeIgniter, класс Upload инициализируется в контроллере функцией **$this->load->library**:

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

После загрузки класса к его объектам можно будет обращаться, используя конструкцию: **$this->upload**

## Установка параметров

Так же, как и в других библиотеках, вы можете задавать критерии загрузки, используя различные параметры. В контроллере, созданном выше, вы использовали следующие из них:

$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);  
// Вы можете также установить параметры, вызвав функцию инициализации. Используйте её, если класс загружается автоматически:  
$this->upload->initialize($config);

Собственно, параметры, описанные выше не нуждаются в разъяснениях. Ниже приведена таблица всех возможных параметров.

## Параметры

Доступны следующие параметры. Значение по умолчанию будет применяться, если параметр не задан явно.

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Опции** | **Описание** |
| **upload\_path** | Нет | Нет | Путь до папки, куда будет загружен файл. У папки должны быть выставлены права на запись, а путь может быть как абсолютным, так и относительным. |
| **allowed\_types** | Нет | Нет | Типы MIME, описывающие типы файлов, разрешенных для загрузки. Обычно в качестве MIME-типа используется расширение файла. Несколько типов разделяются вертикальной чертой. |
| **overwrite** | FALSE | TRUE/FALSE (boolean) | Если TRUE, и в папке уже есть файл с тем же именем, что и заливаемый, то он будет перезаписан. Если false, то перезаписи не будет, а к имени заливаемого файла добавится порядковый номер. |
| **max\_size** | 0 | Нет | Максимальный размер файла (в килобайтах). Если ограничения нет, то пишем 0. На заметку: на многих серверах с установленным PHP имеется ограничение на размер заливаемых файлов, записанное в файле php.ini. Как правило, по умолчанию устанавливается 2 МБ (2048 КБ). |
| **max\_width** | 0 | Нет | Максимальная ширина картинки в пикселях. 0 — не ограниченно. |
| **max\_height** | 0 | Нет | Максимальная высота картинки в пикселях. 0 — не ограниченно. |
| **encrypt\_name** | FALSE | TRUE/FALSE (boolean) | Если TRUE, то имя файла преобразуется в случайным образом сгенерированную строку. Может быть полезно, если хотите, чтобы залитый файл не мог быть распознан заливающим. |
| **remove\_spaces** | TRUE | TRUE/FALSE (boolean) | Если TRUE, то все пробелы в имени файла будут преобразованы в знак подчеркивания. Рекомендуется всегда использовать данную опцию. |

## Устанавливаем параметры через файл конфигурации

Если вам не нравится метод задания параметров, описанный выше, можете задавать их, используя файл конфигурации. Просто создайте новый файл с именем **upload.php** и добавьте туда массив **$config**. Затем сохраните файл в папку **config/upload.php**, и он подключится автоматически. При этом вам НЕ НУЖНО использовать конструкцию **$this->upload->initialize**.

## Описание функций

Доступны следующие функции

## $this->upload->do\_upload()

Начинает загрузку файла с параметрами, заданными вами ранее. На заметку: по умолчанию upload ожидает файл из поля с именем **userfile**, а сама форма должна быть типа multipart

<form method="post" action="some\_action" enctype="multipart/form-data" />

Если же поле имеет другое имя, то просто передайте его в качестве параметра функции **do\_upload**:

$field\_name = "some\_field\_name";  
$this->upload->do\_upload($field\_name)

## $this->upload->display\_errors()

Возвращает сообщения об ошибке, если **do\_upload()** вернула FALSE. Сообщения не выдаются на экран автоматически. Функция возвращает данные, которые затем в любой момент можно вывести на экран

### Форматированный вывод ошибок

По умолчанию функция, описанная выше, выводит ошибки, не заключая их в теги <p>. Вы можете задать свои собственные разделители. Например:

$this->upload->display\_errors('**<p>**', '**</p>**');

## $this->upload->data()

Это функция-хелпер, возвращающая массив, содержащий все данные о загруженном файле. Ниже показан прототип массива:

Array  
(  
    [file\_name]    => mypic.jpg  
    [file\_type]    => image/jpeg  
    [file\_path]    => /path/to/your/upload/  
    [full\_path]    => /path/to/your/upload/mypic.jpg  
    [raw\_name]     => mypic  
    [orig\_name]    => mypic.jpg  
    [file\_ext]     => .jpg  
    [file\_size]    => 22.2  
    [is\_image]     => 1  
    [image\_width]  => 800  
    [image\_height] => 600  
    [image\_type]   => jpeg  
    [image\_size\_str] => width="800" height="200"  
)

### Пояснение

Далее приводится описание полей вышеописанного массива:

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| **file\_name** | Имя загруженного файла, включая расширение. |
| **file\_type** | MIME-тип файла |
| **file\_path** | Абсолютный путь к файлу на сервере. |
| **full\_path** | Абсолютный путь до файла на сервере, включая имя файла |
| **raw\_name** | Имя файла без расширения |
| **orig\_name** | Первоначальное имя файла. Используется только при включенной опции encrypted\_name. |
| **file\_ext** | Расширение файла с точкой |
| **file\_size** | Размер файла в килобайтах |
| **is\_image** | Проверка на предмет является ли файл изображением. 1 = изображение. 0 = нет. |
| **image\_width** | Ширина изображения. |
| **image\_heigth** | Высота изображения |
| **image\_type** | Тип изображения. Как правило - расширение файла без точки. |
| **image\_size\_str** | Строка, включающая в себя параметры width и height. Полезно использовать внутри тэга img. |

# Класс FTP

Класс FTP позволяет с использованием протокола передачи файлов FTP загружать файлы на удаленный сервер, переименовывать их, перемещать, удалять и устанавливать права на доступ. Также класс содержит функцию создания "зеркал" - точных копий папок с локального сервера.

**Важно:**  SFTP и SSL FTP не поддерживаются, только стандартный FTP.

## Инициализация Класса

Как и многие другие классы в CodeIgniter, класс FTP инициализируется функцией**$this->load->library**:

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

После загрузки, объект будет доступен по вызову: **$this->ftp**

## Примеры Использования

В этом примере открывается соединение с FTP сервером, загружается локальный файл в режиме ASCII . Права на файл устанавливаются 755. Примечание: установка прав на файл доступна только в PHP начиная от версии 5.

$this->load->library('ftp');  
$config['hostname'] = 'ftp.site.com';  
$config['username'] = 'логин';  
$config['password'] = 'пароль';  
$config['debug'] = TRUE;  
$this->ftp->connect($config);  
$this->ftp->upload('/local/path/to/myfile.html', '/public\_html/myfile.html', 'ascii', 0775);  
$this->ftp->close();

В этом примере класс возвращает листинг файлов на сервере .

$this->load->library('ftp');  
$config['hostname'] = 'ftp.site.com';  
$config['username'] = 'логин';  
$config['password'] = 'пароль';  
$config['debug'] = TRUE;  
$this->ftp->connect($config);  
$list = $this->ftp->list\_files('/public\_html/');  
print\_r($list);  
$this->ftp->close();

Этот код создает копию локальной директории на удаленном сервере.

$this->load->library('ftp');  
$config['hostname'] = 'ftp.site.com';  
$config['username'] = 'логин';  
$config['password'] = 'пароль';  
$config['debug'] = TRUE;  
$this->ftp->connect($config);  
$this->ftp->mirror('/path/to/myfolder/', '/public\_html/myfolder/');  
$this->ftp->close();

# Описание Функций

## $this->ftp->connect()

Подсоединение и залогинивание к FTP серверу. Настройки подключения можно задать вручную в массиве или заранее прописать в конфигурационном файле.

Пример установки собственных параметров :

$this->load->library('ftp');  
$config['hostname'] = 'ftp.your-site.com';  
$config['username'] = 'your-username';  
$config['password'] = 'your-password';  
$config['port']     = 21;  
$config['passive']  = FALSE;  
$config['debug']    = TRUE;  
$this->ftp->connect($config);

### Установка параметров FTP в конфигурационном файле

Если Вы желаете сохранить настройки в конф. файле, то следует создать файл**ftp.php**, добавить в нем массив **$config** . Затем сохраните файл по адресу**config/ftp.php** и настройки из него будут использоваться по умолчанию каждый раз при подключении к FTP серверу.

### Доступные опции :

* **hostname** - имя FTP сервера для подключения .Например:  **ftp.example.com**
* **username** - FTP логин.
* **password** - FTP пароль.
* **port** - номер порта. Обычно **21**.
* **debug** - **TRUE/FALSE** (boolean). Включение/отключение режима отладки и отображения ошибок.
* **passive** - **TRUE/FALSE** (boolean). Установка пассивного режима. По умолчанию он включен.

## $this->ftp->upload()

Загрузка файлов на сервер. Параметры: локальный файл, имя файла на удаленном сервере после закачки. Необязательные параметры: режим и права на файл.

$this->ftp->upload('/local/path/to/myfile.html', '/public\_html/myfile.html', 'ascii', 0775);

**Доступные режимы закачки:**  **ascii**, **binary**, и **auto** (по умолчанию). Если используется режим **auto**, то режим определяется по расширению файла.

Установка прав на файл доступна, если используется PHP 5. Права должны быть восьмеричным (тип - **octal**) значением.

## $this->ftp->rename()

Позволяет переименовывать файлы и папки на сервере. Параметры: исходные файл/папка и новое название файла/папки.

// Переименовываем файл green.html в blue.html  
$this->ftp->rename('/public\_html/foo/green.html', '/public\_html/foo/blue.html');

## $this->ftp->move()

Перемещение файла. Параметры: исходный и конечный пути файла :

// Перемещаем blog.html из папки "joe" в "fred"  
$this->ftp->move('/public\_html/joe/blog.html', '/public\_html/fred/blog.html');

Примечание: если конечное имя файла изменено, то файл будет также переименован

## $this->ftp->delete\_file()

Удаление файла. Параметры: путь к файлу .

$this->ftp->delete\_file('/public\_html/joe/blog.html');

## $this->ftp->delete\_dir()

Функция позволяет удалять папки и все, что они содержат. Параметр - путь к папке с конечным слешем (/).

**Важно**  Будьте ОЧНЬ аккуратны с этой функцией, так как она удаляет всё, что находится в папке, включая и подпапки. Перед использованием убедитесь, что путь к папке корректен. Для этого можно использовать функцию **list\_files()** для отображения содержимого папки.

$this->ftp->delete\_dir('/public\_html/path/to/folder/');

## $this->ftp->list\_files()

Возвращает содержимое папки в виде **массива**. Аргумент - имя папки.

$list = $this->ftp->list\_files('/public\_html/');  
print\_r($list);

## $this->ftp->mirror()

Создает точную копию (т.н."зеркало") локальной директории на удаленном сервере, вместе со всеми подпапками и файлами. Структура директории полностью сохраняется.. Функции следует указать путь к локальной папке и ее будущему расположению на удаленном сервере:

$this->ftp->mirror('/path/to/myfolder/', '/public\_html/myfolder/');

## $this->ftp->mkdir()

Позволяет создавать папки на сервере Права устанавливаются восьмеричным (**octal**) значением (Только в PHP 5).

// Создаем папку "bar"  
$this->ftp->mkdir('/public\_html/foo/bar/', 0777);

## $this->ftp->chmod()

Позволяет устанавливать права на файл или папку. В параметрах укажите путь в файлу/папке и желаемые права :

// Установка прав на папку "bar" 777  
$this->ftp->chmod('/public\_html/foo/bar/', 0777);

## $this->ftp->close();

Закрывает соединение с FTP сервером .

# Класс HTML Table

Класс Table предоставляет функции, которые позволяют вам автоматически генерировать HTML код таблиц из массивов или наборов результатов запросов к базе данных.

## Инициализация класса

Как и большинство других классов в CodeIgniter, класс Table инициализируется в вашем контроллере при помощи функции **$this->load->library**:

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

Однажды загруженный, объект библиотеки Table будет доступен с помощью:**$this->table**

## Примеры

Вот пример показывающий каким образом вы можете создать таблицу из многомерного массива. Следует отметить, что первый индекс в массиве будет заголовком таблицы (или вы можете задать собственный заголовок при помощи функции **set\_heading()**, описанной ниже в справочнике функций).

$this->load->library('table');  
$data = array(  
             array('Name', 'Color', 'Size'),  
             array('Fred', 'Blue', 'Small'),  
             array('Mary', 'Red', 'Large'),  
             array('John', 'Green', 'Medium')   
             );  
echo $this->table->generate($data);

А вот пример того, как создать таблицу из результата запроса к базе данных. Этот класс автоматически генирирует заголовки основываясь на названиях таблиц (или вы можете задать собственные заголовоки при помощи функции**set\_heading()**, описанной ниже в справочнике функций).

$this->load->library('table');  
$query = $this->db->query("SELECT \* FROM my\_table");  
echo $this->table->generate($query);

Вот пример показывающий как вы можете создать таблицу используя дискретные параметры:

$this->load->library('table');  
$this->table->set\_heading('Name', 'Color', 'Size');  
$this->table->add\_row('Fred', 'Blue', 'Small');  
$this->table->add\_row('Mary', 'Red', 'Large');  
$this->table->add\_row('John', 'Green', 'Medium');  
echo $this->table->generate();

А вот тот же пример, но только вместо отдельных параметров используются массивы:

$this->load->library('table');  
$this->table->set\_heading(array('Name', 'Color', 'Size'));  
$this->table->add\_row(array('Fred', 'Blue', 'Small'));  
$this->table->add\_row(array('Mary', 'Red', 'Large'));  
$this->table->add\_row(array('John', 'Green', 'Medium'));  
echo $this->table->generate();

## Изменение внешнего вида вашей таблицы

Класс Table позволяет задать шаблон, в соответвии с которым вы можете определить дизайн вашей таблицы. Вот прототип шаблона:

$tmpl = array (  
      'table\_open'          => '<table border="0" cellpadding="4" cellspacing="0">',  
      'heading\_row\_start'   => '<tr>',  
      'heading\_row\_end'     => '</tr>',  
      'heading\_cell\_start'  => '<th>',  
      'heading\_cell\_end'    => '</th>',  
      'row\_start'           => '<tr>',  
      'row\_end'             => '</tr>',  
      'cell\_start'          => '<td>',  
      'cell\_end'            => '</td>',  
      'row\_alt\_start'       => '<tr>',  
      'row\_alt\_end'         => '</tr>',  
      'cell\_alt\_start'      => '<td>',  
      'cell\_alt\_end'        => '</td>',  
      'table\_close'         => '</table>'  
);  
$this->table->set\_template($tmpl);

**Примечание:**  Вы наверное заметили, что в шаблоне представленно два набора «row» блоков. Они позволяют вам создавать чередующиеся цвета строк или дизайн элементов, которые чередуются в каждой итерацией по данным строки.

Вам **необязательно** задавать шаблон целиком. Если вам необходимо изменить только часть таблицы, то вы можете задать только те элементы, которые вам необходимы. В этом примере был изменен только открывающий тег таблицы:

$tmpl = array ( 'table\_open'  => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">' );  
$this->table->set\_template($tmpl);

# Справочник функций

## $this->table->generate()

Возвращает строку содержащую сгенерированный код каблицы. Принимает необязательный параметр, которым может быть или массив, или объект результата запроса к базе данных.

## $this->table->set\_caption()

Позволяет вам добавить подпись к таблице.

$this->table->set\_caption('Colors');

## $this->table->set\_heading()

Позволяет вам задать заголовок таблицы. Вы можете использовать или массив, или дискретные параметры:

$this->table->set\_heading('Name', 'Color', 'Size');$this->table->set\_heading(array('Name', 'Color', 'Size'));

## $this->table->add\_row()

Позволяет добавить строку к вашей таблице. Вы можете использовать или массив, или дискретные параметры:

$this->table->add\_row('Blue', 'Red', 'Green');$this->table->add\_row(array('Blue', 'Red', 'Green'));

## $this->table->make\_columns()

Эта функция принимает на входе одномерный массив и создает многомерный массив с глубиной равной числу предполагаемых колонок в талице. Это позволяет массивам с множеством элементов отображаться в таблице с фиксированным числом колонок. Рассмотрим пример:

$list = array('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten', 'eleven', 'twelve');  
$new\_list = $this->table->make\_columns($list, 3);  
$this->table->generate($new\_list);  
// Генерирует таблицу в соответвии с этим прототипом  
<table border="0" cellpadding="4" cellspacing="0">  
<tr>  
<td>one</td><td>two</td><td>three</td>  
</tr><tr>  
<td>four</td><td>five</td><td>six</td>  
</tr><tr>  
<td>seven</td><td>eight</td><td>nine</td>  
</tr><tr>  
<td>ten</td><td>eleven</td><td>twelve</td></tr>  
</table>

## $this->table->set\_template()

Позволяет вам задать шаблон таблицы. Вы можете задать шаблон целиком, либо частично.

$tmpl = array ( 'table\_open'  => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">' );  
$this->table->set\_template($tmpl);

## $this->table->set\_empty()

Позволяет вам задать значение по-умолчанию для пустых ячеек таблицы. Вы можете задать, например, неразрывный пробел:

$this->table->set\_empty("&nbsp;");

## $this->table->clear()

Позволяет очистить заголовок таблица и данные строк. Если вам необходимо отобразить несколько таблиц с различными данными, то эту функцию следует вызывать после генерации каждой таблицы для того, чтобы очистить последующую таблицу от информации предыдущей. Пример:

$this->load->library('table');  
$this->table->set\_heading('Name', 'Color', 'Size');  
$this->table->add\_row('Fred', 'Blue', 'Small');  
$this->table->add\_row('Mary', 'Red', 'Large');  
$this->table->add\_row('John', 'Green', 'Medium');  
echo $this->table->generate();  
**$this->table->clear();**  
$this->table->set\_heading('Name', 'Day', 'Delivery');  
$this->table->add\_row('Fred', 'Wednesday', 'Express');  
$this->table->add\_row('Mary', 'Monday', 'Air');  
$this->table->add\_row('John', 'Saturday', 'Overnight');  
echo $this->table->generate();

# Класс работы с изображениями

Класс работы с изображениями CodeIgniter'а позволяет выполнять следующие действия:

* Изменение размера
* Создание эскиза
* Обрезка
* Поворот изображения
* Наложение "водяных знаков"

Поддерживаются все осноные библиотеки: GD/GD2, NetPBM, и ImageMagick

**Примечание:** Наложение "водяных знаков" доступно только в библиотеке GD/GD2 . Также GD используется для вычисления параметром изображения. Обработка же может проводиться с помощью Вашей библиотеки.

## Инициализация класса

Как и большинство библиотек в CodeIgniter, этот класс загружается с помощью функции **$this->load->library** :

$this->load->library('image\_lib');

После загрузки вызов класса будет производиться так: **$this->image\_lib**

## Обработка изображений

Вне зависимости от типа обработки (обрезка, поворот и т.д.), общая структура будет одинаковой. Сначала надо задать параметры, а затем вызвать необходимую функцию. Рассмотрим на примере создания эскиза:

$config['image\_library'] = 'gd2'; // выбираем библиотеку  
$config['source\_image'] = '/путь/к/фалу/mypic.jpg';   
$config['create\_thumb'] = TRUE; // ставим флаг создания эскиза  
$config['maintain\_ratio'] = TRUE; // сохранять пропорции  
$config['width'] = 75; // и задаем размеры  
$config['height'] = 50;  
$this->load->library('image\_lib', $config); // загружаем библиотеку   
$this->image\_lib->resize(); // и вызываем функцию

Приведенный код вызывает функцию **image\_resize** для обработки mypic.jpg, расположеннй в папке **source\_image** , затем создает эскиз размером 75 X 50 пикселей с использованием библиотеки GD2 **image\_library**. Затем флаг**maintain\_ratio** указывает, что бы эскиз оказался как можно менее искаженным при новых размерах. Эскиз будет назван mypic\_thumb.jpg

**Примечание:** Не забудьте установить права на запись для папки с изображением, иначе изображение не будет сохранено!

## Функции обработки

Есть четыре функций для обработки изображений и одна дополнительная:

* $this->image\_lib->resize()
* $this->image\_lib->crop()
* $this->image\_lib->rotate()
* $this->image\_lib->watermark()
* $this->image\_lib->clear()

Эти функции возвращают TRUE при успешном выполнении и FALSE в противном случае. Если произошла ошибка, то ее текст можно получить так:

echo $this->image\_lib->display\_errors();

Хорошим стилем является сразу создание кода, предусматривающего вывод ошибок:

if ( ! $this->image\_lib->resize())  
{  
    echo $this->image\_lib->display\_errors();  
}

Примечание: Так же можно задать свой стиль вывода ошибок с помощью передачи функции в параметрах HTML-тегов:

$this->image\_lib->display\_errors('**<p>**', '**</p>**');

## Параметры

Параметры функций описывают необходимые действия, что бы добиться желаемого результата.

Учтите, что не все параметры подходят для каждой функции. Например, x/y координаты применимы только для функции обрезки. Подобным образом, для обрезки параметры ширины и высоты не производят никакого действия. Таблица совместимости показывает, какие функции используют тот или иной параметр.

Легенда:

* **R** - Изменение размера
* **C** - Обрезка
* **X** - Поворот
* **W** - Наложение "водяного знака"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Опции** | **Описание** | **Совметимость** |
| **image\_library** | GD2 | GD, GD2, ImageMagick, NetPBM | Указывает библиотеку для обработки. | R, C, X, W |
| **library\_path** | Нет | Нет | Указывает полные пути к библиотеке ImageMagick или NetPBM. Указывайте, если вы их используете. | R, C, X |
| **source\_image** | Нет | Нет | Путь к файлу и его имя. Должен быть прямой серверный путь, не URL. | R, C, S, W |
| **dynamic\_output** | FALSE | TRUE/FALSE (логичекое) | Указывает, сохранять ли полученное изображение или использовать для динамического вывода. Примечание: если выбрано динамическое использование, то изображение будет выведено с заголовками изображения - то есть только само изображение, без html и других изображений. | R, C, X, W |
| **quality** | 90% | 1 - 100% | Устанавливает качество изображения. Чем выше качество, тем больше размер файла. И наоборот. | R, C, X, W |
| **new\_image** | Нет | Нет | Указывает путь и имя изображения для сохранения. Используется при создании копии изображения. Путь долженен быть относительным или абсолютным серверным путем, не URL. | R, C, X, W |
| **width** | Нет | Нет | Устанавливает желаемую ширину изображения. | R, C |
| **height** | Нет | Нет | Устанавливает желаемую высоту изображения. | R, C |
| **create\_thumb** | FALSE | TRUE/FALSE (логическое) | Указывает функции, что надо создать эскиз. | R |
| **thumb\_marker** | \_thumb | Нет | Указывает имя для эскиза. Будет вставлено перед расширением файла, например mypic.jpg станет mypic\_thumb.jpg | R |
| **maintain\_ratio** | TRUE | TRUE/FALSE (логическое) | Указывает, сохранять пропорции или нет. | R, C |
| **master\_dim** | auto | auto, width, height | Устанавливает размеры для создаваемого эскиза и при изменении. Например, вы хотите создать изображение 100 X 75 пикселей. Ели исходное изображение не позволяет изменить размер без сжатия, то данная настройка позволит изменить "жестко" размер. "auto" само определяет, шире или выше данное изображение нужного. | R |
| **rotation\_angle** | None | 90, 180, 270, vrt, hor | Указывает угол поворота изображения. Учтите, что в PHP градусы измеряются против часовой стрелки, и поворот на 90 градусов вправо будет задан как на 270. | X |
| **x\_axis** | Нет | Нет | Устанавливает X координату при обрезке. Например, при 30 обрежет изображение на 30 пикселей слева. | C |
| **y\_axis** | Нет | Нет | Устанавливает Y координату при обрезке. Например, при 30 обрежет изображение на 30 пикселей сверху. | C |

## Установка параметров в конфигурационном файле

Все параметры можно изначально задать в файле. Для этого создайте файл**image\_lib.php**, добавьте в него массив **$config**. Затем сохраните в папке**config/image\_lib.php**, и он будет использован автоматически. В таком случае отпадает необходимость в испльзовании функции **$this->image\_lib->initialize** .

## $this->image\_lib->resize()

Эта функция позволяет изменять размер файла, создавать его копию и эскиз.

Практически нет никакой разницы между созданием эскиза и копии с измененным размером (разве только в установке названия типа mypic\_thumb.jpg).

Эта функция поддерживает все параметры кроме трех: rotation\_angle, x\_axis и y\_axis.

### Создание эскиза

Функция создает эскиз в новом файле, если задан параметр create\_thumb :

$config['create\_thumb'] = TRUE;

### Создание копии

Функция создает копию в новом файле и/или папке, если задан параметр new\_image :

$config['new\_image'] = '/новый/путь/new\_image.jpg';

Примечания к данному параметру:

* Если задано только имя файла, то он будет сохранен в одной папке с оригинальным.
* Если задан только путь, то файл будет иметь такое же имя, как и оригинальный.

### Изменение размера

Если ни один из предыдущих параметров не задан, то действия будут производиться с оригинальным файлом.

## $this->image\_lib->crop()

Функция обрезки аналогична функции изменения размера, за исключением того, что надо задать смещение по X и Y (в пикселях), указывающее, с какого места обрезать. Например, будет оставлена часть 100 \* 40:

$config['x\_axis'] = '100';  
$config['y\_axis'] = '40';

Все настройка доступны, кроме этих: rotation\_angle, width, height, create\_thumb, new\_image.

Пример, как можно обрезать изображение:

$config['image\_library'] = 'imagemagick';  
$config['library\_path'] = '/usr/X11R6/bin/';  
$config['source\_image'] = '/path/to/image/mypic.jpg';  
$config['x\_axis'] = '100';  
$config['y\_axis'] = '60';  
$this->image\_lib->initialize($config);   
if ( ! $this->image\_lib->crop())  
{  
    echo $this->image\_lib->display\_errors();  
}

Замечание. Без визуального интерфейса очень сложно правильно обрезать изображение, и эта функция будет более полезной, если вы сделаете к ней визуальную оболочку. Наример, подобную сделали авторы CodeIgniter'а в своей CMS ExpressionEngine в модуле галереи.

## $this->image\_lib->rotate()

Функция поворота изображения позволяет повернуть изображение на заданный угол:

$config['rotation\_angle'] = '90';

Есть 5 опций поворота:

1. 90 - поворот против часовой стрелки на 90 градусов.
2. 180 - поворот на 180 градусов.
3. 270 - поворот против часовой стрелки на 270 градусов.
4. hor - зеркальное отображение по горизонтали.
5. vrt - зеркальное отображение по вертикали.

Пример поворота изображения:

$config['image\_library'] = 'netpbm';  
$config['library\_path'] = '/usr/bin/';  
$config['source\_image'] = '/path/to/image/mypic.jpg';  
$config['rotation\_angle'] = 'hor';  
$this->image\_lib->initialize($config);   
if ( ! $this->image\_lib->rotate())  
{  
    echo $this->image\_lib->display\_errors();  
}

## $this->image\_lib->clear()

Функция очистки обнуляет все предыдущие параметры. Используется для выполнения последовательных действий над изображением.

$this->image\_lib->clear();

# Создание "водяных знаков"

Функция создания "водяных знаков" требует наличия библиотеки GD/GD2 .

## Два типа "водяных знаков"

Есть два типа "водяных знаков":

* **Наложение текста**: "Водяной знак" будет создан из текста, с использованием шрифтов True Type (если задали), или с помощью стандартных шрифтов. Если используются True Type шрифты, то версия GD должна поддерживать работу с ими.
* **Наложение изображения**: "Водяной знак" будет создан из изображения (обычно это полупрозрачные PNG или GIF) с наложением его поверх исходного изображения.

## Создание "водяных знаков"

Как и при работе с другими функциями сначала следует задать параметры, а затем вызвать функцию обработки. Например:

$config['source\_image'] = '/path/to/image/mypic.jpg';  
$config['wm\_text'] = 'Copyright 2009 - Tutulkin';  
$config['wm\_type'] = 'text';  
$config['wm\_font\_path'] = './system/fonts/texb.ttf';  
$config['wm\_font\_size'] = '16';  
$config['wm\_font\_color'] = 'ffffff';  
$config['wm\_vrt\_alignment'] = 'bottom';  
$config['wm\_hor\_alignment'] = 'center';  
$config['wm\_padding'] = '20';  
$this->image\_lib->initialize($config);   
$this->image\_lib->watermark();

Этот код использует True Type шрифт размера 16 пикселей для создания текста. Текст будет расположен вверху по центру изображения, со смещением 20 пикселей относительно верха.

**Важно:** Что бы изменения сохранились, файл должен быть доступен на "запись". Установите права, например, 0777

## Параметры "водяных знаков"

Эта таблица отображает параметры для обоих типов "водяных знаков"

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Опции** | **Описание** |
| **wm\_type** | text | text, overlay | Устанавливает тип "водяного знака" |
| **source\_image** | Нет | Нет | Указывает путь к исходному файлу |
| **dynamic\_output** | FALSE | TRUE/FALSE (boolean) | Указывает, сохранять ли полученное изображение или использовать для динамического вывода. Примечание: если выбрано динамическое использование, то изображение будет выведено с заголовками изображения - то есть только само изображение, без html и других изображений. |
| **quality** | 90% | 1 - 100% | Устанавливает качество изображения. Чем выше качество, тем больше размер файла. И наоборот. |
| **padding** | Нет | Число | Устанавливает смещение в пикселях. |
| **wm\_vrt\_alignment** | bottom | top, middle, bottom | Устанавливает вертикальное выравнивание. |
| **wm\_hor\_alignment** | center | left, center, right | Устанавливает горизонтальное выравнивание. |
| **wm\_hor\_offset** | Нет | Нет | Устанавливает горизонтальное смещение (в пикселях) относительно выравнивания. Перемещает в сторону, обрантую выравниванию. При центровке - вправо. |
| **wm\_vrt\_offset** | Нет | Нет | Устанавливает вертикальное смещение (в пикселях) относительно выравнивания. Перемещает в сторону, обрантую выравниванию. При центровке - вниз. |

### Параметры текста

Таблица показывает парамерты, применимые к текстовому типу "водяных знаков".

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Опции** | **Описание** |
| **wm\_text** | Нет | Нет | Текст, который желаете вывести. Обычно это копирайты. |
| **wm\_font\_path** | Нет | Нет | Серверный путь к папке с True Type шрифтом, который желаете использовать. Если не задано, то будет использоваться стандартный GD шрифт (не поддерживает кириллицу!). |
| **wm\_font\_size** | 16 | Нет | Размер шрифта. Для стандартных - от 1 до 5. Для TT - любое число (размер в пикселях). |
| **wm\_font\_color** | ffffff | Нет | Цвет шрифта в hex-коде. Замечание: использовать полную hex-запись цвета; запись типа fff является неверной! |
| **wm\_shadow\_color** | Нет | Нет | Цвет тени в hex-коде. Краткая запись так же запрещена. |
| **wm\_shadow\_distance** | 3 | Нет | Расстояние (в пикселях) тени от основной надписи. |

### Параметры наложения рисунка

Таблица показывает парамерты, применимые к "водяным знакам" в виде изображения.

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Значение по умолчанию** | **Опции** | **Описание** |
| **wm\_overlay\_path** | Нет | Нет | Серверный путь с изображению для наложения. Если задан, то будет использоваться данный тип "водяного знака" |
| **wm\_opacity** | 50 | 1 - 100 | Прозрачность накладываемого изображения |
| **wm\_x\_transp** | 4 | Число | Если используется для наложения изображение типа PNG или GIF , то можно задать цвет, который будет "прозрачным". Задается координата пикселя этого цвета. Отсчет от верхнего левого угла; X координата |
| **wm\_y\_transp** | 4 | Число | Y координата "пикселя прозрачности". |

# Класс Input

Класс Input предназначен для решения двух задач:

1. Предварительная обработка глобальных входных данных для обеспечения безопасности.
2. Предоставления вспомогательных функций для извлечения входных данных и их предварительной обработки.

**Примечание:** Этот класс автоматически инициализируется системой, поэтому нет необходимости делать это вручную.

## Безопасная фильтрация

Функция безопасной фильтрации вызывается автоматически при вызове нового [контроллера](http://code-igniter.ru/user_guide/general/controllers.html). При этом происходит следующее:

* Уничтожается глобальный массив GET. Так как CodeIgniter не использует GET строки, то и нет необходимости предоставлять к ним доступ.
* Уничтожаются все глобальные переменные в том случае, если включена опция register\_globals.
* Фильтрация ключей массивов POST/COOKIE. В них разрешается использовать только буквенно-цифровые (и несколько других) символы.
* Производится XSS (Cross-site Scripting атаки) фильтрация. Это функция может быть включена глобально, либо по запросу.
* Преобразование символов новой строки в \n

## XSS фильтрация

CodeIgniter поставляется с фильтром предотвращающим XSS атаки, который может запускаться автоматически и фильтровать все POST и COOKIE-данные, с которыми сталкивается, или же вы может запускать его для каждого элемента вручную. По-умолчанию он **не запускается** глобально поскольку его работа приводит к дополнительному расходу системных ресурсов, в то время как необходимость его применения возникает не всегда.

XSS фильтр выявляет наличие часто используемых способов запуска скриптов JavaScript или других видов кода, которые пытаются перехватить cookie или произвести какие-то другие злонамеренные действия. Если фильтр сталкивается с чем-то запрещенным, то для безопасности он переводит данные в символы-мнемоники.

Примечание: Эту функцию следует использовать только когда это действительно необходимо. Эта не та функция, которую следует применять в общих целях, так как она требует значительного расхода системных ресурсов.

Для фильтрации данный с помощью XSS фильтра используйте эту функцию:

## $this->input->xss\_clean()

Вот пример ее использования:

$data = $this->input->xss\_clean($data);

Если вы хотите, чтобы фильтр запускался автоматически всякий раз, когда используются POST или COOKIE-данные, то откройте файл**application/config/config.php** и установите следующую переменную:

$config['global\_xss\_filtering'] = TRUE;

Примечание: Если вы используете класс Validation, то он также предоставляет возможность XSS фильтрации.

## Использование POST, COOKIE или SERVER-данных

CodeIgniter поставляется с тремя вспомогательными функциями, которые позволяют вам извлекать POST, COOKIE или SERVER-элементы. Главное преимущество этих функций заключается в том, что помимо прямого извлечения элементов ($\_POST['something']), эти функции позволяют проверить существует ли элемент, и, если нет, то возврящают FALSE (boolean). Это дает вам возможность удобного использования данных без предварительной проверки элемента на существование. Другими словами, как правило, вы возможно делаете что-то вроде этого:

if ( ! isset($\_POST['something']))  
{  
    $something = FALSE;  
}  
else  
{  
    $something = $\_POST['something'];  
}

С помощью встроенных функций CodeIgniter того же результата можно достичь следующим образом:

$something = $this->input->post('something');

Вот все эти три функции:

* $this->input->post()
* $this->input->cookie()
* $this->input->server()

## $this->input->post()

Первый параметр будет содержать имя POST элемента, который вы ищете:

$this->input->post('some\_data');

Эта функция возвращает FALSE (boolean), если элемент, который вы пытаетесь получить, не существует.

Второй необязательный параметр позволяет пропустить данные через XSS фильтр. Это становится возможным, если установить второй параметр в TRUE;

$this->input->post('some\_data', TRUE);

## $this->input->cookie()

Эта функция идентична post функции, с тем лишь отличием, что она извлекает COOKIE-данные:

$this->input->cookie('some\_data', TRUE);

## $this->input->server()

Эта функция идентична приведенным выше функциям, с той лишь разницей, что она извлекает SERVER данные:

$this->input->server('some\_data');

## $this->input->ip\_address()

Возвращает IP адрес текущего пользователя. Если IP адрес неверен, то функция вернет IP адрес: 0.0.0.0

echo $this->input->ip\_address();

## $this->input->valid\_ip($ip)

На входе получает IP адрес и возвращает TRUE или FALSE (boolean), в зависимости от того, является ли адрес правильным или нет. Примечание: приведенная выше функция $this->input->ip\_address() проверяет IP адрес автоматически.

if ( ! $this->input->valid\_ip($ip))  
{  
     echo 'Not Valid';  
}  
else  
{  
     echo 'Valid';  
}

## $this->input->user\_agent()

Возвращает название пользовательского агента (web-браузера), используемого текущим пользователем. Возвращает FALSE, если это имя недоступно.

echo $this->input->user\_agent();

# Класс Loader

Как следует из имени, класс Loader используется для загрузки различных компонентов. Ими могут быть [отображения](http://code-igniter.ru/user_guide/general/views.html), [хелперы](http://code-igniter.ru/user_guide/general/helpers.html), [плагины](http://code-igniter.ru/user_guide/general/plugins.html), или ваши собственные файлы.

**Замечание:** Этот класс инициализируется системой автоматически, поэтому нет нужды делать это самому.

Имеются следующие функции:

## $this->load->library('имя\_класса')

Эта функция используется для загрузки основных классов, где **имя\_класса** — имя класса, который вы желаете загрузить. Замечание: Используемые нами термины «класс» и «библиотека» взаимозаменяемы.

Например, если вам надо будет отсылать письма из CodeIgniter, первым шагом будет загрузка класса email в контроллер:

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

Будучи однажды загруженной, библиотека становится пригодной для работы с использованием **$this->email->любая\_функция()**. Каждая библиотека подробно описывается на специально посвященной ей странице, поэтому черпайте полную информацию по конкретным вопросам именно там.

Аргументы могут быть переданы в библиотеку в виде массива (второй аргумент).

## $this->load->view('имя\_файла', $data, true/false)

Эта функция используется для загрузки файлов отображений. Если Вы еще не познакомились с разделом [отображений](http://code-igniter.ru/user_guide/general/views.html) руководства пользователя, то рекомендуем сделать это, поскольку там рассказано о типичном применении этой функции.

Первый аргумент обязателен. Это имя файла-отображения, который надо загрузить.  Замечание: Расширение .php, в отличие от остальных, можно не указывать. Если загружаемый файл не .php, то расширение необходимо указывать.

Вторым **необязательным** аргументом является либо ассоциативный массив, либо вводимый объект, который пропускается через функцию PHP [extract](http://www.php.net/extract) для извлечения и превращения в переменные, которые могут использоваться вашим отображением. Снова отсылаем вас к странице [с описанием отображений](http://code-igniter.ru/user_guide/general/views.html) для объяснения того, когда это может оказаться полезным.

Третий **необязательный** аргумент позволяет изменить поведение функции так, чтобы она возвращала данные в виде строки, а не отправляла их в браузер. Это может оказаться полезным тогда, когда Вам надо каким-либо образом поработать с этими данными. Если аргумент указан как **true**(boolean), функция возвратит данные. Значением по умолчанию является**false**, и тогда данные попадают в браузер. Таким образом, помните, что указать аргумент необходимо, если нужен возврат, а не вывод данных:

$string = $this->load->view('**myfile**', '', **true**);

## $this->load->database('варианты', true/false)

Эта функция позволяет загрузить класс базы данных. Оба аргумента**необязательны**. Обращайтесь к разделу «[Класс работы с базами данных](http://code-igniter.ru/user_guide/libraries/database/index.html)» за более полной информацией.

## $this->load->scaffolding('имя \_таблицы')

Эта функция включает [скаффолдинг](http://code-igniter.ru/user_guide/general/scaffolding.html).

## $this->load->vars($array)

Эта функция принимает ассоциативный массив и создает из него переменные, используя функцию PHP [extract](http://www.php.net/extract). Эта функция дает тот же результат, что и использование второго аргумента в функции **$this->load->view()**, которая описана выше. Причина, по которой данная функция может понадобиться отдельно, может возникнуть тогда, когда вы захотите, например, установить какие-то глобальные переменные в конструкторе контроллера и сделать их доступными в любом файле отображения, загружаемом какой-либо фунцией. Вызывать данную функцию можно неоднократно. Данные в этом случае будут кэшированы и помещены в единый массив для дальнейшего конвертирования в переменные.

## $this->load->helper('имя\_файла')

Эта функция загружает файл хелпера, где **имя\_файла** является частью имени файла без **\_helper.php**.

## $this->load->plugin('имя\_файла')

Эта функция загрузит файлы плагинов, где **имя\_файла** является частью имени файла без **\_plugin.php**.

## $this->load->file('путь/файл', true/false)

Это основная функция загрузки файлов. При наличии указанного пути к файлу (первый аргумент), функция откроет и прочитает этот файл. По умолчанию данные посылаются в браузер, как из файла отображения, но если указать второй аргумент как **true** (boolean), данные будут возвращены в строковом виде.

## $this->load->lang('имя\_файла')

Эта функция является синонимом [функции загрузки языка](http://code-igniter.ru/user_guide/libraries/language.html): $this->lang->load()

## $this->load->config('имя\_файла')

Эта функция является синонимом [функции загрузки конфигурационного файла](http://code-igniter.ru/user_guide/libraries/config.html): $this->config->load()

# Класс Language

Класс Language предоставляет функции для создания языковых файлов и строк текста для интернационализации.

В вашей системной папке CodeIgniter вы можете найти папку **language**содержащую языковые файлы. Вы можете создать файлы с вашим языком для отображения ошибок и других сообщений на вашем языке.

Языковые файлы, обычно, хранятся в папке **system/language**. Так же вы можете создать папку **language** внутри вашей директории **application** и хранить их там. CodeIgniter сначала просмотрит папку**system/application/language**. Если таковая директория не существует или там не лежат файлы для указанного языка, CI будет искать их в глобальной папке **system/language**.

**Замечание:**  Каждый язык должен быть сохранен в своей папке. Например, файлы английского языка расположены в **system/language/english**.

## Создание языковых файлов

Языковые файлы должны называться с окончанием **\_lang.php**. Например, вы хотите создать файл содержаший сообщения об ошибках. Вы могли бы назвать его **error\_lang.php**.

Внутри файла вы будете присваивать каждую строку текста массиву с названием **$lang** по следующему примеру:

$lang['language\_key'] = "Сообщение, которое должно быть отображено";

**Замечание:** Хорошим примером является использование ключей массива с префиксом, соответствующим названию файла для всех сообщений, чтобы избежать пересечения с прочими сообщениями в других файлах. Например, для файла с ошибками таким префиксом будет **error\_**.

$lang['**error**\_email\_missing'] = "Вы должны указать верный email";  
$lang['**error**\_url\_missing'] = "Вы должны указать URL";  
$lang['**error**\_username\_missing'] = "Вы должны указать имя пользователя";

## Загрузка языкового файла

Для того чтобы получить языковую строку вам необходимо сначала загрузить файл. Загрузка файла выполняется следующей строкой кода:

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

Где **filename** это название вашего языкового файла (без расширения), и**language** это язык (например english). Второй параметр не обязателен, т.к. его значение по умолчанию может быть объявлено в файле**application/config/config.php**.

## Получение строки текста

После загрузки файла вы можете получить конкретный перевод при помощи функции:

$this->lang->line('**language\_key**');

Где **language\_key** соответствует ключу массива, в котором содержится перевод.

Замечание: Эта функция просто вернет строку. Строка не будет выведена, она просто будет возвращена.

## Авто-загрузка языка

Если вы обнаружите, что вам необходим какой-то язык в глобальном масштабе, то можете сообщить CodeIgniter, что его необходимо [загружать автоматически](http://code-igniter.ru/user_guide/general/autoloader.html) при инициализации системы. Это можно сделать открыв файл application/config/autoload.php и добавив язык(и) в массив автозагрузки.

# Класс Output

Класс Output — это маленький класс с одной главной функцией: отправка готовой страницы в браузер посетителя. Он также отвечает за [кеширование](http://code-igniter.ru/user_guide/general/caching.html)вашей страницы, если вы пользуетесь этой функцией.

**Замечание:** Этот класс инициализируется системой автоматически, поэтому нет необходимости делать это вручную.

При обычных обстоятельствах, вы даже не заметите, как работает этот класс, поскольку он работает без вашего вмешательства. Например, когда вы используете класс [Loader](http://code-igniter.ru/user_guide/libraries/loader.html) для загрузки ваших отображений, эта задача автоматически передается классу Output, который будет самостоятельно вызван в конце исполнения программы. Однако, вполне возможно вмешаться в процесс вывода, используя одну из двух функций:

## $this->output->set\_output();

Позволяет вручную установить строку вывода. Используйте пример:

$this->output->set\_output($data);

**Важно:** Если вы производите вывод вручную, то это последнее, что вы должны вызвать из функции. Например, если вы создаете страницу в одной из функций вашего контроллера, не производите вывод, пока не выполнилась вся функция.

## $this->output->get\_output();

Позволяет вам получить весь вывод, который был отправлен к классу Output. Используйте пример:

$string = $this->output->get\_output();

Заметим, что эта функция что-то вернет, только если предварительно данные были отправлены в класс Output функциями вроде **$this->load->view()**.

## $this->output->set\_header();

Позволяет установить вам серверные заголовки, которые будут отправлены перед отправкой информации в браузер пользователя. Пример:

$this->output->set\_header("HTTP/1.0 200 OK");  
$this->output->set\_header("HTTP/1.1 200 OK");  
$this->output->set\_header('Last-Modified: '.gmdate('D, d M Y H:i:s', $last\_update).' GMT');  
$this->output->set\_header("Cache-Control: no-store, no-cache, must-revalidate");  
$this->output->set\_header("Cache-Control: post-check=0, pre-check=0", false);  
$this->output->set\_header("Pragma: no-cache");

## $this->output->enable\_profiler();

Позволяет вам включить/отключить [Профилирование приложения](http://code-igniter.ru/user_guide/general/profiling.html), которое будет отображать производительность и другие данные внизу на ваших страницах для отладки и оптимизации приложения.

Для включения профилирования разместите следующий код в любом месте функции вашего [контроллера](http://code-igniter.ru/user_guide/libraries/controllers.html):

$this->output->enable\_profiler(TRUE);

Когда профилирование включено, то вывод отладочной информации происходит после вывода всех отображений.

Для отключения профилирования используйте:

$this->output->enable\_profiler(FALSE);

# Класс Pagination

Класс Pagination очень прост в использовании, и на 100% настраиваемый, динамически или посредством сохраненных настроек.

Если Вы не знакомы с термином "pagination" (нумерация страниц, разбивка на страницы), то это набор ссылок позволяющих перемещаться по страницам, например:

[« В начало](http://code-igniter.ru/user_guide/libraries/pagination.html)  [1](http://code-igniter.ru/user_guide/libraries/pagination.html)  [2](http://code-igniter.ru/user_guide/libraries/pagination.html)  **3**  [4](http://code-igniter.ru/user_guide/libraries/pagination.html)  [5](http://code-igniter.ru/user_guide/libraries/pagination.html)  [В конец »](http://code-igniter.ru/user_guide/libraries/pagination.html)

## Пример

На этом примере показывается, как можно создать нумерацию страниц в одном из ваших [контроллеров](http://code-igniter.ru/user_guide/general/controllers.html):

$this->load->library('pagination');  
$config['base\_url'] = 'http://www.your-site.com/index.php/test/page/';  
$config['total\_rows'] = '200';  
$config['per\_page'] = '20';   
$this->pagination->initialize($config);   
echo $this->pagination->create\_links();

### Примечание:

Массив **$config** содержит конфигурационные переменные. Он передается в функцию **$this->pagination->initialize**, как показано выше. Хотя массив имеет двенадцать опций, для работы класса необходимо задать минимум три:

* **base\_url** — это полный путь к классу контроллера/функции содержащей Вашу нумерацию страниц. В примере выше контроллер «Test», а функция «page». Имейте ввиду, что можно [изменить URI](http://code-igniter.ru/user_guide/general/routing.html), если вам необходима другая структура.
* **total\_rows** — это общее число страниц, которые будут в результате. Обычно это количество записей, возвращаемых запросом к базе данных.
* **per\_page** — это количество элементов, которое вы намерены показать на странице. В примере выше, будет выводится двадцать элементов на страницу.

Функция **create\_links()** вернет пустую строку, когда не будет необходимости показывать нумерацию.

### Настройка параметров в файле конфигурации

Если вас не устраивает предыдущий метод, то вы можете поместить все настройки в конфигурационный файл. Для этого необходимо создать файл с именем **pagination.php** и определить внутри массив **$config**. Далее сохраните файл в каталоге **config/pagination.php** и он будет загружаться автоматически. Теперь вам НЕ надо писать **$this->pagination->initialize**.

## Настройка нумерации страниц

Ниже перечислен полный список параметров, которые вы можете передать в функцию инициализации.

#### $config['uri\_segment'] = 3;

Функция нумерации страниц автоматически определяет, какой сегмент URI содержит номер страницы. Если Вам необходимо явно указать сегмент, можно задать его этим параметром.

#### $config['num\_links'] = 2;

Количество ссылок, которое будет перед и после номера выбранной страницы. Например, при указании этого параметра равным 2, будет по два числа с каждой стороны, как в примере выше.

## Добавление элементов разметки

Если вы хотите заключить блок нумерации страниц в другой элемент разметки, то можете сделать это с помощью следующих двух параметров:

#### $config['full\_tag\_open'] = '<p>';

Открывающий тег.

#### $config['full\_tag\_close'] = '</p>';

Закрывающий тег.

## Настройка ссылки, указывающей на первую страницу

#### $config['first\_link'] = 'В начало';

С помошью этого параметра задается текст ссылки, указывающей на первую страницу. Например, "На первую страницу".

#### $config['first\_tag\_open'] = '<div>';

Открывающий тег.

#### $config['first\_tag\_close'] = '</div>';

Закрывающий тег.

## Настройка ссылки, указывающей на последнюю страницу

#### $config['last\_link'] = 'Last';

С помошью этого параметра задается текст ссылки, указывающей на последнюю страницу. Например, "На последнюю страницу".

#### $config['last\_tag\_open'] = '<div>';

Открывающий тег.

#### $config['last\_tag\_close'] = '</div>';

Закрывающий тег.

## Настройка ссылки, указывающей на следующую страницу

#### $config['next\_link'] = 'Далее';

С помошью этого параметра задается текст ссылки, указывающей на следующую страницу. Например, "Далее".

#### $config['next\_tag\_open'] = '<div>';

Открывающий тег.

#### $config['next\_tag\_close'] = '</div>';

Закрывающий тег.

## Настройка ссылки, указывающей на предыдущую страницу

#### $config['prev\_link'] = 'Назад';

С помошью этого параметра задается текст ссылки, указывающей на предыдущую страницу. Например, "Назад".

#### $config['prev\_tag\_open'] = '<div>';

Открывающий тег.

#### $config['prev\_tag\_close'] = '</div>';

Закрывающий тег.

## Настройка ссылки, указывающей на текущую страницу

#### $config['cur\_tag\_open'] = '<b>';

Открывающий тег.

#### $config['cur\_tag\_close'] = '</b>';

Закрывающий тег.

## Настройка числовых ссылок

#### $config['num\_tag\_open'] = '<div>';

Открывающий тег.

#### $config['num\_tag\_close'] = '</div>';

Закрывающий тег.

# Класс для работы с сессиями

Класc Session позволяет вам хранить информацию о пользователе и отслеживать его активность пока он находятся на вашем сайте. Класс Session хранит информацию о каждом пользователе как упорядоченную(при необходимости зашифрованную) информацию в cookies. Так же он может сохранять информацию в таблице базы данных - для дополнительной безопасности, это позволяет сопоставлять идентификатор сессии сохраненный в кукисах с идентификатором сохраненным в базе данных. По умолчанию информация сохраняется лишь в cookies. Если вы захотите использовать базу данных для хранения информации о сессии, то вам будет необходимо создать таблицу как указано ниже.

**Примечание:** Класс Session **не** использует родные PHP сессии. Он генерирует свои специфические данные, что позволяет разработчикам получить более гибкие возможности.

## Создание сессии

Класс Session, как правило, необходим на каждой странице, так что он должен быть [инициализирован](http://code-igniter.ru/user_guide/general/libraries.html) в конструкторе вашего [контроллера](http://code-igniter.ru/user_guide/general/controllers.html), либо автоматически подключаться с помощью **application/config/autoload.php**.

Для того чтобы инициализировать класс Session вручную в конструкторе вашего контроллера используйте функцию **$this->load->library**:

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

После загрузки объект класс Session станет доступным: **$this->session**

## Как работает класc Session ?

Когда страница загружена класс Session проверит - есть ли в cookies пользователя корректные на данный момент данные сессии. Если данных нет(или они устарели) то будет создана новая сессия, которая будет сохранена в cookies. Если сессия существует, то информация о ней будет обновлена, соответственно cookies тоже будет обновлены. С каждым обновлением session\_id будет заменятся на вновь сгенерированный session\_id.

Важно понять, что будучи инициализированным класс Session далее работает самостоятельно. Нет ничего, что вам предстоит сделать для того чтобы он работал. Вы сможете работать с данными сессии, или даже добавлять собственные данные в сессию пользователя, однако процесс чтения, записи и обновления сессии является автоматическим.

## Что такое данные сессии?

Сессия является массивом содержащим следующую информацию::

* Уникальный ID сессии (случайная строка, закодированная алгоритмом md5, генерируется заново каждые 5 минут(по умолчанию))
* IP адрес пользователя
* User Agent пользователя(первые 50 символов)
* Время последней активности(timestamp)

Вся эта информация хранится в cookies как упорядоченный массив, соответствующий этому примеру:

[array]  
(  
     'session\_id'    => random hash,  
     'ip\_address'    => 'string - user IP address',  
     'user\_agent'    => 'string - user agent data',  
     'last\_activity' => timestamp  
)

Если вы используете возможность шифрования данных сессии, то массив будет сначала зашифрован, а затем сохранен в cookies пользователя. Это повышает безопасность хранения информации и делает невозможным ее чтение либо изменение третьими лицами. Больше про шифрование вы можете найти [здесь](http://code-igniter.ru/user_guide/libraries/encryption.html), класс Session производит шифрование данных автоматически.

**Примечание:** Чтобы снизить нагрузку на процессор, по умолчанию cookies обновляются каждые 5 минут. Если вы часто обновляете страницу, то можете заметить, что время последней активности обновляется только тогда, когда пройдет около пяти минут, с того времени как cookies были записаны в прошлый раз. Это время можно регулировать, изменяя параметр $config['time\_to\_update'] в файле system/config/config.php

## Получение данных сессии

Любой элемент массива сессии становится доступным с помощью следующей функции:

$this->session->userdata('**item**');

Где **item** является индексом массива сессии. Например, для того чтобы получить ID сессии необходимо сделать так:

$session\_id = $this->session->userdata('**session\_id**');

**Примечание:** Функция возвращает FALSE если параметра который вы пытаетесь получить не существует.

## Добавление данных в сессию

Сессии позволяют сохранять собственную информацию в cookies пользователя. Зачем это нужно? Вот пример:

Допустим, что пользователь залогинился на вашем сайте. После авторизации вы можете добавить его username и email в cookies сессии, что сделает эту информацию доступной везде - без необходимости подключения к базе данных.

Для того чтобы добавить информацию в массив сессии необходимо передать собственный массив(массив с новой информацией) в эту функцию:

$this->session->set\_userdata(**$array**);

Где **$array** является ассоциативным массивом, который содержит вашу новую информацию. Пример:

$newdata = array(  
                   'username'  => 'johndoe',  
                   'email'     => 'johndoe@some-site.com',  
                   'logged\_in' => TRUE  
               );  
$this->session->set\_userdata(**$newdata**);

Если вам необходимо добавить одно значение, то вы можете воспользоваться следующим синтаксисом функции set\_userdata():

$this->session->set\_userdata('some\_name', 'some\_value');

**Примечание:** cookies могут содержать только 4kb информации, следите за тем чтобы не превысить этот объем. Зашифрованная информация занимает больше места чем не зашифрованная.

## Удаление данных сессии

Так же как set\_userdata() может быть использована для добавления информации в сессию, unset\_userdata() может быть использована для удаления этой информации. Например, если вы хотите удалить 'some\_name' из вашей сессии:

$this->session->unset\_userdata('some\_name');

Эта функция так же поддерживает ассоциативный массив элементов, которые необходимо удалить:

$array\_items = array('username' => '', 'email' => '');  
$this->session->unset\_userdata(**$array\_items**);

## Flashdata

CodeIgniter поддерживает "flashdata", эта возможность позволяет сделать данные сессии доступными только для следующего запроса сервера, а после этого они будет автоматически удалены. Эта возможность может оказаться очень полезной, чаще всего ее используют для хранения статусных сообщений(например: "запись 2 удалена").

**Примечание:** Flash данные начинаются с префикса "flash\_", так что избегайте его в именах своих собственных данных.

Для того чтобы добавить flashdata необходимо:

$this->session->set\_flashdata('item', 'value');

Вы можете передать массив в качестве параметра set\_flashdata(), это выглядит так же как и с функцией set\_userdata().

Для того чтобы прочесть переменную flashdata необходимо:

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

Если вам необходимо сохранить flashdata переменную, т.е. сохранить ее на еще один запрос сервера, вы можете сделать это с помощью функции keep\_flashdata():

$this->session->keep\_flashdata('item');

## Сохранение данных о сессии в базу данных

Пока информация хранящаяся в cookies пользователя содержит ID сессии, вы не имеете возможности проверить его, в отличие от варианта когда данные сессии хранятся в базе данных. Для приложений, которые не требуют высокого уровня безопасности, проверка ID сессии не является обязательной, однако, если ваше приложение требует высокого уровня безопасности, то проверка становится обязательной.

Если данные сессии находятся в базе данных, то каждый раз, когда в cookies пользователя обнаруживается рабочая сессия, осуществляется запрос к базе данных - с целью сравнить ID сессий. Если ID сессии не совпадают, то сессия разрушается. ID сессии никогда не обновляется, он может быть лишь сгенерированным, когда сессия создается.

Если вы хотите размещать сессии в базе данных, то сперва необходимо создать таблицу для этих целей. Ниже приведен простейший пример для субд mysql:

CREATE TABLE IF NOT EXISTS `ci\_sessions` (

session\_id varchar(40) DEFAULT '0' NOT NULL,

ip\_address varchar(16) DEFAULT '0' NOT NULL,

user\_agent varchar(50) NOT NULL,

last\_activity int(10) unsigned DEFAULT 0 NOT NULL,

user\_data text NOT NULL,

PRIMARY KEY (session\_id)

);

**Примечание:** по умолчанию таблица называется **ci\_sessions**, однако вы можете назвать ее как захотите - для этого необходимо изменить файл**application/config/config.php**. После того как вы создали таблицу вы можете активировать возможность сохранения данных сессии в базе данных в вашем config.php:

$config['sess\_use\_database'] = TRUE;

После того как эта возможность активирована, класс Session будет размещать данные в базе данных.

Убедитесь, что вы правильно указали название таблицы в файле конфигурации:

$config['sess\_table\_name'] = 'ci\_sessions";

**Примечание:** Класс Session содержит встроенный чистильщик мусора, который удаляет истекшие сессии, так что у вас нет необходимости писать собственный.

## Уничтожение сессий

Для того чтобы очистить текущую сессию:

$this->session->sess\_destroy();

**Примечание:** Эта функция должна вызываться последней, после ее вызова даже flash переменные становятся не доступными. Если вам необходимо уничтожить лишь некоторые переменные, то воспользуйтесь**unset\_userdata()**.

## Настройка класса Session

Все ниже представленные настройки содержатся в файле **application/config/config.php**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Значение** | **По умолчанию** | **Варианты** | **Описание** |
| **sess\_cookie\_name** | ci\_session | Нет | Имя cookies куда будет сохранятся сессия. |
| **sess\_expiration** | 7200 | Нет | Количество секунд по прошествии которых сессия считается истекшей. Значение по умолчанию 2 часа. Ели вы ходите чтобы ваша сессия была бесконечной, то выставляйте значение 0. |
| **sess\_encrypt\_cookie** | FALSE | TRUE/FALSE (boolean) | Шифровать ли информацию сессии? |
| **sess\_use\_database** | FALSE | TRUE/FALSE (boolean) | Размещать ли информацию о сессии в базе данных? (Вы должны создать таблицу до того как активируете эту опцию). |
| **sess\_table\_name** | ci\_sessions | Любое SQL-корректное имя таблицы | Имя таблицы где будет хранится информация о сессиях. |
| **sess\_time\_to\_update** | 300 | Время в секундах | Опция отвечает за то как часто класс Session будет регенирировать ID сессии. |
| **sess\_match\_ip** | FALSE | TRUE/FALSE (boolean) | Сверять ли IP пользователя, во время чтения данных сессии? (Обратите внимание, что некоторые интернет провайдеры динамически меняют IP. Так что если вы хотите использовать бесконечные сессии, то выставьте данную опцию в значение FALSE) |
| **sess\_match\_useragent** | TRUE | TRUE/FALSE (boolean) | Сверять ли User Agent во время чтения данных сессии? |

# Класс Trackback

Класс Trackback позволяет использовать технологию трэкбэков в Ваших приложениях

Более подробно о технологии Trackback можно почитать [здесь](http://ru.wikipedia.org/wiki/Trackback).

## Инициализация Класса

Как и многие другие классы в CodeIgniter, класс Trackback инициализируется функцией **$this->load->library**:

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

После этого библиотека будет доступна по вызову **$this->trackback**

## Отправка трэкбэка (trackback)

Trackback может быть отправлен с любого контроллера с использованием следующего кода:

$this->load->library('trackback');  
$tb\_data = array(  
                'ping\_url'  => 'http://example.com/trackback/456',  
                'url'       => 'http://www.my-example.com/blog/entry/123',  
                'title'     => 'Заголовок',  
                'excerpt'   => 'Текст.',  
                'blog\_name' => 'Название блога',  
                'charset'   => 'utf-8'  
                );  
if ( ! $this->trackback->send($tb\_data))  
{  
     echo $this->trackback->display\_errors();  
}  
else  
{  
     echo 'Trackback успешно отправлен!';  
}

Описание параметров массива :

* **ping\_url** - URL статьи сайта, которому посылается трэкбэк. Можно отсылать трэкбэки нескольким сайтам, разделяя их URL запятой.
* **url** - URL ВАШЕГО сайта, где размещена информация, связанная с ping\_url.
* **title** - Заголовок Вашей статьи.
* **excerpt** - Содержание записи. Примечание: класс Trackback автоматически посылает первые 500 символов записи без учета форматирования.
* **blog\_name** - Имя Вашего блога/сайта.
* **charset** - Кодировка, в которой написана статья. Обычно UTF-8 или CP-1251.

Функция возвращает значения TRUE/FALSE (boolean), если trackback отправлен/не отправлен. Вы можете посмотреть сообщение об ошибке следующим способом:

$this->trackback->display\_errors();

## Получение трэкбэков

Перед тем, как Вы сможете получать Trackback'и, следует создать блог/статьи, иначе получать трэкбэки не имеет смысла.

Система получения трэкбэков более сложная, чем отправки. Необходима база данных для хранения бэков. Каждый трэкбэк следует валидировать (проверить и подтвердить). Необходимо будет создать защиту от спама, например ограничить количество трэкбэков, создать спам-списки сайтов и IP-адресов и т. д. Сам процесс получения не так сложен, как процесс валидации.

## Ваш Ping URL

Для получения Trackback'ов следует создать постоянный URL для каждой статьи, что бы другие люди могли посылать Вам трэкбэки по этому адресу ("Ping URL").

Следует создать контроллер обработки трэкбэков. В Ping URL должен передаваться идентификатор записи ( id )

Например, контроллер-обработчик назван **Trackback**, функция **receive**получает трэкбэк. Тогда Ваш Ping URLбудет выглядеть примерно так:

http://www.your-site.com/index.php/trackback/receive/**entry\_id**

Где **entry\_id** это идентификатор записи, которой предназначается трэкбэк.

## Создание таблицы трэкбэков

Перед получением Trackback'ов следует создать место для их хранения. Рассмотрим это на примере таблицы trackbacks в бд MySQL:

CREATE TABLE trackbacks (

tb\_id int(10) unsigned NOT NULL auto\_increment,

entry\_id int(10) unsigned NOT NULL default 0,

url varchar(200) NOT NULL,

title varchar(100) NOT NULL,

excerpt text NOT NULL,

blog\_name varchar(100) NOT NULL,

tb\_date int(10) NOT NULL,

ip\_address varchar(16) NOT NULL,

PRIMARY KEY (tb\_id),

KEY (entry\_id)

);

Как видим, таблица включает не только обязательные параметры(имя записи, текст трэкбэка), но и вспомогательные, для расширения функциональности (дата, IP отправителя и т.д.)

## Обработка трэкбэков

Этот пример показывает способ получения и обработки Trackback'ов.

$this->load->library('trackback');  
$this->load->database();  
if ($this->uri->segment(3) == FALSE)  
{  
    $this->trackback->send\_error("Не указан ID записи ");  
}  
if ( ! $this->trackback->receive())  
{  
    $this->trackback->send\_error("Trackback содержит некорректные данные!");  
}  
$data = array(  
                'tb\_id'      => '',  
                'entry\_id'   => $this->uri->segment(3),  
                'url'        => $this->trackback->data('url'),  
                'title'      => $this->trackback->data('title'),  
                'excerpt'    => $this->trackback->data('excerpt'),  
                'blog\_name'  => $this->trackback->data('blog\_name'),  
                'tb\_date'    => time(),  
                'ip\_address' => $this->input->ip\_address()  
                );  
$sql = $this->db->insert\_string('trackbacks', $data);  
$this->db->query($sql);  
$this->trackback->send\_success();

#### Примечания:

ID записи должен быть 3 параметром в URL. Например:

http://www.your-site.com/index.php/trackback/receive/**10**

Получение 3 параметра осуществляется кодом :

$this->uri->segment(3);

Функция **$this->trackback->receive()** производит простую проверку на наличие обязательных данных (url, заголовок, имя блога). Она возвращает TRUE при успешной проверке или FALSE, если произошла ошибка (текст ошибки можно вывести ф-цией, расположенной в коде).

Также данные Trackback'а могут быть проверены с помощью функции:

$this->trackback->data('**item**')

Где **item** это один из обязательных параметров: url, title, excerpt, или blog\_name

Если Trackback успешно получен, можно отправить об этом сообщение, использую функцию:

$this->trackback->send\_success();

**Важно:** Приведенный выше пример не содежрит валидации (проверки) входящих данных на спам. Надеемся, Вы сами сможете разработать данную проверку.

Класс Template Parser

Класс Template Parser позволяет разбирать псевдо-переменные расположенные в ваших файлах Отображения. Он позволяет разбирать как простые переменные, так и переменные парных тегов. Если вы никогда не использовали шаблонизаторы, то псевдо-переменные выглядят так:

<html>  
<head>  
<title>{blog\_title}</title>  
</head>  
<body>  
<h3>{blog\_heading}</h3>  
{blog\_entries}  
<h5>{title}</h5>  
<p>{body}</p>  
{/blog\_entries}  
</body>  
</html>

Эти переменные на самом деле не PHP переменные, скорее это обычные текстовые представления, которые позволяют вам отделить PHP код от ваших шаблонов (файлов Отображения).

**Примечание:** CodeIgniter **не требует** от вас использовать этот класс, так как использование «чистого» PHP в ваших страницах Отображения позволяет им выполняться немного быстрее. Тем не менее, некоторые разработчики предпочитают использовать шаблонизаторы, особенно в тех случаях, когда они работают с дизайнерами, у которых работа с PHP может вызвать затруднения.

**Еще примечание:** Класс Template Parser **не полнофункциональный**шаблонизатор. Мы специально оставили его очень простым, чтобы сохранить максимум производительности.

## Инициализация класса

Как и большинство других классов в CodeIgniter, класс Parser инициализируется в вашем контроллере при помощи функции **$this->load->library**:

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

Однажды загруженный, объект библиотеки Parser будет доступен с помощью:**$this->parser**

В этой библиотеке доступны следующие функции:

## $this->parser->parse()

Этот метод принимает имя шаблона и массив входных данных, и генерирует разобранную версию шаблона. Пример:

$this->load->library('parser');  
$data = array(  
            'blog\_title' => 'My Blog Title',  
            'blog\_heading' => 'My Blog Heading'  
            );  
$this->parser->parse('blog\_template', $data);

Первый параметр содержит имя файла [Отображения](http://code-igniter.ru/user_guide/general/views.html) (в этом примере этот файл назван blog\_template.php), а второй параметр содержит ассоциативный массив с данными, которые необходимо заменить в шаблоне. В вышеприведенном примере, шаблон содержт две переменные: {blog\_title} и {blog\_heading}

Нет никакой необходимости выводить «echo» или делать что-то с данными возвращаемых **$this->parser->parse()**. Данные автоматически передаются в Output класс для отправки в броузер. Однако если вы хотите, чтобы функция все-таки возвращала данные, вместо отправки их в Output класс, вам необходимо передать TRUE (boolean) в качестве третьего параметра:

$string = $this->parser->parse('blog\_template', $data, TRUE);

## Парные переменные

Вышеприведенный пример показывает как заменять простые переменные. Но что, если вам необходимо заменить переменные в текущем блоке, которые при каждой итерации содержат новые значения? Пример рассматриваемого шаблона мы привели в начале этой страницы:

<html>  
<head>  
<title>{blog\_title}</title>  
</head>  
<body>  
<h3>{blog\_heading}</h3>  
{blog\_entries}  
<h5>{title}</h5>  
<p>{body}</p>  
{/blog\_entries}  
</body>  
</html>

В вышеприведенном примере вы возможно уже заметили парные переменные:**{blog\_entries}** данные... **{/blog\_entries}**. В случае подобного этому, фрагмент данных между этими парными переменными будет повторяться несколько раз, в соотвествии с количеством строк в результате.

Разбор парных переменных происходит с использованием идентичного кода, приведенного выше и показывающего разбор одиночных переменных, за исключением того, что в данном случае еще добавляется многомерных массив соответвующий данным парных переменных. Рассмотрим это пример:

$this->load->library('parser');  
$data = array(  
              'blog\_title'   => 'My Blog Title',  
              'blog\_heading' => 'My Blog Heading',  
              'blog\_entries' => array(  
                                      array('title' => 'Title 1', 'body' => 'Body 1'),  
                                      array('title' => 'Title 2', 'body' => 'Body 2'),  
                                      array('title' => 'Title 3', 'body' => 'Body 3'),  
                                      array('title' => 'Title 4', 'body' => 'Body 4'),  
                                      array('title' => 'Title 5', 'body' => 'Body 5')  
                                      )  
            );  
$this->parser->parse('blog\_template', $data);

Если ваши «парные» данные поступают из результата запроса к базе данных, который уже является многомерным массивом, то вы можете просто использовать функцию базы данных result\_array():

$query = $this->db->query("SELECT \* FROM blog");  
$this->load->library('parser');  
$data = array(  
              'blog\_title'   => 'My Blog Title',  
              'blog\_heading' => 'My Blog Heading',  
              'blog\_entries' => $query->result\_array()  
            );  
$this->parser->parse('blog\_template', $data);

# Класс Оформления

Класс Оформления помогает отформатировать текст.

## Инициализация класса

Как и многие другие классы в CodeIgniter, данный класс инициализируется функцией **$this->load->library**:

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

После этого класс будет доступен по вызову: **$this->typography**

## auto\_typography()

Форматирует текст, что бы он был верно оформлен с точки зрения HTML. Получает строку и возвращает ее с следующими изменениями:

* Заключает параграфы в теги <p></p> (два перевода строки считаются границей параграфа).
* Переводы строки конвертируются в <br />, если они не находятся внутри тега <pre>.
* Блочные элементы, типа <div>, не разделяются параграфами, но при необходимости разделяется текст внутри их.
* Кавычки конвертируются в энтити, если они не являются частью тега и его параметром .
* Апострофы конвертируются в html-энтити .
* Двойные тире (напр., -- ) конвертируются в — тире.
* Три последовательных точки заменяются символом многоточия …
* Двойные пробелы после предложений переводятся в неразрывной пробел.

Пример использования :

$string = $this->typography->auto\_typography($string);

### Параметры

Функция содержит два необязательных параметра:

1. **Удаление кода JavaScript**. Параметр позволяет убрать JS-код из текста по причине безопасности. Использование - логическое **TRUE** или **FALSE**.
2. **Уменьшение пустых строк**. Позволяет заменить три и более пустых строк двумя. Использование - логическое **TRUE**или **FALSE**.

По умолчанию функция убирает JS и не заменяет пустые строки, то есть выглядит так

$string = $this->typography->auto\_typography($string, **TRUE**, **FALSE**);

**Примечание:** Если форматирование будет применяться достаточно часто, то можно или сохранить отформатированный текс в базе, или использовать функцию [кэширования](http://code-igniter.ru/user_guide/general/caching.html) .

## convert\_characters()

Функция подобна **auto\_typography** , но она работает только с символами:

* Кавычки конвертируются в энтити, если они не являются частью тега и его параметром .
* Апострофы конвертируются в html-энтити .
* Двойные тире (напр, -- ) конвертируются в — тире.
* Три последовательных точки заменяются символом многоточия …
* Двойные пробелы после предложений переводятся в неразрывной пробел.

Пример:

$string = $this->typography->convert\_characters($string);

## nl2br\_except\_pre()

Переводы строки конвертируются в <br />, если они не находятся внутри тега <pre>. Функция подобна стандартной функции PHP **nl2br()**, но стандартная игнорирует тег <pre>.

Пример:

$string = $this->typography->nl2br\_except\_pre($string);

## protect\_braced\_quotes

При использовании библиотеки Оформления с библиотекой Парсера шаблонов порой надо оставить кавычки между фигурными скобками. Для этого используйте свойство **protect\_braced\_quotes** со значением **TRUE**.

Пример:

$this->load->library('typography');  
$this->typography->protect\_braced\_quotes = TRUE;

# Класс Unit Testing

Класс Unit testing (класс тестирования) позволяет программистам тестировать написанный код. Если Вы не сильны в этом, то можете [поискать](http://google.com/) справку по данному вопросу.

Этот класс является простым и состоит всего из нескольких функций. Это есть не полноценная система тестирования и отладки скриптов, а всего лишь простой механизм проверки скриптов на возвращаемые значения и типы данных.

## Инициализация класса

Как и многие другие классы в CodeIgniter, класс инициализируется функцией**$this->load->library**:

$this->load->library('unit\_test');

После загрузки объект Unit Test будет доступен по обращению **$this->unit**

## Запуск теста

При запуске теста производится сравнение результатов :

## $this->unit->run( test, expected result, 'test name' );

Где **test** это значение для теста , **expected result** - ожидаемый тип данных,**test name** необязательное название теста. Например:

$test = 1 + 1;  
$expected\_result = 2;  
$test\_name = 'Один плюс один';  
$this->unit->run($test, $expected\_result, $test\_name);

Ожидаемый результат и значение для теста могут сравниваться по значениям или по типам данных. Например, сравнение по значению :

$this->unit->run('Foo', 'Foo');

А это сравнение с типом данных :

$this->unit->run('Foo', 'is\_string');

Второй параметр "is\_string" говорит функции, что результат ожидается строкового типа. Список доступных для тестирования типов данных:

* is\_string
* is\_bool
* is\_true
* is\_false
* is\_int
* is\_numeric
* is\_float
* is\_double
* is\_array
* is\_null

## Создание отчета

Вы можете отображать результат после каждого теста, или вывести отчет о выполнении всех тестов. Результат теста можно выводить и обычной функцией echo:

echo $this->unit->run($test, $expected\_result);

Для запуска полного отчета о все тестах используется код

echo $this->unit->report();

Отчет будет сформирован в виде таблицы для наглядности. Если нужны результаты в виде массива, то следует использовать функцию result():

echo $this->unit->result();

## Строгий режим

По умолчанию используется простой режим сравнения по значениям. Рассмотрим пример :

$this->unit->run(1, TRUE);

Тест сравнивает значение типа integer с boolean . PHP автоматически приводит их к одному типу данных, а ведь 1 и TRUE являются равными сущностями:

if (1 == TRUE) echo 'Считается как true';

Если необходимо строгое сравнение, без приведения к одному типу, как тут:

if (1 === TRUE) echo 'Считается как FALSE';

То следует включить строгий режим следующим кодом:

$this->unit->use\_strict(TRUE);

## Включение/Отключение тестирования

Если Вы хотите оставить код тестирования в своих скриптах, но не желаете, что бы проходило тестирование, то можете его отключить:

$this->unit->active(FALSE)

## Создание шаблона

Если Вы хотите оформить отчет по-своему, то можете использовать шаблоны для вывода. Примечание: в примере используется псевдокод

$str = '  
<table border="0" cellpadding="4" cellspacing="1">  
    **{rows}**  
        <tr>  
        <td>**{item}**</td>  
        <td>**{result}**</td>  
        </tr>  
    **{/rows}**  
</table>';  
**$this->unit->set\_template($str);**

**Важно:** Шаблон должен быть объявлен в коде **до** начала тестирования

# Класс URI

Класс URI предоставляет функции помогающие извлекать информацию из строки URI. Если используется URI роутинг, то также извлекается информация о перенаправленных сегментах.

**Примечание:** Этот класс инициализируется системой автоматически, поэтому нет необходимости делать это вручную.

## $this->uri->segment(n)

Позволяет извлекать определенные сегменты, где **n** — это номер извлекаемого сегмента. Сегменты нумеруются слева на право. Например, если полный URL такой:

http://www.your-site.com/index.php/news/local/metro/crime\_is\_up

, то номера сегментов будут следующие:

1. news
2. local
3. metro
4. crime\_is\_up

По-умолчанию функция возвращает FALSE (boolean), если сегменты отсутствуют. Опциональный второй параметр позволяет задать собственное значение по-умолчанию для отсутствующего сегмента. Например, следующим образом можно указать функции возвращать ноль в случае ошибки:

$product\_id = $this->uri->segment(3, 0);

Это помогает избежать написания подобного кода:

if ($this->uri->segment(3) === FALSE)  
{  
    $product\_id = 0;  
}  
else  
{  
    $product\_id = $this->uri->segment(3);  
}

## $this->uri->rsegment(n)

Эта функция идентична предыдущей за тем исключением, что она позволяет извлечь определенный сегмент из перенаправленного сегмента URI, в случае использования функционала [URI роутинга](http://code-igniter.ru/user_guide/general/routing.html) CodeIgniter'а.

## $this->uri->slash\_segment(n)

Эта функция также идентична **$this->uri->segment()** за тем исключением, что она позволяет с помощью второго параметра добавить к результату начальный и/или конечный слэш. Если параметр не задан, то добавляется конечный слэш. Примеры::

$this->uri->slash\_segment(**3**);  
$this->uri->slash\_segment(**3**, 'leading');  
$this->uri->slash\_segment(**3**, 'both');

Возвращает:

1. segment/
2. /segment
3. /segment/

## $this->uri->slash\_rsegment(n)

Эта функция идентична предыдущей, за тем исключением, что она позволяет добавить слэши к определенному сегменту в перенаправленном URI, в случае использования функционала [URI роутинга](http://code-igniter.ru/user_guide/general/routing.html) CodeIgniter'а.

## $this->uri->uri\_to\_assoc(n)

Эта функция позволяет преобразовать сегменты URI в ассоциативный массив пар ключ/значение. Рассмотрим следующий URI:

index.php/user/search/name/joe/location/UK/gender/male

Используя эту функцию, можно преобразовать URI в ассоциативный массив следующего вида:

[array]  
(  
    'name' => 'joe'  
    'location' => 'UK'  
    'gender' => 'male'  
)

Первый параметр функции позволяет задать смещение. С тех пор как первый и второй сегмент URI содержат имя контроллера и функции, значение смещения по-умолчанию устанавливается в **3**. Пример:

$array = $this->uri->uri\_to\_assoc(3);  
echo $array['name'];

Второй параметр позволяет задать название ключа по-умолчанию, таким образом возвращаемый функцией масив всегда будут содержать ожидаемые индексы даже, если сегменты URI отсутствуют. Пример:

$default = array('name', 'gender', 'location', 'type', 'sort');  
$array = $this->uri->uri\_to\_assoc(3, $default);

Если URI не содержит значение по-умолчанию, индексу массива все равно будет присвоенно это название, а его значение установленно в FALSE.

Наконец, если для данного ключа не найденно соответствующее значение (если заданно нечетное количество сегментов URI), то значение будет установенно в FALSE (boolean).

## $this->uri->ruri\_to\_assoc(n)

Эта функция идентична предыдущей за тем исключением, что она создает ассоциативный массив используя перенаправленный URI, в случае использования функционала [URI роутинга](http://code-igniter.ru/user_guide/general/routing.html) CodeIgniter'а.

## $this->uri->assoc\_to\_uri()

Принимает на входе ассоциативный массив и генерирует из него строку URI. Ключи массива будут включенны в строку. Пример:

$array = array('product' => 'shoes', 'size' => 'large', 'color' => 'red');  
$str = $this->uri->assoc\_to\_uri($array);  
// Генерирует: product/shoes/size/large/color/red

## $this->uri->uri\_string()

Возвращает строку с полным URI. Например, если полный URL имеет следующий вид:

http://www.your-site.com/index.php/news/local/345

, то функция вернет:

/news/local/345

## $this->uri->ruri\_string(n)

Эта функция идентична предыдущей за тем исключением, что она возвращает перенаправленный URI, в случае использования функционала [URI роутинга](http://code-igniter.ru/user_guide/general/routing.html)CodeIgniter'а.

## $this->uri->total\_segments()

Возвращает общее число сегментов.

## $this->uri->total\_rsegments()

Эта функция идентична предыдущей за тем исключением, что она возвращает общее число сегментов в перенаправленном URI, в случае использования функционала [URI роутинга](http://code-igniter.ru/user_guide/general/routing.html) CodeIgniter'а.

## $this->uri->segment\_array()

Возвращает массив содержащий сегменты URI. Например:

$segs = $this->uri->segment\_array();  
foreach ($segs as $segment)  
{  
    echo $segment;  
    echo '<br />';  
}

## $this->uri->rsegment\_array(n)

Эта функция идентична предыдущей за тем иключением, что она возвращает массив сегменов в перенаправленном URI, в случае использования функционала [URI роутинга](http://code-igniter.ru/user_guide/general/routing.html) CodeIgniter'а.

# Класс User Agent

Этот класс предоставляет функции, которые помогут выяснить версию браузера, мобильного устройства или робота, который посетил сайт. Также вы можете получить информацию об адресе с которого произошел переход и поддерживаемый пользовательским устройством язык.

## Инициализация класса

Как и для загрузки большинства прочих классов в CodeIgniter, вам необходимо использовать функцию **$this->load->library**:

$this->load->library('user\_agent');

После единоразовой загрузки, к классу можно обращаться как: **$this->agent**

## Определение браузера пользователя

Функции определения браузера пользователя располагаются в файле:**application/config/user\_agents.php**. В случае необходимости вы можете расширять список возможных браузеров.

## Пример

Когда инициализируется этот класс, то первым делом он пытается определить, кто сейчас просматривает текущую страницу: браузер пользователя или робот. Он так же определит версию операционной системы, если таковая информация будет передана.

$this->load->library('user\_agent');  
if ($this->agent->is\_browser())  
{  
    $agent = $this->agent->browser().' '.$this->agent->version();  
}  
elseif ($this->agent->is\_robot())  
{  
    $agent = $this->agent->robot();  
}  
elseif ($this->agent->is\_mobile())  
{  
    $agent = $this->agent->mobile();  
}  
else  
{  
    $agent = 'Unidentified User Agent';  
}  
echo $agent;  
echo $this->agent->platform(); // Platform info (Windows, Linux, Mac, etc.)

# Функции класса

## $this->agent->is\_browser()

Вернет TRUE/FALSE (boolean) если запрос был произведен из веб-браузера.

## $this->agent->is\_mobile()

Вернет TRUE/FALSE (boolean) если запрос был произведен от мобильного устройства.

## $this->agent->is\_robot()

Вернет TRUE/FALSE (boolean) если пользователь будет опознан как робот.

**Примечание:**  Библиотека, которой пользуется этот класс, содержит только наиболее распространенные заголовки, по которым определяется робот. Это далеко не полный список. Если вы обнаружите, что какой-то робот посещает ваш сайт и не определяется методами этого класса, вы можете добавить его в файл **application/config/user\_agents.php**.

## $this->agent->is\_referral()

Вернет TRUE/FALSE (boolean) если пользователь пришел к вам с другого сайта.

## $this->agent->browser()

Вернет строку, содержащую имя веб-браузера, из которого произошло обращение.

## $this->agent->version()

Вернет строку, содержащую версию браузера, из которого произошло обращение.

## $this->agent->mobile()

Вернет строку, содержащую имя мобильного устройства, из которого произошло обращение.

## $this->agent->robot()

Вернет строку, содержащую имя робота, который обратился к текущей странице.

## $this->agent->platform()

Вернет строку, содержащую описание операционной системы пользователя (Linux, Windows, OS X и т.п.).

## $this->agent->referrer()

Вернет адрес сайта, с которого пришел пользователь. Обычно вы будете использовать эту функцию так:

if ($this->agent->is\_referral())  
{  
    echo $this->agent->referrer();  
}

## $this->agent->agent\_string()

Вернет строку с полным описанием «User Agent» пользователя. Как правило она будет представлять из себя что-то вроде этого:

Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.0.4) Gecko/20060613 Camino/1.0.2

## $this->agent->accept\_lang()

Позволяет определить принимает ли браузер пользователя конкретный язык:

if ($this->agent->accept\_lang('ru'))  
{  
    echo 'Вы можете читать на русском!';  
}

**Примечание:** Эта функция, как правило, не является надежной, т.к. не все браузеры посылают информацию об используемом языке. Даже когда информация о языке передана, она может быть недостоверной.

## $this->agent->accept\_charset()

Позволяет определить, принимает ли браузер пользователя символы в конкретной кодировке:

if ($this->agent->accept\_charset('utf-8'))  
{  
    echo 'Ваш браузер поддерживает UTF-8!';  
}

**Примечание:** Эта функция, как правило, не является надежной, т.к. не все браузеры посылают информацию об используемой кодировке. Даже когда информация о кодировке передана, она может быть недостоверной.

# Проверка данных web-форм

Вот примерный сценарий взаимодействия с пользователем с помощью форм :

1. Показ форм .
2. Заполнение ее и отправка пользователем
3. Если заполнены не все необходимые поля или данные имеют не тот формат - показываем сообщение об ошибке и возвращаем данные пользователю для последующей правки.
4. Этот процесс продолжается, пока сервер не получит верные данные

После приема формы следует :

1. Проверить, заполнены ли все поля .
2. Убедится, что данные нужного формата и не содержат посторонних символов . (Например, имя пользователя может содержать только буквы. Если уже с таким именем есть пользователь, следует вывести об этом сообщение. И т.д. )
3. Фильтрация данных для безопасности (защита от XSS/SQL-inj и пр.)
4. Переформатирование данных (Напр., данные имеют разметку).
5. Подготовить данные к добавлению в БД.

Хотя приведенный алгоритм прост, это требует значительного количества кода и логики, что бы предусмотреть все случаи. Если создавать систему валидации "с нуля", то это очень утомительная и грязная работа.

**CodeIgniter представляет фреймворк, который способен сократить объемы готового кода. Он также имеет готовое решение для валидации форм.**

## Обзор системы

Для проверки данных с помощью CodeIgniter'а нужны три вещи:

1. Файл [отображения](http://code-igniter.ru/user_guide/general/views.html) с формой .
2. Файл должен содержать сообщение об успешной проверке .
3. А так же [контроллер](http://code-igniter.ru/user_guide/general/controllers.html) для получения и обработки данных.

Рассмотрим пример на основе создания формы входа .

## Форма

Создадим файл **myform.php** и разместим его в папке **applications/views/**:

<html>

<head>

<title>Моя Форма</title>

</head>

<body>

<?=$this->validation->error\_string; ?>

<?=form\_open('form'); ?>

<h5>Логин</h5>

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

<h5>Пароль</h5>

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

<h5>Подтверждение пароля</h5>

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

<h5>Email</h5>

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

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

</form>

</body>

</html>

## Сообщение об успешной проверке

Теперь создадим файл **formsuccess.php** и так же сохраним в папке**applications/views/**:

<html>

<head>

<title>Моя Форма</title>

</head>

<body>

<h3>Ваша форма была успешно отправлена!</h3>

<p><?=anchor('form', 'Еще раз!'); ?></p>

</body>

</html>

## Контроллер

В папке **applications/controllers/** создадим контроллер **form.php**.

<?php

class Form extends Controller {

function index()

{

$this->load->helper(array('form', 'url'));

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

if ($this->validation->run() == FALSE)

{

$this->load->view('myform');

}

else

{

$this->load->view('formsuccess');

}

}

}

?>

## Пробуем!

Теперь для проверки примера следует ввести примерно такой адрес :

www.your-site.com/index.php/**form**/

**Если Вы отправили форму, то увидите только ее обновление. Это произойдет из-за того, что мы не указали ни одного правила проверки данных.**

## Объяснение

Вы могли заметить некоторые необычные детали в приведенном коде:

**Форма** (myform.php) стандартная веб-форма, содержащая некоторые изменения:

1. Для "открытия" формы используется **хелпер form** . В принципе, это не обязательно - можно сделать это же и средствами HTML. Но тогда следует знать точный адрес к контроллеру, что может сказаться в следующих версиях скрипта лишними проблемами.
2. Вверху формы было добавлено следующее выражение:

<?=$this->validation->error\_string; ?>

Этот код показывает текст ошибки при возврате формы пользователю. Если ошибок не было, поле останется пустым.

**Контроллер**(form.php) имеет одну функцию: **index()**. Она инициализирует контроллер, загружает **хелпер form** and **URL хелпер**, а так же **запускает**процесс валидации. На основе результатов проверки будет отображена страница успешного выполнения или сообщение об ошибке.

**Пока мы не указываем никаких условий, функция возвращает "false" (boolean false). Функция run() возвращает "true" если правила проверки успешно применены.**

## Установка правил проверки данных

CodeIgniter позволяет устанавливать неограниченное число правил проверки. Рассмотрим пример.

В **контроллере** (form.php) добавьте следующий код после функции инициализации:

$rules['username'] = "required";  
$rules['password'] = "required";  
$rules['passconf'] = "required";  
$rules['email'] = "required";  
$this->validation->set\_rules($rules);

Теперь контроллер будет иметь вид:

<?php

class Form extends Controller {

function index()

{

$this->load->helper(array('form', 'url'));

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

$rules['username'] = "required";

$rules['password'] = "required";

$rules['passconf'] = "required";

$rules['email'] = "required";

$this->validation->set\_rules($rules);

if ($this->validation->run() == FALSE)

{

$this->load->view('myform');

}

else

{

$this->load->view('formsuccess');

}

}

}

?>

**Теперь отправьте форму с пустыми полями - появится сообщение об ошибке.**

**Примечание:** на данный момент мы не возвращали поля формы при ошибке. Это будет рассмотрено далее.

## Изменение оформления ошибки

По умолчанию ошибки заключаются в тег <p>. Можно изменить способ оформления, воспользовавшись этим кодом :

$this->validation->set\_error\_delimiters('**<div class="error">**', '**</div>**');

В нашем примере, текст ошибки будет заключен в тег <div>

## Расширение правил

CodeIgniter позволяет объединять условия проверки. Например:

$rules['username'] = "required|min\_length[5]|max\_length[12]";  
$rules['password'] = "required|matches[passconf]";  
$rules['passconf'] = "required";  
$rules['email'] = "required|valid\_email";

Эти правила указывают :

1. Длина поля username должна быть от 5 до 12 символов.
2. Пароль должен совпадать в обоих полях .
3. Поле email должно содержать корректный адрес.

**Примечание:** Нумерованные правила так же доступны для использования.

## Подготовка данных

В дополнение к проверки данные можно подготовить определенным образом и отфильтровать :

$rules['username'] = "**trim**|required|min\_length[5]|max\_length[12]|**xss\_clean**";  
$rules['password'] = "**trim**|required|matches[passconf]|**md5**";  
$rules['passconf'] = "**trim**|required";  
$rules['email'] = "**trim**|required|valid\_email";

В этом коде мы обрезаем пробелы на концах строк, переводим пароль в MD5-хеш, и запускаем функцию фильтрации "xss\_clean", удаляющую вредоносный код .

**Любая функция PHP должна иметь только один параметр, для использования при проверке. Напр., htmlspecialchars, trim, MD5.**

**Примечание:** Остальные способы обработки следует использовать **после**проверки соответствия правилам, так как в случае ошибки мы решили возвращать данные пользователю.

## Callbacks: создание собственных правил

Система валидации поддерживает использование Ваших функций проверки данных. Это позволяет расширить способы обработки. Например, можно создать функцию проверки логина на уникальность - нет ли пользователя с таким же логином в БД? Рассмотрим создание своих правил на примере.

В контроллере измените правило для поля "username" на следующее:

$rules['username'] = "callback\_username\_check";

Добавим новую функцию **username\_check** в ваш контроллер. Теперь он будет иметь следующий вид:

<?php

class Form extends Controller {

function index()

{

$this->load->helper(array('form', 'url'));

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

$rules['username'] = "callback\_username\_check";

$rules['password'] = "required";

$rules['passconf'] = "required";

$rules['email'] = "required";

$this->validation->set\_rules($rules);

if ($this->validation->run() == FALSE)

{

$this->load->view('myform');

}

else

{

$this->load->view('formsuccess');

}

}

function username\_check($str)

{

if ($str == 'test')

{

$this->validation->set\_message('username\_check', 'The %s field can not be the word "test"');

return FALSE;

}

else

{

return TRUE;

}

}

}

?>

Обновите форму и введите в поле "Логин" слово "test". Вы увидите сообщение об ошибке, сгенерированное нашей функцией.

**Для использования своих функций следует применять префикс "callback\_" в правилах .**

Описание ошибки задается функцией **$this->validation->set\_message**. Для этого следует указать название функции первым параметром, а текст - вторым.

**Важно:** Вы так же можете изменять сообщения других функций. Напр, для правила "required" сделаем следующее:

$this->validation->set\_message('required', 'Ваше сообщение об ошибке');

## Перезаполнение формы

Если произошла ошибка при проверке, то следует перезаполнить поля формы. Для этого вставьте данный код после правил:

$fields['username'] = 'Username';  
$fields['password'] = 'Password';  
$fields['passconf'] = 'Password Confirmation';  
$fields['email'] = 'Email Address';  
$this->validation->set\_fields($fields);

Где ключи массива - реальные названия полей.

Функция index() теперь должна выглядеть так:

function index()

{

$this->load->helper(array('form', 'url'));

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

$rules['username'] = "required";

$rules['password'] = "required";

$rules['passconf'] = "required";

$rules['email'] = "required";

$this->validation->set\_rules($rules);

$fields['username'] = 'Username';

$fields['password'] = 'Password';

$fields['passconf'] = 'Password Confirmation';

$fields['email'] = 'Email Address';

$this->validation->set\_fields($fields);

if ($this->validation->run() == FALSE)

{

$this->load->view('myform');

}

else

{

$this->load->view('formsuccess');

}

}

Откройте **myform.php** вьювер и вставьте этот код:

<html>

<head>

<title>My Form</title>

</head>

<body>

<?=$this->validation->error\_string; ?>

<?=form\_open('form'); ?>

<h5>Username</h5>

<input type="text" name="username" value="<?=$this->validation->username;?>" size="50" />

<h5>Password</h5>

<input type="text" name="password" value="<?=$this->validation->password;?>" size="50" />

<h5>Password Confirm</h5>

<input type="text" name="passconf" value="<?=$this->validation->passconf;?>" size="50" />

<h5>Email Address</h5>

<input type="text" name="email" value="<?=$this->validation->email;?>" size="50" />

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

</form>

</body>

</html>

Теперь обновите форму и введите заранее неверные данные. В описании ошибки увидите названия форм.

**Раздельный показ ошибок**

Если Вы желаете выводить ошибки около поля, где она произошла, то измените форму таким образом:

<h5>Username</h5>

<?=$this->validation->username\_error; ?>

<input type="text" name="username" value="<?=$this->validation->username;?>" size="50" />

<h5>Password</h5>

<?=$this->validation->password\_error; ?>

<input type="text" name="password" value="<?=$this->validation->password;?>" size="50" />

<h5>Password Confirm</h5>

<?=$this->validation->passconf\_error; ?>

<input type="text" name="passconf" value="<?=$this->validation->passconf;?>" size="50" />

<h5>Email Address</h5>

<?=$this->validation->email\_error; ?>

<input type="text" name="email" value="<?=$this->validation->email;?>" size="50" />

Если не произойдет ошибки, то ничего не появится. Иначе у поля с ошибкой будет ее описание ( по умолчанию в тегах <p>).

**Примечание:**Текст ошибки будет доступен по вызову с постфиксом "\_error". Например, ошибка в поле "username" отображается так:  
**$this->validation->username\_error**.

**Описание правил**

Следующая таблица содержит описания стандартных правил для проверки :

|  |  |  |  |
| --- | --- | --- | --- |
| **Правило** | **Параметр** | **Описание** | **Пример** |
| **required** | No | Возвращает FALSE если поле не заполнено |  |
| **matches** | Yes | Возвращает FALSE если элемент не соответствует значению другого элемента | matches[form\_item] |
| **min\_length** | Yes | Возвращает FALSE если значение короче указанного | min\_length[6] |
| **max\_length** | Yes | Возвращает FALSE если длина больше указанной | max\_length[12] |
| **exact\_length** | Yes | Возвращает FALSE если длина не равна заданной | exact\_length[8] |
| **alpha** | No | Возвращает FALSE если элемент содержит не только буквы |  |
| **alpha\_numeric** | No | Возвращает FALSE если элемент содержит не только буквы и цифры |  |
| **alpha\_dash** | No | Возвращает FALSE если элемент содержит не только буквы и знаки препинания. |  |
| **numeric** | No | Возвращает FALSE если значение не числового вида |  |
| **integer** | No | Возвращает FALSE если значение не является целым числом |  |
| **valid\_email** | No | Возвращает FALSE если значения не является корректным e-mail адресом |  |
| **valid\_ip** | No | Возвращает FALSE если IP-адрес не является действительным. |  |
| **valid\_base64** | No | Возвращает FALSE если строка не является base-64 шифром |  |

**Примечание:** Эти правила могут вызываться как функции. Например:

$this->validation->required($string);

**Примечание:** Можно использовать стандартные PHP функции, требующие один параметр.

**Описание фильтров**

Доступны по умолчанию следующие фильтры:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Параметр** | **Описание** |
| **xss\_clean** | No | Проводит фильтр на XSS-уязвимость, используемый в классе [Input](http://code-igniter.ru/user_guide/libraries/input.html) |
| **prep\_for\_form** | No | Конвертирует HTML-элементы в их альтернативное значение для отображения |
| **prep\_url** | No | Добавляет "http://" к URL при надобности |
| **strip\_image\_tags** | No | Извлекает URL из ссылки на картинку |
| **encode\_php\_tags** | No | Конвертирует теги php в текст |

**Примечание:** Можно использовать стандартные PHP функции, требующие один параметр, напр. **trim**, **htmlspecialchars**, **urldecode**

**Установка своих сообщений об ошибках**

Все стандартные сообщения расположены по адресу: **language/english/validation\_lang.php**

Для установки своего описания к правилу используйте следующую функцию

$this->validation->set\_message(**'rule'**, **'Error Message'**);

Где **rule** это имя правила, а **Error Message** - описание ошибки.

**Работа с выпадающими списками, одиночными и мульти маркерами**

Класс Validation имеет три функции, позволяющие работать с этими элементами:

**set\_select()**

Позволяет выбирать значение из списка. Параметры: имя меню и значение. Например:

<select name="myselect">  
<option value="one" **<?= $this->validation->set\_select('myselect', 'one'); ?>**>One</option>  
<option value="two" **<?= $this->validation->set\_select('myselect', 'two'); ?>**>Two</option>  
<option value="three" **<?= $this->validation->set\_select('myselect', 'three'); ?>**>Three</option>  
</select>

**set\_checkbox()**

Позволяет устанавливать значение чекбокса (маркера) Параметры: имя маркера и значение. Например:

<input type="checkbox" name="mycheck" value="1" **<?= $this->validation->set\_checkbox('mycheck', '1'); ?>** />

**set\_radio()**

Позволяет устанавливать значение радиобокса (мультимаркера) Параметры: имя маркера и значение. Например:

<input type="radio" name="myradio" value="1" **<?= $this->validation->set\_radio('myradio', '1'); ?>**/>

Классы XML-RPC и XML-RPC Сервер

Класс XML-RPC позволяет отправлять запросы на другие сервера или сделать собственный XML-RPC сервер для получения запросов.

**Что такое XML-RPC?**

Это технология, позволяющая устанавливать связь между двумя компьютерами с помощью языка XML. Один компьютер, называемый **клиент**, отправляет XML-RPC **запросы** на другой компьютер, называемый **сервер**. После этого сервер обрабатывает запрос и отправляет **ответ** клиенту.

К примеру, используя MetaWeblog API, XML-RPC клиент (обычно это пользовательское приложение) отправляет данные XML-RPC серверу, расположенному в Интернете. Запросом является обычно или новый пост блога, или отредактированная запись. Когда сервер XML-RPC получает данные, они обрабатываются, а пользователю - клиенту - отправляется ответ - результат обработки запроса

Для более полной спецификации посетите сайт [XML-RPC](http://www.xmlrpc.com/).

**Инициализация Класса**

Как и многие классы, XML-RPC и XML-RPCS инициализируются с использованием функции **$this->load->library**:

Для загрузки XML-RPC клиента используется этот код:

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

А для загрузки XML-RPC сервера - этот:

$this->load->library('xmlrpc');  
$this->load->library('xmlrpcs');

Объект будет доступен по вызову **$this->xmlrpc** (клиент) и **$this->xmlrpcs**(сервер)

**Важно:**  При использовании XML-RPC сервера также необходимо загрузить и класс XML-RPC клиента.

**Отправка XML-RPC запросов**

Для отправки запросов XML-RPC серверу следует указать следующие данные:

* URL сервера
* Метод, вызываемый на сервере
* Собственно сами *данные*.

Вот пример простой отправки данных на Weblogs.com с использованием [Ping-o-Matic](http://pingomatic.com/)

$this->load->library('xmlrpc');  
$this->xmlrpc->server('http://rpc.pingomatic.com/', 80);  
$this->xmlrpc->method('weblogUpdates.ping');  
$request = array('Мой фотоблог', 'http://www.my-site.com/photoblog/');  
$this->xmlrpc->request($request);  
if ( ! $this->xmlrpc->send\_request())  
    echo $this->xmlrpc->display\_error();

**Объяснение**

Приведенный код запускает XML-RPC класс, устанавливает URL сервера и метод (тут - weblogUpdates.ping). Запрос (в этом случае URL и заголовок сайта) размещается в массиве и подготавливается к передаче функцией request() . Затем готовый запрос посылается с помощью **send\_request()**, и если произошла ошибка, то отображается ее текст, полученный с XML-RPC сервера.

**Структура запроса**

XML-RPC **запрос** это данные, посылаемые на XML-RPC сервер. Каждая часть данных является **параметром запроса**. Приведенный выше пример имел два параметра: URL и заголовок сайта. Когда XML-RPC получает запрос, он извлекает из него переданные параметры.

Все параметры должны находится в одном массиве, и каждый из параметров должен соответствовать определенному типу данных (строка, число и пр.).

Пример массива с тремя переменными :

$request = array('John', 'Doe', 'www.some-site.com');  
$this->xmlrpc->request($request);

Если типы данных разные, то следует поместить каждый параметр в субмассив и вторым индексом указывать их тип, например:

$request = array (  
                   array('John', 'string'),  
                   array('Doe', 'string'),  
                   array(FALSE, 'boolean'),  
                   array(12345, 'int')  
                 );   
$this->xmlrpc->request($request);

Раздел [Типы данных](http://code-igniter.ru/user_guide/libraries/xmlrpc.html#datatypes) содержит полный список доступных типов.

**Создание XML-RPC сервера**

XML-RPC сервер включает в себя функции получения и обработки данных в соответствии с заданными методами.

Что бы создать XML-RPC сервер следует инициализировать класс XML-RPC сервера в контроллере, указать, что посылать в ответе клиенту и какие методы будут обрабатывать данные.

Пример, описывающий это:

$this->load->library('xmlrpc');  
$this->load->library('xmlrpcs');  
$config['functions']['new\_post']  = array('function' => 'My\_blog.new\_entry');  
$config['functions']['update\_post'] = array('function' => 'My\_blog.update\_entry');  
$this->xmlrpcs->initialize($config);  
$this->xmlrpcs->serve();

Приведенный пример загружает сервер и указывает, какими методами будут обрабатываться данные. Методы указываются в левой части выражения, а обрабатывающие их функции - в правой.

К примеру, XML-RPC отправляет запрос с использованием метода **new\_post**, а сервер загружает класс **My\_blog** и вызывает функцию **new\_entry** Аналогично метод **update\_post** указывает серверу использовать функцию **update\_entry**класса **My\_blog**.

Имена функций и методов приведены к примеру. Вы можете сами решить, разрабатывать новый способ обработки, или использовать уже готовые стандарты, такие как Blogger или MetaWeblog API.

**Обработка запросов**

Когда XML-RPC сервер получил данные и вызвал соответствующую методу функцию, следует обработать полученные данные и отправить ответ.

Продолжаем на основе приведенных выше примеров. Получив метод**new\_post** , сервер загружает класс, оформленный подобным образом:

class My\_blog extends Controller {  
    function new\_post($request)  
    {  
    }  
}

Переменная **$request** является объектом с данными, принятыми от XML-RPC клиента. Используя этот объект, можно получить доступ к *параметрам запроса* . Когда запрос обработан, следует отправить **ответ** клиенту.

Рассмотрим реальный пример с использованием Blogger API. Один из методов Blogger API - **getUserInfo()**. Используя его, XML-RPC клиент отправляет серверу логин и пароль, и сервер возвращает данные об этом пользователе ( ник, ID, email адрес и пр.). Вот реализация этого способа:

class My\_blog extends Controller {  
    function getUserInfo($request)  
    {  
        $username = 'smitty';  
        $password = 'secretsmittypass';  
        $this->load->library('xmlrpc');  
        $parameters = $request->output\_parameters();  
        if ($parameters['1'] != $username AND $parameters['2'] != $password)  
        {  
            return $this->xmlrpc->send\_error\_message('100', 'Invalid Access');  
        }  
        $response = array(array('nickname'  => array('Smitty','string'),  
                                'userid'    => array('99','string'),  
                                'url'       => array('http://yoursite.com','string'),  
                                'email'     => array('jsmith@yoursite.com','string'),  
                                'lastname'  => array('Smith','string'),  
                                'firstname' => array('John','string')  
                                ),  
                         'struct');  
        return $this->xmlrpc->send\_response($response);  
    }  
}

**Примечания:**

Функция **output\_parameters()** возвращает индексированный массив с данными, полученными от клиента. В нашем примере это username (логин) и password (пароль).

Если логин и/или пароль не верные, сообщение об ошибке отправляется с помощью **send\_error\_message()**.

Если все прошло успешно, то клиент в ответе получит массив с данными о пользователе

**Формат ответа**

Как и *запрос*, *ответ* должен быть оформлен в виде массива.. Но ответный массив должен содержать **только одно значение**. Это значение может быть массивом с субмассивами, но должно иметь только один индекс.. Иначе говоря, структура ответа должна быть такой:

$response = array('Response data', 'array');

Вот пример структуры готового ответа:

$response = array (  
                   array(  
                         'first\_name' => array('John', 'string'),  
                         'last\_name' => array('Doe', 'string'),  
                         'member\_id' => array(123435, 'int'),  
                         'todo\_list' => array(array('clean house', 'call mom', 'water plants'), 'array'),  
                        ),  
                 'struct'  
                 );

Заметьте, что ответ имеет тип **struct**. Это наиболее предпочтительный тип для отправки ответов.

Как и массиве запроса, в ответе могут содержаться данные определенных[типов](http://code-igniter.ru/user_guide/libraries/xmlrpc.html#datatypes) .

**Отправка сообщения об ошибке**

Для отправки сообщения от ошибке используется следующий код:

return $this->xmlrpc->send\_error\_message('123', 'Запрашиваемые данные недоступны');

Первый параметр - код ошибки, второй - ее описание.

**Создание собственного клиента и сервера**

Для полного понимания данного раздела давайте создадим собственные XML-RPC клиент и сервер. вы можете использовать данный клиент для отправки запросов, а сервер - для из получения, анализа и ответов.

**Клиент**

Создадим контроллер **xmlrpc\_client.php** и поместим его в директорию **applications/controllers/** :

<?php

class Xmlrpc\_client extends Controller {

function index()

{

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

$server\_url = site\_url('xmlrpc\_server');

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

$this->xmlrpc->server($server\_url, 80);

$this->xmlrpc->method('Greetings');

$request = array('How is it going?');

$this->xmlrpc->request($request);

if ( ! $this->xmlrpc->send\_request())

{

echo $this->xmlrpc->display\_error();

}

else

{

echo '<pre>';

print\_r($this->xmlrpc->display\_response());

echo '</pre>';

}

}

}

?>

Примечание: приведенный код использует "url helper". Информация о нем доступна на странице [Вспомогательных функций](http://code-igniter.ru/user_guide/general/helpers.html).

**Сервер**

Создадим контроллер с именем **xmlrpc\_server.php** и сохраним его в папке**applications/controllers/**:

<?php

class Xmlrpc\_server extends Controller {

function index()

{

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

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

$config['functions']['Greetings'] = array('function' => 'Xmlrpc\_server.process');

$this->xmlrpcs->initialize($config);

$this->xmlrpcs->serve();

}

function process($request)

{

$parameters = $request->output\_parameters();

$response = array(

array(

'you\_said' => $parameters['0'],

'i\_respond' => 'Not bad at all.'), 'struct');

return $this->xmlrpc->send\_response($response);

}

}

?>

**Пробуйте!**

Теперь можете проверить работу, перейдя по ссылке, подобной этой:

www.your-site.com/index.php/**xmlrpc\_client**/

Вы увидите отправленное сообщение серверу и полученный ответ.

Созданный клиент посылает сообщение ("How's is going?") серверу, использую метод "Greetings". Сервер обрабатывает его в функции "process" , где и формируется ответ.

Описание функций XML-RPC

**$this->xmlrpc->server()**

Устанавливает адрес и порт сервера для передачи запроса:

$this->xmlrpc->server('http://www.sometimes.com/pings.php', 80);

**$this->xmlrpc->timeout()**

Задает тайм-аут (в секундах) после которого запрос отменяется:

$this->xmlrpc->timeout(6);

**$this->xmlrpc->method()**

Задает метод, которым будет обрабатывать запрос XML-RPC сервер:

$this->xmlrpc->method('method');

Где **method** имя метода.

**$this->xmlrpc->request()**

Получает массив данных и создает запрос для отправки XML-RPC серверу:

$request = array(array('My Photoblog', 'string'), 'http://www.yoursite.com/photoblog/');  
$this->xmlrpc->request($request);

**$this->xmlrpc->send\_request()**

Отправка запроса. Возвращает TRUE или FALSE

**$this->xmlrpc->set\_debug(TRUE);**

Включение отладочного режима - отображение различной информации и сообщения об ошибках.

**$this->xmlrpc->display\_error()**

Возвращает описание и номер ошибки, если она произошла.

echo $this->xmlrpc->display\_error();

**$this->xmlrpc->display\_response()**

Возвращает ответ сервера в виде ассоциативного массива.

$this->xmlrpc->display\_response();

**$this->xmlrpc->send\_error\_message()**

Функция отправляет сообщение об ошибке клиенту. Параметры: номер ошибки и ее описание:

return $this->xmlrpc->send\_error\_message('123', 'Requested data not available');

**$this->xmlrpc->send\_response()**

Отправляет ответ сервера клиенту. Параметр: массив с данными.

$response = array(  
                 array(  
                        'flerror' => array(FALSE, 'boolean'),  
                        'message' => "Thanks for the ping!")  
                     )  
                 'struct');  
return $this->xmlrpc->send\_response($response);

**Типы данных**

В соответствии с [XML-RPC спецификацией](http://www.xmlrpc.com/spec) предопределены семь типов данных, используемых при передаче XML-RPC:

* *int* или *i4* - целые числа
* *boolean* - логическое
* *string* - строка
* *double* - большие дробные числа
* *dateTime.iso8601* - дата
* *base64* - base64-строка.
* *struct* (массив значений)
* *array* (массив значений)

Класс Zip Encoding

Класс ZIP позволяет создавать zip архивы. Архивы могут быть скачены на компьютер или сохранены в директорию

**Инициализация класса**

Как и многие другие классы в CodeIgniter, класс Zip инициализируется в вашем контроллере с использованием функции **$this->load->library**:

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

Однажды загруженный, объект ZIP будет доступен, с использованием: **$this->zip**

**Пример**

Этот пример показывает как сжать файл, сохранить его в папку на сервере и скачать его на компьютер.

$name = 'mydata1.txt';  
$data = 'A Data String!';  
$this->zip->add\_data($name, $data);  
// Запишет zip архив в папку на сервере. С именем"my\_backup.zip"  
$this->zip->archive('/path/to/directory/my\_backup.zip');   
// Загружает файл на компьютер. С именем "my\_backup.zip"  
$this->zip->download('my\_backup.zip');

Функции класса ZIP

**$this->zip->add\_data()**

Позволяет добавлять данные в Zip архив. Первый параметр должен содержать имя для файла, второй параметр данные с типом строка:

$name = 'my\_bio.txt';  
$data = 'Я был рожден в лифте...';  
$this->zip->add\_data($name, $data);

Можно многократно вызывать эту функцию, чтобы добавить несколько файлов в архив. Пример:

$name = 'mydata1.txt';  
$data = 'A Data String!';  
$this->zip->add\_data($name, $data);  
$name = 'mydata2.txt';  
$data = 'Another Data String!';  
$this->zip->add\_data($name, $data);

Или можно передать список файлов и данных в массиве:

$data = array(  
                'mydata1.txt' => 'A Data String!',  
                'mydata2.txt' => 'Another Data String!'  
            );  
$this->zip->add\_data($data);  
$this->zip->download('my\_backup.zip');

Если необходимо сжать данные и организовать их по папкам, включите путь в имя файла:

$name = 'personal/my\_bio.txt';  
$data = 'I was born in an elevator...';  
$this->zip->add\_data($name, $data);

В этом примере файл **my\_bio.txt** будет расположен в папке с именем **personal**.

**$this->zip->add\_dir()**

Позволяет добавлять директории. Обычно эта функция не используется, потому что возможно разместить данные по папкам используя функцию **$this->zip->add\_data()**, но если нужно создать пустую папку можно использовать её. Пример:

$this->zip->add\_dir('myfolder'); // Создаст папку с именем "myfolder"

**$this->zip->read\_file()**

Позволяет сжать файл, который находится на сервере. Укажите путь к файлу и класс Zip прочитает его и добавит в архив:

$path = '/path/to/photo.jpg';  
$this->zip->read\_file($path);   
// Загружает файл на компьютер. С именем "my\_backup.zip"  
$this->zip->download('my\_backup.zip');

Если необходимо сохранить в Zip архиве структуру папок, укажите **TRUE**(булева) во втором параметре. Пример:

$path = '/path/to/photo.jpg';  
$this->zip->read\_file($path, TRUE);   
// Загружает файл на компьютер. С именем "my\_backup.zip"  
$this->zip->download('my\_backup.zip');

В этом примере файл **photo.jpg** будет расположен в папке: **path/to/**

**$this->zip->read\_dir()**

Позволяет сжимать папки (и данные в них) которые расположены на сервере. Укажите путь к дериктории и класс ZIP рекурсивно прочитает и пересоздаст их как Zip архив. Все файлы и любые под-директории расположенные в пределах указанного пути будут закодированы(сжаты). Пример:

$path = '/path/to/your/directory/';  
$this->zip->read\_dir($path);   
// Загружает файл на компьютер. С именем "my\_backup.zip"  
$this->zip->download('my\_backup.zip');

**$this->zip->archive()**

Записывает Zip файл в директорию на сервере. Укажите правильный путь на сервере, заканчивающийся именем файла. У директории должны стоять права на запись(666 или 777 обычно). Пример:

$this->zip->archive('/path/to/folder/myarchive.zip'); // Creates a file named myarchive.zip

**$this->zip->download()**

Заставляет ZIP файл скачиваться с вашего сервера. В функцию необходимо передать имя zip файла.Пример:

$this->zip->download('latest\_stuff.zip'); // Файл будет с именем "latest\_stuff.zip"

**Примечание:** Не выводите никаких данных в контроллере в котором вызывается эта функция так как посылаются заголовки от сервера, которые заставляют скачиваться файл в двоичном формате.

**$this->zip->get\_zip()**

Возвращает сжатые Zip данные. Обычно эта функция используется только если необходимо что-то сделать с данными. Пример:

$name = 'my\_bio.txt';  
$data = 'I was born in an elevator...';  
$this->zip->add\_data($name, $data);  
$zip\_file = $this->zip->get\_zip();

**$this->zip->clear\_data()**

Zip класс кеширует zip данные, чтобы не пришлось пересоздавать Zip архив каждый раз когда вы используете какую-либо функцию из перечисленных выше. Если необходимо создать несколько архивов с разными данными можно очистить кеш перед вызовом функции. Пример:

$name = 'my\_bio.txt';  
$data = 'I was born in an elevator...';  
$this->zip->add\_data($name, $data);  
$zip\_file = $this->zip->get\_zip();  
$this->zip->clear\_data();   
$name = 'photo.jpg';  
$this->zip->read\_file("/path/to/photo.jpg"); //Читает содержимое файла  
$this->zip->download('myphotos.zip');

**Глава 4. Хелперы**

# Хелпер Array

Хелпер Array содержит функции, которые помогают работать с массивами.

## Загрузка хелпера

Для загрузки хелпера используется следующий код:

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

Доступны следующие функции:

## element()

Позволяет выбрать элемент массива. Функция проверяет, существует ли элемент в массиве и содержит ли он значение. Если значение существует, функция вернет его. Если значение не существует, будет возвращено FALSE или то, что вы указали третьим параметром в качестве значения по-умолчанию. Пример:

$array = array('color' => 'red', 'shape' => 'round', 'size' => '');  
// вернёт "red"  
echo element('color', $array);  
// вернёт NULL  
echo element('size', $array, NULL);

## random\_element()

Получает массив на вход и возвращает случайный его элемент. Пример использования:

$quotes = array(  
            "Я понял, что чем усердней я работаю, тем более я удачлив. - Томас Джефферсон",  
            "Не оставайся в кровати, если ты не можешь там делать деньги. - Джордж Бёрнс",  
            "Мы не проиграли игру; у нас просто вышло время. - Винс Ломбарди",  
            "Если всё под контролем, увеличьте темп. - Марио Андретти",  
            "Реальность — просто иллюзия, хотя и очень стойкая. - Альберт Эйнштейн",  
            "Шанс служит лишь подготовленному уму - Луис Пастер"  
            );  
echo random\_element($quotes);

# Хелпер Compatibility

Хелпер совместимости содержит реализацию на PHP 4 некоторых "родных" функций PHP 5 . Это может быть полезным, если Ваш код используется на сервере с PHP 4, но Вы желаете использовать в нем новые функции PHP5. Этот хелпер задан в [Автозагрузке](http://code-igniter.ru/user_guide/general/autoloader.html), так что не стоит его загружать вручную.

**Важно:** Это несколько функций, описанных в файле Compat.php. На свое усмотрение Вы можете не использовать данный хелпер, так как функции все равно будут доступны по умолчанию.

## Загрузка хелпера

Хелпер загружается следующим кодом :

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

## Доступные константы

На данный момент доступна константа

### PHP\_EOL

Она содержит в себе знак перехода на новую строку, например на сервере Windows - "\r\n", на \*nix - "\n".

## Доступные функции

Доступны данные функции (Подробнее про них читайте в документации по PHP)

### [file\_put\_contents()](http://us.php.net/manual/en/function.file-put-contents.php) - Четвертый параметр $context не поддерживается.

### [fputcsv()](http://us.php.net/manual/en/function.fputcsv.php)

### [http\_build\_query()](http://us.php.net/manual/en/function.http-build-query.php)

### [str\_ireplace()](http://us.php.net/manual/en/function.str-ireplace.php) - Четвертый параметр $count не поддерживается, PHP4 создает его самостоятельно..

### [stripos()](http://us.php.net/manual/en/function.stripos.php)

# Хелпер Cookie

Хелпер Cookie содержит функции, которые помогают работать с cookie.

## Загрузка хелпера

Для загрузки хелпера используется следующий код:

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

Доступны следующие функции:

## set\_cookie()

Записывает в cookie указанные значения. Есть два способа передать параметры данной функции: массивом и отдельными параметрами:

#### Передача массивом

В данном случае первым параметром передаётся ассоциативный массив:

$cookie = array(  
                   'name'   => 'Имя Cookie',  
                   'value'  => 'Значение',  
                   'expire' => '86500',  
                   'domain' => '.some-domain.com',  
                   'path'   => '/',  
                   'prefix' => 'myprefix\_',  
               );  
set\_cookie($cookie);

**Примечания:**

Обязательны только имя и значение.

Время жизни указывается **в секундах** относительно текущего времени. Не указывайте абсолютное значение. Используйте значение в секундах относительно текущего времени. Если указан 0 — cookie будет доступен до закрытия браузера.

Для удаления cookie укажите пустое значение.

Для использования cookie вне зависимости от адреса, по которому был загружен сайт, добавьте URL к **домену** начиная с точки, например как: .your-domain.com

Путь чаще всего не требуется т.к. в функции определен путь до корня.

Префикс требуется только если вам необходимо избежать коллизии имен с другими cookie со сходными именами того же сервера.

#### Передача отдельными параметрами

Если вы предпочитаете, вы можете установить cookie путем указания отдельных параметров:

set\_cookie($name, $value, $expire, $domain, $path, $prefix);

## get\_cookie()

Позволяет вам получить установленный cookie. Первый парметр задает имя cookie:

get\_cookie('some\_cookie');

Функция возвращает FALSE если элемент, который вы хотите получить не существует.

Второй необязательный параметр позволяет обработать данные фильтром XSS. Для его использования укажите вторым параметром TRUE;

get\_cookie('some\_cookie', TRUE);

## delete\_cookie()

Позволяет удалить cookie. Если при создании вы не указывали свой путь или другие параметры, требуется только имя cookie:

delete\_cookie("name");

Данная функция похожа на **set\_cookie()**, но не позволяет указать значение и время жизни. Вы можете передать массив значений первым параметром, либо использовать отдельные параметры.

delete\_cookie($name, $domain, $path, $prefix)

# Хелпер Date

Хелпер Date содержит функции, помогающие работать с датами.

## Загрузка хелпера

Для загрузки хелпера используется следующий код:

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

Доступны следующие функции:

## now()

Возвращает текущее время в формате Unix timestamp. В зависимости от настроек времени в конфигурационном файле берется либо локальное время сервера либо время по гринвичу(GMT). Если вы не хотите получать время по гринвичу (чаще всего это требуется если вы разрабатываете сайт, позволяющий пользователю выставить свою временную зону), нет никакой выгоды от использования данной функции по сравнению со стандартной для PHP функцией time().

## mdate()

Данная функция идентична функции PHP [date()](http://www.php.net/date) за исключенем того, что она позволяет вам использовать синтаксис шаблонов дат MySQL, где перед каждым кодовым символом стоит знак процента: %Y %m %d и т.д.

Преимущество создание дат таким образом в том, что вам не нужно заботится об экранировании символов, не являющихся кодовыми символами, что приходится делать при использовании date(). Пример:

$datestring = "Год: %Y Месяц: %m День: %d - %h:%i %a";  
$time = time();  
echo mdate($datestring, $time);

Если вторым параметром не указывается конкретное значение timestamp — используется текущее время.

## standard\_date()

Позволяет вам получить дату в одном из стандартных форматов. Пример:

$format = 'DATE\_RFC822';  
$time = time();  
echo standard\_date($format, $time);

Первый параметр должен задавать формат, второй — дату в виде Unix timestamp.

Поддерживаемые форматы:

* DATE\_ATOM
* DATE\_COOKIE
* DATE\_ISO8601
* DATE\_RFC822
* DATE\_RFC850
* DATE\_RFC1036
* DATE\_RFC1123
* DATE\_RFC2822
* DATE\_RSS
* DATE\_W3C

## local\_to\_gmt()

Получает в качестве аргумента время в формате Unix timestamp и возвращает время по гринвичу. Пример:

$now = time();  
$gmt = local\_to\_gmt($now);

## gmt\_to\_local()

Получает в качестве аргумента время в формате Unix timestamp (приведенное к GMT) и конвертирует его в локальное время в зависимости от часового пояса и летнего/зимнего времени. Пример:

$timestamp = '1140153693';  
$timezone = 'UM8';  
$daylight\_saving = TRUE;  
echo gmt\_to\_local($timestamp, $timezone, $daylight\_saving);

**Примечание:** Список часовых поясов приведен в конце данной страницы.

## mysql\_to\_unix()

Получает в качестве параметра время в формате MySQL Timestamp и возвращает его в формате Unix. Пример:

$mysql = '20061124092345';  
$unix = mysql\_to\_unix($mysql);

## unix\_to\_human()

Получает в качестве параметра время в формате Unix timestamp и возвращает его в понятном человеку формате используя следующий шаблон:

YYYY-MM-DD HH:MM:SS AM/PM

Это может быть полезным если вам необходимо отобразить дату в поле формы для ее просмотра.

Время может возвращаться с секундами или без них в Европейском формате или формате США. Если передано только время, оно будет возвращено без секунд в формате США. Примеры:

$now = time();  
echo unix\_to\_human($now); // Время в формате США без секунд  
echo unix\_to\_human($now, TRUE, 'us'); // Время в формате США с секундами  
echo unix\_to\_human($now, TRUE, 'eu'); // Время в Европейском формате с секундами

## human\_to\_unix()

Противоположна предыдущей функции. Получает в качестве параметра время в понятном человеку формате и возвращает его в формате Unix. Данная функция может быть полезной если вы обрабатываете понятные человеку даты, получаемые из формы. Возвращает FALSE в том случае, если дата на входе находится в формате, отличном от того, что был указан ранее. Пример:

$now = time();  
$human = unix\_to\_human($now);  
$unix = human\_to\_unix($human);

## timespan()

Возвращает unix timestamp в таком виде:

1 Year, 10 Months, 2 Weeks, 5 Days, 10 Hours, 16 Minutes

Первый параметр должен содержать Unix timestamp. Второй — время, большее, чем то, что было указано в первом параметре. Если второй параметр пуст, будет использовано текущее время. Самый распространенный сценарий использования данной функции — показать, сколько времени прошло начиная с некоторого момента до текущего. Пример:

$post\_date = '1079621429';  
$now = time();  
echo timespan($post\_date, $now);

**Примечание:** Текст, возвращаемый данной функцией находится в файле перевода: language/<ваш\_язык>/date\_lang.php

## days\_in\_month()

Возвращает количество дней в указанном месяце указанного года. Также учитывает високосный год. Пример:

echo days\_in\_month(06, 2005);

Если второй параметр пуст, используется текущий год.

## timezones()

Принимает в качестве параметра часовой пояс (список часовых поясов вы можете найти в соответствующем разделе далее) и возвращает разницу с всемирным временем(UTC) в часах.

echo timezones('UM5');

Данная функция может оказаться полезной при использовании вместе с timezone\_menu().

## timezone\_menu()

Возвращает выпадающий список часовых поясов:

Начало формы



Конец формы

Данное меню полезно, когда вы создаёте многопользовательский сайт, пользователи которого могут задать свой часовой пояс.

Первый параметр позволяет указать значение, выбранное по-умолчанию. К примеру, выставить тихоокеанское время в качестве значения по-умолчанию, вам необходимо будет использовать следующий код:

echo timezone\_menu('UM8');

Список часовых поясов для данного меню см. далее.

Второй параметр позволяет задать класс CSS для меню.

**Замечание:** Текст, содержащийся в меню может быть найден файле перевода: language/<ваш\_язык>/date\_lang.php

## Список часовых поясов

В следующей таблице отражены часовые пояса вместе с их местоположением.

|  |  |
| --- | --- |
| **Часовой пояс** | **Местоположение** |
| UM12 | (UTC - 12:00) Enitwetok, Kwajalien |
| UM11 | (UTC - 11:00) Nome, Midway Island, Samoa |
| UM10 | (UTC - 10:00) Hawaii |
| UM9 | (UTC - 9:00) Alaska |
| UM8 | (UTC - 8:00) Pacific Time |
| UM7 | (UTC - 7:00) Mountain Time |
| UM6 | (UTC - 6:00) Central Time, Mexico City |
| UM5 | (UTC - 5:00) Eastern Time, Bogota, Lima, Quito |
| UM4 | (UTC - 4:00) Atlantic Time, Caracas, La Paz |
| UM25 | (UTC - 3:30) Newfoundland |
| UM3 | (UTC - 3:00) Brazil, Buenos Aires, Georgetown, Falkland Is. |
| UM2 | (UTC - 2:00) Mid-Atlantic, Ascention Is., St Helena |
| UM1 | (UTC - 1:00) Azores, Cape Verde Islands |
| UTC | (UTC) Casablanca, Dublin, Edinburgh, London, Lisbon, Monrovia |
| UP1 | (UTC + 1:00) Berlin, Brussels, Copenhagen, Madrid, Paris, Rome |
| UP2 | (UTC + 2:00) Kaliningrad, South Africa, Warsaw |
| UP3 | (UTC + 3:00) Baghdad, Riyadh, Moscow, Nairobi |
| UP25 | (UTC + 3:30) Tehran |
| UP4 | (UTC + 4:00) Adu Dhabi, Baku, Muscat, Tbilisi |
| UP35 | (UTC + 4:30) Kabul |
| UP5 | (UTC + 5:00) Islamabad, Karachi, Tashkent |
| UP45 | (UTC + 5:30) Bombay, Calcutta, Madras, New Delhi |
| UP6 | (UTC + 6:00) Almaty, Colomba, Dhakra |
| UP7 | (UTC + 7:00) Bangkok, Hanoi, Jakarta |
| UP8 | (UTC + 8:00) Beijing, Hong Kong, Perth, Singapore, Taipei |
| UP9 | (UTC + 9:00) Osaka, Sapporo, Seoul, Tokyo, Yakutsk |
| UP85 | (UTC + 9:30) Adelaide, Darwin |
| UP10 | (UTC + 10:00) Melbourne, Papua New Guinea, Sydney, Vladivostok |
| UP11 | (UTC + 11:00) Magadan, New Caledonia, Solomon Islands |
| UP12 | (UTC + 12:00) Auckland, Wellington, Fiji, Marshall Island |

# Хелпер Directory

Хелпер Directory содержит функции, помогающие в работе с директориями.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## directory\_map('source directory')

Эта функция считывает содержимое директории, путь к которой указан в первом параметре, и строит массив с содержимым директории. Пример:

$map = directory\_map('./mydirectory/');

**Замечание:** Путь должен быть указан относительно основного файла index.php.

Поддиректории так же будут считаны. Если вы хотите прочитать только первую указанную папку, то укажите вторым параметром значение **true** (boolean):

$map = directory\_map('./mydirectory/', TRUE);

Каждое имя папки будет представлять из себя массив, в то время как содержимое этой папки будет значениями массива. Вот пример обычной работы функции:

Array  
(  
   [libraries] => Array  
   (  
       [0] => benchmark.html  
       [1] => config.html  
       [database] => Array  
       (  
             [0] => active\_record.html  
             [1] => binds.html  
             [2] => configuration.html  
             [3] => connecting.html  
             [4] => examples.html  
             [5] => fields.html  
             [6] => index.html  
             [7] => queries.html  
        )  
       [2] => email.html  
       [3] => file\_uploading.html  
       [4] => image\_lib.html  
       [5] => input.html  
       [6] => language.html  
       [7] => loader.html  
       [8] => pagination.html  
       [9] => uri.html  
)

# Хелпер Download

Хелпер Download позволяет вам загрузить информацию на ваш локальный компьютер.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## force\_download('filename', 'data')

Генерирует заголовки, заставляющие данные закачиваться на ваш компьютер. Превращая страницу в загрузочный файл. Первый параметр, это **имя, которым будет называться скачанный файл**, второй параметр — информация. Пример:

$data = 'Вот произвольный текст!';  
$name = 'mytext.txt';  
force\_download($name, $data);

Если вы хотите загрузить существующий на вашем сервере файл, то его необходимо считать в строку:

$data = file\_get\_contents("/path/to/photo.jpg"); // Считываем содержимое файла  
$name = 'myphoto.jpg';  
force\_download($name, $data);

# Хелпер Email

Хелпер содержит функции, помогающие работать с Email. Более надежное решение смотрите в [Классе Email](http://code-igniter.ru/user_guide/libraries/email.html).

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## valid\_email('email')

Проверяет email на соответствие формату email. Заметим, что это не гарантированная проверка адреса, это всего лишь соответствие синтаксису, но на самом деле такого email может не существовать.

Функция возвращает TRUE/FALSE

$this->load->helper(email);  
if (valid\_email('email@somesite.com'))  
{  
    echo 'email верен';  
}  
else  
{  
    echo 'email не верен';  
}

## send\_email('получатель', 'тема', 'сообщение')

Посылка email с использованием родной для PHP функции [mail()](http://www.php.net/function.mail). Для большей надежности смотрите [Класс Email](http://code-igniter.ru/user_guide/libraries/email.html).

# Хелпер File

Хелпер содержит функции, помогающие работать с файлами.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## read\_file('path')

Возвращает полное содержимое файла, расположенного по переданному адресу. Пример:

$string = read\_file('./путь/до/файла/file.php');

Путь может быть относительным или полным. Вернет FALSE (boolean) если файл не найден.

**Замечание:** Путь должен быть указан относительно положения главного файла index.php, а НЕ контроллера или отображения.

Если ваш сервер запущен с ограничением open\_basedir, то эта функция может не работать.

## write\_file('path', $data)

В файл, указанный в первом параметре записывается информация, указанная во второй. Если файл не существует, то функция создаст его. Пример:

$data = 'Любая информация';  
if ( ! write\_file('./path/to/file.php', $data))  
{  
     echo 'Не удалось записать данные в файл!';  
}  
else  
{  
     echo 'Файл записан!';  
}

Опционально вы можете указать режим записи в третьем параметре:

write\_file('./path/to/file.php', $data, **'r+'**);

По умолчанию указан режим **wb**. Пожалуйста, посмотрите [Пользовательское руководство по PHP](http://php.net/fopen) для того чтобы узнать о прочих режимах.

Замечание: Чтобы функция работала, на файл должны быть поставлены права на запись (666, 777 и т.п.). Если файл не существует, то для записи должна быть доступна директория.

**Замечание:** Путь должен быть указан относительно положения главного файла index.php, а НЕ контроллера или отображения.

## delete\_files('path')

Удаляет ВСЕ файлы по переданному пути. Пример:

delete\_files('./path/to/directory/');

Если второй параметр установлен в **true**, то любые папки, содержащиеся по этому пути, так же будут удалены. Пример:

delete\_files('./path/to/directory/', TRUE);

**Замечание:** Должна быть разрешена запись в эти файлы, чтобы их удалить.

## get\_filenames('path/to/directory/')

Принимает путь на сервере до директории и возвращает все файлы, содержащиеся в ней. Так же файлы можно вернуть с полным адресом до них, если передать вторым параметром TRUE.

# Хелпер Form

Хелпер Form содержит функции, помогающие в работе с формами.

## Загрузка хелпера

Для загрузки хелпера используется следующий код:

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

Доступны следующие функции:

## form\_open()

Создаёт открывающий тэг формы, используя базовый URL, **который берётся из файла конфигурации**. Также позволяет добавить в форму атрибуты и скрытые поля ввода.

Главное преимущество использования хелпера перед html-тэгом в том, что хелпер делает сайт более стабильным в случае переезда и связанного с ним изменения URL.

Вот простой пример:

echo form\_open('email/send');

Создаётся форма, которая будет указывать на ваш базовый URL плюс URI "email/send":

<form method="post" action="http:/www.your-site.com/index.php/email/send" />

#### Добавляем атрибуты

Атрибуты могут быть добавлены путём передачи ассоциативного массива вторым параметром:

$attributes = array('class' => 'email', 'id' => 'myform');  
echo form\_open('email/send', $attributes);

Приведённый выше пример создаст форму, подобную этой:

<form method="post" action="http:/www.your-site.com/index.php/email/send"  class="email"  id="myform" />

#### Добавляем скрытые поля ввода

Скрытые поля могут быть добавлены путём передачи ассоциативного массива третьим параметром:

$hidden = array('username' => 'Александр', 'member\_id' => '234');  
echo form\_open('email/send', '', $hidden);

Приведённый выше пример создаст форму, подобную этой:

<form method="post" action="http:/www.your-site.com/index.php/email/send"  class="email"  id="myform" />  
<input type="hidden" name="username" value="Александр" />  
<input type="hidden" name="member\_id" value="234" />

## form\_open\_multipart()

Данная функция абсолютно идентична **form\_open()**, рассмотренной выше, за исключением того, что она добавляет атрибут multipart, необходимый в случае, если вы хотите загружать файлы.

## form\_hidden()

Позволяет сгенерировать скрытые поля ввода. Для создания одного поля вы можете указать имя и значение:

form\_hidden('username', 'Александр');  
// Выдаст:  
<input type="hidden" name="username" value="Александр" />

Также вы можете создать несколько полей, передав ассоциативный массив:

$data = array(  
              'name'  => 'Александр',  
              'email' => 'alexander@example.com',  
              'url'   => 'http://www.example.com'  
            );  
echo form\_hidden($data);  
// Выдаст:  
<input type="hidden" name="name" value="Александр" />  
<input type="hidden" name="email" value="alexander@example.com" />  
<input type="hidden" name="url" value="http://www.example.com" />

## form\_input()

Позволяет сгенерировать стандартное текстовое однострочное поле ввода. Достаточно указать имя и значение первым и вторым параметром:

echo form\_input('username', 'Александр');

Также вы можете передать ассоциативный массив, содержащий любые данные, которые будут помещены в форму:

$data = array(  
              'name'        => 'username',  
              'id'          => 'username',  
              'value'       => 'Александр',  
              'maxlength'   => '100',  
              'size'        => '50',  
              'style'       => 'width:50%',  
            );  
echo form\_input($data);  
// Выдаст:  
<input type="text" name="username" id="username" value="Александр" maxlength="100" size="50" style="width:50%" />

Если вам требуется поместить в форму дополнительные данные, такие как JavaScript, вы можете передать их строкой третьим параметром:

$js = 'onClick="some\_function()"';  
echo form\_input('username', 'Александр', $js);

## form\_password()

Данная функция идентична **form\_input()**, описанной ранее, за исключением того, что она устанавливает тип поля в "password" (пароль).

## form\_upload()

Данная функция идентична **form\_input()**, описанной ранее, за исключением того, что она устанавливает тип поля в "file", позволяя тем самым загружать файлы.

## form\_textarea()

Данная функция идентична **form\_input()**, описанной ранее, за исключением того, что она устанавливает тип поля в "textarea". Замечание: Вместо атрибутов "maxlength" и "size" указываются "rows" и "cols".

## form\_dropdown()

Позволяет создать стандартный выпадающий список. Первым параметром задаётся имя поля, вторым — ассоциативный массив опций и третьим — выбранное значение. Также вы можете передать третьим параметром массив. В этом случае CodeIgniter создаст список с возможностью множественного выбора. Пример:

$options = array(  
                  'small'  => 'Маленькая уточка',  
                  'med'    => 'Средняя уточка',  
                  'large'   => 'Большая уточка',  
                  'xlarge' => 'Огромная утка!',  
                );  
$ducks\_on\_sale = array('small', 'large');  
echo form\_dropdown('ducks', $options, 'large');  
// Выдаст:  
<select name="ducks">  
<option value="small">Маленькая уточка</option>  
<option value="med">Средняя уточка</option>  
<option value="large" selected="selected">Большая уточка</option>  
<option value="xlarge">Огромная утка!</option>  
</select>  
echo form\_dropdown('ducks', $options, $ducks\_on\_sale);  
// Выдаст:  
<select name="shirts" multiple="multiple">  
<option value="small" selected="selected">Маленькая уточка</option>  
<option value="med">Средняя уточка</option>  
<option value="large" selected="selected">Большая уточка</option>  
<option value="xlarge">Огромная утка!</option>  
</select>

Если вы хотите, чтобы открывающий <select> содержал дополнительные данные, такие как JavaScript, вы можете передать их строкой четвёртым параметром:

$js = 'onChange="some\_function()"';  
echo form\_dropdown('ducks', $options, 'large', $js);

## form\_fieldset()

Создаёт поля fieldset/legend.

echo form\_fieldset('Мой адрес');  
echo "<p>Содержимое группы полей.</p>\n";  
echo form\_fieldset\_close();   
// Выдаст  
<fieldset>   
<legend>Мой адрес</legend>   
<p>Содержимое группы полей.</p>   
</fieldset>

Также, как и в остальных функциях, если вам необходимы дополнительные атрибуты, вы можете передать их вторым параметром в виде ассоциативного массива.

$attributes = array('id' => 'address\_info', 'class' => 'address\_info');  
echo form\_fieldset('Мой адрес', $attributes);  
echo "<p>Содержимое группы полей.</p>\n";  
echo form\_fieldset\_close();   
// Выдаст  
<fieldset id="address\_info" class="address\_info">   
<legend>Мой адрес</legend>   
<p>Содержимое группы полей.</p>   
</fieldset>

## form\_fieldset\_close()

Создаёт закрывающий тэг </fieldset>. Единственным преимуществом данной функции является то, что она позволяет вам передать данные, которые будут добавлены после тэга. К примеру:

$string = "</div></div>";  
echo fieldset\_close($string);  
// Выдаст:  
</fieldset>  
</div></div>

## form\_checkbox()

Позволяет создать поле типа checkbox. Простой пример:

echo form\_checkbox('newsletter', 'accept', TRUE);  
// Выдаст:  
<input type="checkbox" name="newsletter" value="accept" checked="checked" />

Третьим параметром можно передать TRUE или FALSE для того, чтобы указать, должен ли чекбокс быть отмечен.

Также, как и в остальных функциях данного хелпера, вы можете передать функции массив атрибутов:

$data = array(  
              'name'        => 'newsletter',  
              'id'          => 'newsletter',  
              'value'       => 'accept',  
              'checked'     => TRUE,  
              'style'       => 'margin:10px',  
            );  
echo form\_checkbox($data);  
// Выдаст:  
<input type="checkbox" name="newsletter" id="newsletter" value="accept" checked="checked" style="margin:10px" />

Также, как и в остальных функциях, если вам необходимо, чтобы тэг содержал дополнительные данные, такие как JavaScript, вы можете передать их строкой четвёртым параметром:

$js = 'onClick="some\_function()"';  
echo form\_checkbox('newsletter', 'accept', TRUE, $js)

## form\_radio()

Функция идентична **form\_checkbox()** за исключением того, что она присваивает полю тип "radio".

## form\_submit()

Создаёт стандартную кнопку для отправки формы. Простой пример:

echo form\_submit('mysubmit', 'Отправить!');  
// Выведет:  
<input type="submit" name="mysubmit" value="Отправить!" />

Также, как и в остальных функциях, вы можете передать первым параметром ассоциативный массив, если вам необходимо установить свои атрибуты. Третий параметр позволяет добавить в форму дополнительные данные, такие как JavaScript.

## form\_label()

Позволяет создать тэг <label>. Простой пример:

echo form\_label('Введите ваше имя', 'username');  
// Выведет:   
<label id="username">Введите ваше имя</label>

Также, как и в остальных функциях, вы можете передать третьим параметром ассоциативный массив, если вам необходимо установить свои атрибуты.

$attributes = array(  
'class' => 'mycustomclass',  
'style' => 'color: #000;',  
);  
echo form\_label('Введите ваше имя', 'username', $attributes);  
// Выведет:   
<label id="username" class="mycustomclass" style="color: #000;">Введите ваше имя</label>

## form\_reset()

Позволяет создать стандартную кнопку для сброса полей формы. Используется также, как и **form\_submit()**.

## form\_close()

Создаёт закрывающий тэг </form>. Единственным преимуществом данной функции является то, что она позволяет вам передать данные, которые будут добавлены ниже тэга. К примеру:

$string = "</div></div>";  
echo form\_close($string);  
// Выведет:  
</form>  
</div></div>

## form\_prep()

Позволяет вам безопасно использовать HTML и такие символы, как кавычки в элементах формы, не разрывая её. Пример:

$string = 'Строка содержит **"текст в кавычках"**.';  
<input type="text" name="myform" value="**$string**" />

Так как приведённая выше строка содержит кавычки, она вызовет разрыв формы. Функция form\_prep сконвертирует HTML таким образом, что этого не произойдёт:

<input type="text" name="myform" value="**<?php echo form\_prep($string); ?>**" />

**Примечание:** Если вы используете какие либо функции с этой страницы, значения формы будут обрабатываться автоматически. Повторно вызывать данную функцию в этом случае не нужно. Используйте её только в том случае, если создаёте свои элементы.

# Хелпер HTML

Хелпер HTML содержит функции, помогающие работать с HTML.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## br()

Генерирует тег окончания строки (<br />) указанное количество раз. Пример:

echo br(3);

Будет выведено: <br /><br /><br />

## heading()

Позволяет создавать HTML <h1> теги. Первый параметр содержит текст заголовка, а второй глубину вложенности. Пример:

echo heading('Приветствуем!', 3);

Будет сгенерировано: <h3>Приветствуем!</h3>

## img()

Позволяет вам создать тег <img />. Первый параметр содержит путь к картинке. Пример:

echo img('images/picture.jpg');  
// получим <img src="http://site.com/images/picture.jpg" />

Так же можно указать второй параметр TRUE/FALSE который укажет надо ли добавлять значение $config['index\_page'] в качестве префикса.

echo img('images/picture.jpg', TRUE);  
// получим <img src="http://site.com/index.php/images/picture.jpg" />

Кроме того в img() может быть передан ассоциативный массив, для детального указания всех параметров.

$image\_properties = array(  
          'src' => 'images/picture.jpg',  
          'alt' => 'Я демонстрирую как можно съесть 4 ломтика пицы за раз',  
          'class' => 'post\_images',  
          'width' => '200',  
          'height' => '200',  
          'title' => 'Это довольно глубокая ночь',  
          'rel' => 'lightbox',  
);  
img($image\_properties);  
// <img src="http://site.com/index.php/images/picture.jpg" alt="Я демонстрирую как можно съесть 4 ломтика пицы за раз" class="post\_images" width="200" height="200" title="Это довольно глубокая ночь" rel="lightbox" />

## link\_tag()

Позволяет создать тег <link />. Это необходимо для указания стилей и прочих ссылок. Обязателен первый параметр href, и опциональны параметры rel, type, title, media and index\_page. index\_page устанавливается в TRUE/FALSE чтобы сообщить о необходимости включения $config['index\_page'] в путь в качестве префикса.

echo link\_tag('css/mystyles.css');  
// gives <link href="http://site.com/css/mystyles.css" rel="stylesheet" type="text/css" />

Еще пример:

echo link\_tag('favicon.ico', 'shortcut icon', 'image/ico');  
// <link href="http://site.com/favicon.ico" rel="shortcut icon" type="image/ico" />   
echo link('feed', 'alternate', 'application/rss+xml', 'Мой RSS Feed');  
// <link href="http://site.com/feed" rel="alternate" type="application/rss+xml" title="Мой RSS Feed" />

Так же в функцию можно передать ассоциативный массив, чтобы точно контролировать все атрибуты и значения.

$link = array(  
          'href' => 'css/printer.css',  
          'rel' => 'stylesheet',  
          'type' => 'text/css',  
          'media' => 'print'  
);  
echo link\_tag($link);  
// <link href="http://site.com/css/printer.css" rel="stylesheet" type="text/css" media="print" />

## nbs()

Генерирует непереносимый пробел (&nbsp;) в указанном количестве. Пример:

echo nbs(3);

Из примера получим: &nbsp;&nbsp;&nbsp;

## ol()  and  ul()

Позволяет вам генерировать упорядоченные или неупорядоченные списки HTML из простых или многомерных массивов. Пример:

$this->load->helper('html');  
$list = array(  
            'red',   
            'blue',   
            'green',  
            'yellow'  
            );  
$attributes = array(  
                    'class' => 'boldlist',  
                    'id'    => 'mylist'  
                    );  
echo ul($list, $attributes);

Приведенный выше код выведет этот текст:

<ul class="boldlist" id="mylist">  
  <li>red</li>  
  <li>blue</li>  
  <li>green</li>  
  <li>yellow</li>  
</ul>

Вот более сложный пример с использованием многомерного массива:

$this->load->helper('html');  
$list = array(  
            'colors' => array(  
                                'red',  
                                'blue',  
                                'green'  
                            ),  
            'shapes' => array(  
                                'round',   
                                'square',  
                                'circles' => array(  
                                                    'ellipse',   
                                                    'oval',   
                                                    'sphere'  
                                                    )  
                            ),  
            'moods'    => array(  
                                'happy',   
                                'upset' => array(  
                                                    'defeated' => array(  
                                                                        'dejected',  
                                                                        'disheartened',  
                                                                        'depressed'  
                                                                        ),  
                                                    'annoyed',  
                                                    'cross',  
                                                    'angry'  
                                                )  
                            )  
            );  
echo ul($list);

Приведенный код выведет следующее:

<ul class="boldlist" id="mylist">   
  <li>colors  
    <ul>  
      <li>red</li>  
      <li>blue</li>  
      <li>green</li>  
    </ul>  
  </li>  
  <li>shapes  
    <ul>  
      <li>round</li>  
      <li>suare</li>  
      <li>circles  
        <ul>  
          <li>elipse</li>  
          <li>oval</li>  
          <li>sphere</li>  
        </ul>  
      </li>  
    </ul>  
  </li>  
  <li>moods  
    <ul>  
      <li>happy</li>  
      <li>upset  
        <ul>  
          <li>defeated  
            <ul>  
              <li>dejected</li>  
              <li>disheartened</li>  
              <li>depressed</li>  
            </ul>  
          </li>  
          <li>annoyed</li>  
          <li>cross</li>  
          <li>angry</li>  
        </ul>  
      </li>  
    </ul>  
  </li>  
</ul>

# Хелпер Inflector

Хелпер Inflector содержит функции позволяющие вам менять форму слова на множественное число, единственное, camel case и т.п.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## singular()

Меняет множественное число на единственное. Пример:

$word = "dogs";  
echo singular($word); // Вернет "dog"

## plural()

Меняет единственное число на множественное. Пример:

$word = "dog";  
echo plural($word); // Вернет "dogs"

Чтобы слово заканчивалось на «es» вторым аргументом напишите «true».

$word = "pass";  
echo plural($word, TRUE); // Вернет "passes"

## camelize()

Изменяет строку слов, разделенных пробелами на формат camel case. Пример:

$word = "my\_dog\_spot";  
echo camelize($word); // Вернет "myDogSpot"

## underscore()

Принимает слова, разделенные пробелами, вернет их через подчеркивание. Пример:

$word = "my dog spot";  
echo underscore($word); // Вернет "my\_dog\_spot"

## humanize()

Принимает строку со словами через подчеркивание, поменяет на пробел. Каждое слово будет с большой буквы. Пример:

$word = "my\_dog\_spot";  
echo humanize($word); // Вернет "My Dog Spot"

# Хелпер Language

Языковой хелпер содержит функции для работы с языковыми файлами.

## Загрузка хелпера

Хелпер загружается следующим кодом :

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

Доступные функции :

## lang('language line', 'element id')

Функция возвращает значение элемента языкового файла. Аналогична функции **$this->lang->line()**. Второй параметр позволяет возвращать строку как HTML-слой формы для заданного элемента. Например:

echo lang('**language\_key**', '**form\_item\_id**');  
// То же, что и <label for="form\_item\_id">language\_key</label>

# Хелпер Number

Хелпер Number помогает работать с численными типами данных.

## Загрузка хелпера

Хелпер загружается следующим кодом :

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

Станут доступны следующие функции :

## byte\_format()

Формирует строку на основе полученного числа как количества байтов. Так же добавляется суффикс. Примеры:

echo byte\_format(456); // Возвращает 456 Bytes  
echo byte\_format(4567); // Возвращает 4.5 KB  
echo byte\_format(45678); // Возвращает 44.8 KB  
echo byte\_format(456789); // Возвращает 447.8 KB  
echo byte\_format(3456789); // Возвращает 3.3 MB  
echo byte\_format(12345678912345); // Возвращает 1.8 GB  
echo byte\_format(123456789123456789); // Возвращает 11,228.3 TB

**Примечание:** эта функция содержит языковой файл, позволяющий перевести названия на другой язык. Файл расположен по адресу: language/ /number\_lang.php

# Хелпер Path

Хелпер Path содержит функции, помогающие работать с путями файлов на сервере.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## set\_realpath()

Проверяет существование пути. Эта функция вернет путь к файлу без символической ссылки или относительности к месту вызова. Опциональный второй параметр сгенерирует ошибку, если путь отсутствует.

$directory = './../../etc/passwd';  
echo set\_realpath($directory);  
// вернет "/etc/passwd"  
$non\_existent\_directory = './../../path/not/found';  
echo set\_realpath($non\_existent\_directory, TRUE);  
// вернет "/path/not/found"  
echo set\_realpath($non\_existent\_directory, FALSE);  
// вернет сообщение **об ошибке** о том, что файл не найден

# Хелпер Security

Хелпер Security содержит функции связанные с безопасностью.

## Загрузка этого хелпера

Хелпер загружается, используя следующий код:

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

Хелпер содержит следующие функции:

## xss\_clean()

Обеспечивает фильтрацию на Cross Site Script. Эта функция является псевдонимом одноименной функции в [классе Input](http://code-igniter.ru/user_guide/libraries/input.html). Больше информации об этом можно найти там.

## dohash()

Разрешает вам создавать SHA1 или MD5 для хеширования паролей. Функция работает с SHA1 по умолчанию. Примеры:

$str = dohash($str); // SHA1  
$str = dohash($str, 'md5'); // MD5

## strip\_image\_tags()

Функция занимается вырезанием тегов изображений из входящей строки. Она оставляет URL изображения как обычный текст.

$string = strip\_image\_tags($string);

## encode\_php\_tags()

Это функция безопасности, которая конвертирует PHP теги в сущности. Замечание: Если вы используете функцию для фильтрации XSS, то она это сделает автоматически.

$string = encode\_php\_tags($string);

# Хелпер Smiley

Хелпер Smiley содержит функции, помогающие в работе со смайлами.

## Загрузка этого хелпера

Хелпер загружается, используя следующий код:

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

## Обзор

Хелпер Smiley принимает текст с типогрфическими смайлами, вроде **:-)** и превращает их в картинки, например smile!

Также он будет отображать ряд смайликов, которые будут вставляться в текст при нажатии на них. Например, если вы ведете блог, в котором пользователи могут комментировать записи, то можете расположить форму добавления смайликов рядом с формой комментирования. Пользователи могут нажимать на понравившиеся смайлики и они будут добавлены в поле комментария при помощи JavaScript.

## Руководство по кликабельным смайликам

Вот пример, показывающий, как можно поставить ряд смайликом рядом с формой добавления комментария. Этот пример требует чтобы вы сначала загрузили и установили пакет смайликов, а затем создали контроллер и отбражение, как это описано.

**Важно:** Прежде чем вы начнете, пожалуйста [загрузите пакет изображений](http://codeigniter.com/download_files/smileys.zip) и положите в доступное место на вашем сервере. Так же хелпер полагает, что у вас есть массив замены букв на изображения в файле**application/config/smileys.php**.

### Контроллер

В папке **application/controllers/** создайте файл с названием **smileys.php** и поместите в него код, приведенный ниже.

**Важно:** Измените URL в функции **get\_clickable\_smileys()** ниже, чтобы он указывал на папку **smiley**.

Вы заметите, что в дополнение к хелперу со смайликами мы используем [класс Table](http://code-igniter.ru/user_guide/libraries/table.html).

<?php

class Smileys extends Controller {

function Smileys()

{

parent::Controller();

}

function index()

{

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

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

$image\_array = get\_clickable\_smileys('http://www.your-site.com/images/smileys/');

$col\_array = $this->table->make\_columns($image\_array, 8); $data['smiley\_table'] = $this->table->generate($col\_array);

$this->load->view('smiley\_view', $data);

}

}

?>

В вашей папке **application/views/** создайте файл **smiley\_view.php** и поместите в него следующий код:

<html>

<head>

<title>Smileys</title>

<?php echo js\_insert\_smiley('blog', 'comments'); ?>

</head>

<body>

<form name="blog">

<textarea name="comments" cols="40" rows="4"></textarea>

</form>

<p>Click to insert a smiley!</p>

<?php echo $smiley\_table; ?>

</body>

</html>

Когда вы создали отображение и контроллер и загрузили их, посетите [**http://www.your-site.com/index.php/smileys/**](http://www.your-site.com/index.php/smileys/)

# Справочник функций

## get\_clickable\_smileys()

Возвращает массив, содержащий соответствие смайлика и кликабельной ссылки на него. Вы должны передать URL папки со смайликами первым параметром:

$image\_array = get\_clickable\_smileys("http://www.your-site.com/images/smileys/");

## js\_insert\_smiley()

Генерирует JavaScript-код, который позволяет нажать на изображения, чтобы вставить их в форму. Первый параметр должен содержать название вашей формы, второй — название поля формы. Эта функция должна вызываться в области <head> вашей веб-страницы.

<?php echo js\_insert\_smiley('blog', 'comments'); ?>

## parse\_smileys()

Принимает строку текста и заменяет в ней все текстовые смайлы на их графические эквиваленты. В первом параметре должен быть текст, во втором ссылка на папку ваших смайликов:

$str = 'Вот несколько смайликов: :-) ;-)'; $str = parse\_smileys($str, "http://www.your-site.com/images/smileys/"); echo $str;

# Хелпер String

Хелпер String содержит функции, помогающие в работе со строками.

## Загрузка этого хелпера

Хелпер загружается, используя следующий код:

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

Хелпер содержит следующие функции:

## random\_string()

Генерирует произвольную строку, в зависимости от указанного типа и длины. Функция полезна для создания паролей и случайных хешей.

Первый параметр определяет тип строки, второй ее длину. Доступны следующие варианты:

* **alnum**:  Буквенно-цифровая строка с буквами в верхнем и нижнем регистрах.
* **numeric**:  Цифровая строка.
* **nozero**:  Цифровая строка без нуля.
* **unique**:  Хеш MD5 и uniqid(). Замечание: Параметр длины не доступен для этого типа. Вернет строку длиной 32 символа.

Пример использования:

echo random\_string('alnum', 16);

## alternator()

Принимает от двух и более параметров и возвращает их поочередно. Пример:

for ($i = 0; $i < 10; $i++)  
{  
    echo alternator('string one', 'string two');  
}

Вы можете добавить любое число параметров и с каждым разом будет возвращаться следующий за предыдущим.

for ($i = 0; $i < 10; $i++)  
{  
    echo alternator('one', 'two', 'three', 'four', 'five');  
}

**Замечание:** Для повторного вызова этой функции вызовите ее дополнительно без аргументов, чтобы повторно инициализировать.

## repeater()

Генерирует строку, повторяя первый параметр указанное количество раз. Пример:

$string = "\n";  
echo repeater($string, 30);

Приведенный выше пример вернет 30 символов новой строки.

## reduce\_double\_slashes()

Преобразует двойные слеши в строке в одну косую черту, за исключением http://. Пример:

$string = "http://www.example.com//index.php";  
echo reduce\_double\_slashes($string); // результатом будет "http://www.example.com/index.php"

## trim\_slashes()

Удаляет любые начальные/конечные косые черты из строки. Пример:  
$string = "/this/that/theother/";  
echo trim\_slashes($string); // результатом будет this/that/theother

## reduce\_multiples()

Удаляет использование нескольких подряд указанных символов. Пример:

$string = "Fred, Bill,, Joe, Jimmy";  
$string = reduce\_multiples($string,","); // результатом будет "Fred, Bill, Joe, Jimmy"

Функция принимает следующие параметры:

reduce\_multiples(string: текст, string: символ вхождения, boolean: удалять ли найденный символ и из конца строки)

Первый параметр содержит строку, в которой необходимо сократить множество повторений. Второй параметр — это символ для удаления. Третий установлен в False по умолчанию. Если установить True, то вхождения будут удалены и из начала и конца строки. Пример:

$string=",Fred, Bill,, Joe, Jimmy,";  
$string=reduce\_multiples($string,",",true); // результатом будет "Fred, Bill, Joe, Jimmy"

## quotes\_to\_entities()

Конвертирует одинарные и двойные кавычки в HTML сущности. Пример:

$string="Joe's \"dinner\"";  
$string=quotes\_to\_entities($string); // результатом будет "Joe&#39;s &quot;dinner&quot;"

## strip\_quotes()

Удаляет двойные и одинарные кавычки из строки. Пример:

$string="Joe's \"dinner\"";  
$string=strip\_quotes($string); // результатом будет "Joes dinner"

# Хелпер Text

Хелпер Text содержит функции, помогающие в работе с текстом.

## Загрузка этого хелпера

Загрузить хелпер можно следующим образом:

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

Хелпер содержит следующие функции:

## word\_limiter()

Сокращает строку до количества указанных **слов**. Например:

$string = "Это прекрасная строка текста из семи слов.";  
$string = word\_limiter($string, 4);  
// Вернет: Это прекрасная строка текста…

Третьим параметром можно передать суффикс, который добавится после обрезанной строки. По умолчанию добавляется многоточие.

## character\_limiter()

Округляет строку до количества указанных **символов**. Функция сохраняет слова целыми, поэтому результирующая строка может быть чуть большей или меньшей, чем вы просили. Пример:

$string = "Это прекрасная строка текста из семи слов.";  
$string = character\_limiter($string, 20);  
// Вернет: Это прекрасная строка…

Третьим параметром можно передать суффикс, который добавится после обрезанной строки. По умолчанию добавляется многоточие.

## ascii\_to\_entities()

Конвертирует значение ASCII в символьные сущности, работая с символами за пределами ASCII и символами MS Word которые могут вызвать проблемы при использовании на веб-странице, так что они могут быть безопасно показаны на странице или сохранены в базу данных. Существует определенная зависимость между настройками сервера и поддерживаемыми символами, поэтому на эту функцию нельзя полагаться на 100%. Тем не менее основная часть символов за пределами стандартного диапазона должна определяться корректно. Пример:

$string = ascii\_to\_entities($string);

## entities\_to\_ascii()

Эта функция работает обратно предыдущей; она преобразует символы обратно в ASCII.

## word\_censor()

Включает цензор слов во входящей строке. Первым параметром принимает оригинальную строку. Второй параметр должен содержать массив слов, которые должны быть запрещены. Третий (опциональный) параметр может содержать строку, которой будут заменены найденные слова. Если он не указан, то будет происходить замена на следующие символы: ####. Пример:

$disallowed = array('проклятье', 'черт', 'черт побери!', 'дьявол');  
$string = word\_censor($string, $disallowed, '<пип>');

## highlight\_code()

Функция подсвечивает синтаксис в строке кода (PHP, HTML и т.д.). Пример:

$string = highlight\_code($string);

Используется функция PHP highlight\_string(), поэтому используемые цвета необходимо изменять в файле php.ini.

## highlight\_phrase()

Выделяет подстроку в строке, окружая ее указанными символами. Первый параметр должен содержать оригинальную строку, вторым параметром необходимо указать строку для подсветки. Третий и четвертый параметры должны содержать открывающий/закрывающий HTML теги, которыми вы хотите выделить подстроку. Пример:

$string = "Это прекрасная строка текста из семи слов.";  
$string = highlight\_phrase($string, "прекрасная строка", '<span style="color:#990000">', '</span>');

Вернется преобразованная строка:

Это прекрасная строка текста из семи слов.

## word\_wrap()

Разбиение текста на строки по указанному количеству **символов**. Пример:

$string = "Это простая строка, которая продемонстрирует работу описанной функции";  
echo word\_wrap($string, 25);  
// Would produce:  
Это простая строка,  
которая продемонстрирует  
работу описанной функции

# Хелпер Typography

Хелпер Typography содержит функции, помогающие форматировать текст для приведения его к семантически корректному виду.

## Загрузка хелпера

Для загрузки хелпера используется следующий код:

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

Доступны следующие функции:

## auto\_typography()

Форматирует текст таким образом, чтобы на выходе получился корректный с точки зрения семантики и типографики HTML. Принимает строку в качестве параметра и возвращает её со следующими изменениями:

* Окружает параграфы тэгами <p></p> (за параграф считается двойной перевод строки).
* Одиночные переводы строки преобразуются в <br />, если они не находятся внутри тэгов <pre>.
* Блочные элементы, такие как <div>, не обрамляются параграфами. Текст внутри блочных элементов обрамляется, если содержит переносы строк.
* Кавычки вне тэгов преобразуются в корректные для английского текста кавычки–лапки.
* Апострофы преобразуются в апострофы–лапки.
* Двойной минус (вроде -- этого или вот--этого) преобразуются в —тире.
* Три точки подряд предшествующие или следующие за словом преобразуются в троеточие…
* Двойные пробелы, следующие после существительного преобразуется в неразрывные пробелы для того, чтобы отображался именно двойной пробел.

Пример использования:

$string = auto\_typography($string);

**Примечание:** Типографское форматирование может загружать процессор, особенно при форматировании большого объёма текста. Если вы хотите использовать данную функцию, возможно вам будет полезно использовать [кэширование](http://code-igniter.ru/user_guide/general/caching.html)страниц.

## nl2br\_except\_pre()

Преобразует переводы строк вне тэгов <pre> в тэги <br />. Данная функция идентична встроенной в PHP **nl2br()** за исключением того, что она игнорирует пробелы в тэгах <pre>.

Пример использования:

$string = nl2br\_except\_pre($string);

# Хелпер URL

Хелпер URL содержит функции, помогающие работать с URL.

## Загрузка хелпера

Для загрузки хелпера используется следующий код:

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

Доступны следующие функции:

## site\_url()

Возвращает URL вашего сайта, указанный в файле конфигурации. Файл index.php (или тот, который вы выставили в**index\_page** в файле конфигурации) будет добавлен к URL также, как и любые URI сегменты, которые вы передаете в функцию.

Данная функция рекомендуется к использованию в том случае, если необходимо сгенерировать локальный URL для улучшения переносимости страниц в процессе изменения URL.

Сегменты могут быть переданы в функцию либо как строка, либо как массив. Пример строки:

echo site\_url("news/local/123");

Приведенный в примере код выведет нечто вроде: http://www.your-site.com/index.php/news/local/123

Пример передачи сегментов массивом:

$segments = array('news', 'local', '123');  
echo site\_url($segments);

## base\_url()

Возвращает корневой для сайта URL, указанный в файле конфигурации. Пример:

echo base\_url();

## index\_page()

Возвращает адрес индексной страницы сайта, указанной в файле конфигурации. Пример:

echo index\_page();

## anchor()

Создаёт стандартную HTML–ссылку на основе локального URL сайта:

<a href="http://www.your-site.com">Щёлк меня!</a>

Тэгу можно задать три необязательных параметра:

anchor(**uri segments**, **text**, **attributes**)

Первый параметр может содержать любые сегменты, которые вы хотите добавить к URL. Также, как и в функции**site\_url()**, описанной выше, сегменты можно указать строкой или массивом.

**Примечание:**  Если вы создаете ссылки, являющиеся для вашего приложения внутренними, не указывайте базовый адрес (http://...). Он будет считан из файла конфигурации и добавлен автоматически. Включайте только сегменты URI, которые вы хотите добавить к URL.

Второй аргумент — текст, который вы хотите отображать. Если его оставить пустым, будет использован URL ссылки.

Третий параметр может содержать список атрибутов, которые вы хотите добавить к ссылке. Можно указать их строкой, либо при помощи ассоциативного массива.

Пара примеров:

echo anchor('news/local/123', 'Мои новости');

Выведет: <a href="http://www.your-site.com/index.php/news/local/123" title="Мои новости">Мои новости</a>

echo anchor('news/local/123', 'Мои новости', array('title' => 'Самые лучшие новости!'));

Выведет: <a href="http://www.your-site.com/index.php/news/local/123" title="Самые лучшие новости!">Мои новости</a>

## anchor\_popup()

Почти идентична функции **anchor()** за тем исключением, что URL открывается в новом окне. Вы можете указать JavaScript–атрибуты окна третьим параметром для того, чтобы задать, как именно оно откроется. Если третий параметр не задан, будет открыто новое окно с установками, заданными в вашем браузере. Пример задания аттрибутов:

$atts = array(  
              'width'      => '800',  
              'height'     => '600',  
              'scrollbars' => 'yes',  
              'status'     => 'yes',  
              'resizable'  => 'yes',  
              'screenx'    => '0',  
              'screeny'    => '0'  
            );  
echo anchor\_popup(news/local/123, 'Щёлк меня!', $atts);

Примечание: Приведённые атрибуты являются атрибутами по умолчанию для данной функции, поэтому вам необходимо переопределить только те, которые от них отличаются. Если вы хотите, чтобы использовались значения по умолчанию — передайте в функицю пустой массив третьим параметром:

echo anchor\_popup('news/local/123', 'Щёлк меня!', array());

## mailto()

Создает стандартную почтовую HTML–ссылку. Пример использования:

echo mailto('me@my-site.com', 'Нажми на меня для отправки письма!');

Также, как и в функции **anchor()** вы можете задать атрибуты, используя третий параметр.

## safe\_mailto()

Идентична предыдущей функции за исключением того, что почтовый адрес выводится в обфусцированном виде с использованием порядковых номеров символов, которые генерируются при помощи JavaScript. Таким образом адрес скрывается от спам–ботов.

## auto\_link()

Автоматически преобразует в ссылки адреса URL и email, содержащиеся в тексте. Пример:

$string = auto\_link($string);

Второй параметр определяет, будут ли определяться оба адреса: URL и email или же только один из них. Если параметр не указан – определяются оба.

Определяет и преобразует только URL:

$string = auto\_link($string, 'url');

Определяет и преобразует только Email:

$string = auto\_link($string, 'email');

Третий параметр определяет открывать ли ссылки в новом окне. Значение может быть TRUE или FALSE:

$string = auto\_link($string, 'both', TRUE);

## url\_title()

Получает в качестве параметра строку и отдаёт человекопонятный URL. Очень полезно, если, допустим, у вас есть блог, в котором вы бы хотели использовать название заметок в качестве URL. Пример:

$title = "What's wrong with CSS?";  
$url\_title = url\_title($title);  
// Получим: whats-wrong-with-css

Второй параметр задаёт разделитель между словами. По–умолчанию используются минусы. Возможные варианты:**dash**(минус), **underscore**(подчёркивание):

$title = "What's wrong with CSS?";  
$url\_title = url\_title($title, 'underscore');  
// Получим: whats\_wrong\_with\_css

### prep\_url()

Данная функция добавляет **http://** в том случае, если его в URL нет. Передавайте функции URL, похожий на этот:

$url = "www.example.com";  
$url = prep\_url($url);

## redirect()

Реализует перенаправление к локальному URI с помощью заголовков. Также, как и все остальные функции данного хелпера, данная функция производит перенаправление к локальному URL сайта. Вам **не нужно** указывать полный URL сайта. Вместо него используются сегменты URI контроллера, к которому вы хотите обратиться. Функция создаёт URL на основе значений, указанных в файле конфигурации.

Второй параметр позволяет вам выбрать между методом "location" (по–умолчанию) и методом "refresh". Метод location быстрее, но он иногда вызывает проблемы на серверах Windows. Пример:

if ($logged\_in == FALSE)  
{  
     redirect('/login/form/', 'refresh');  
}

**Примечание:** Для корректной работы данной функции она должна быть использована до любого вывода в браузер так как она использует заголовки сервера.

# Хелпер XML

Файл XML Helper содержит функции, помогающие работать с информацией в формате XML.

## Загрузка хелпера

Хелпер загружается при помощи следующего кода:

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

Доступны следующие функции:

## xml\_convert('string')

Принимает строку и преобразует в ней зарезервированные XML-символы в их сущности:

Амперсанд: &  
Знаки больше и меньше: < >  
Одинарная и двойная кавычки: '  "  
Дефис: -

Эта функция игнорирует амперсанд, если он является частью существующих сущностей. Пример:

$string = xml\_convert($string);

**Глава 5. Дополнительная информация**

# Общая схема фреймворка

Чтобы получить PDF версию диаграммы, [нажмите сюда](http://codeigniter.com/download_files/ci_quick_ref.pdf).

